U N I V E R S I D A D   D E   M U R C I A

DEPARTAMENTO DE INGENIERÍA Y TECNOLOGÍA DE COMPUTADORES

# Optimización Automática de Software Paralelo de Álgebra Lineal

Tesis propuesta para la obtención del Grado de Doctor en Informática

Presentada por:
Antonio Javier Cuenca Muñoz

Dirigida por:
Domingo Giménez Cánovas
José González González

Murcia, Marzo de 2004

*A Ana, por estar siempre a mi lado*

# Resumen

En esta memoria se describe un proyecto de desarrollo de software paralelo de álgebra lineal con capacidad de adaptarse automáticamente a las condiciones de su entorno con el objetivo de optimizar sus prestaciones. Se aplican distintos tipos de ajustes automáticos: número de procesadores a utilizar, topología lógica de estos procesadores, tamaño del bloque de cálculo, distribución del trabajo a realizar entre los procesadores, selección de la mejor librería en cada momento de entre las disponibles (polilibrerías) y elección del mejor algoritmo con el que resolver un problema de entre varios equivalentes (polialgoritmos). La metodología planteada es válida para distintos tipos de plataformas paralelas que se pueden programar, con un alto grado de eficiencia, mediante el paradigma de paso de mensajes.

El problema del ajuste automático se ha abordado desde un punto de vista unificado, a partir del modelo analítico del tiempo de ejecución de cada rutina. Este modelo cuenta con una estructura teórica que define el comportamiento general de la rutina, sobre la que se introducen, en forma de parámetros del sistema, las características de la plataforma, de su software básico y de sus condiciones de carga de trabajo en cada momento.

En plataformas donde la carga de trabajo no sufre importantes variaciones, el ajuste de la rutina se realiza en la fase de instalación. En el caso de plataformas donde la carga de trabajo oscila notablemente se ha ideado un proceso de ajuste en dos fases. Durante la instalación de la rutina se recogen las características estáticas de la plataforma para, más tarde, en el momento de la ejecución, recoger los datos sobre la carga del sistema y realizar un ajuste de los datos iniciales en función de esta carga. Los resultados alcanzados muestran cómo las rutinas se adaptan perfectamente a las características de la plataforma.

El sistema software propuesto se integra en la estructura jerárquica de librerías de software de álgebra lineal clásica, de manera que para generar la información de las rutinas pertenecientes a librerías de niveles superiores se reaprovechan los datos obtenidos por las rutinas de los niveles inferiores, siguiendo el mismo patrón de reutilización existente de manera implícita en esta jerarquía de librerías.

# Agradecimientos

En primer lugar, quisiera agradecer a todas las personas e instituciones que han colaborado directamente en la realización de esta tesis:

- a los directores de esta tesis. A Domingo, por la dedicación y el entusiasmo que ha depositado en este trabajo. Durante todo su desarrollo, desde sus primeros pasos hasta su culminación, él ha sido un excelente director y un incansable compañero de trabajo. A Pepe, por sus sabios consejos, por su especial punto de vista sobre las cuestiones tratadas y por todo el tiempo que ha que sacado, no sé de dónde, para dedicárselo a este trabajo.

- a otros investigadores con los que hemos colaborado en diferentes fases de este trabajo y que nos han aportado multitud de ideas interesantes. Entre ellos destacaría a Miguel Valero-García de la UPC y a Jack Dongarra y Ken Roche de la Universidad de Tennessee.

- a Luis Pedro García, Pedro Alberti, Pedro Alonso y Antonio Vidal, por habernos cedido su software para la realización de algunos de los experimentos mostrado en esta tesis.

Y ahora, desde un plano más personal me gustaría dar las gracias:

- a mis familiares. A mis padres por trabajar doce horas al día, seis días a la semana, para poder pagarnos los estudios a mis hermanos y a mí. A mis hermanos, Santi y Paco, por mostrarme el camino académico a seguir (como buenos hermanos mayores). A mi tío Angel, por enseñarme que se puede ser Doctor sin tener que estudiar medicina.

- a todos los profesores que he tenido a lo largo de las diferentes etapas académicas por las que he pasado. Todos y cada uno de ellos pusieron su granito de arena en mi formación académica y personal.

- a mis compañeros de departamento (Pilar, Pedro, Lorenzo, Juan-Pe, Antonio, Juan Luis, Manolo, Grego, Oscar, Juan-Pi, Diego, Pepe, Paco, ...) con los que he trabajado, discutido, reído y tomado muchos litros de sucedáneo de café en estos últimos años. Especial mención a Pilar que, no sé cómo, me ha soportado seis años como compañero de despacho.

- a Ana, por todo el apoyo que me ha dado y la paciencia que ha tenido durante la escritura de esta tesis.

- y al resto de personas que me han ayudado en alguna medida a realizar este trabajo y cuyos nombres no recuerdo en este momento.

GRACIAS A TODOS.

# Índice General

# Índice de Figuras

# Índice de Tablas

# Capítulo 1. Introducción

## 1.1 Introducción

A lo largo de los últimos años, diferentes especialidades de la ciencia (química, biología, aeronáutica, ...) han incrementado notablemente la necesidad de contar con plataformas de altas prestaciones. Esta situación ha conllevado el desarrollo de una nueva disciplina llamada computación científica, cuyo objetivo general consiste en el desarrollo, mejora y mantenimiento de software de cálculo matemático para la resolución de los problemas planteados por científicos e ingenieros. Por lo tanto, los avances en el campo de la computación científica influyen de manera directa en la evolución de multitud de disciplinas científicas, que pueden abordar de esta manera la resolución de problemas de mayores dimensiones, con una mayor precisión en los resultados.

En los últimos años han surgido una serie de propuestas (ATLAS [WPD01], LFC [CDL+03], FFTW [FK01], ILIB [KKK01], ...) cuyo objetivo principal es dotar al software de computación científica de una cierta capacidad de modificarse a sí mismo de manera automática, de cara a adaptarse lo mejor posible a las condiciones del entorno donde se esté utilizando. Varios factores han motivado la aparición de este nuevo campo de investigación:

- Los usuarios habituales del software de computación científica necesitan contar con una alta capacidad de cómputo para desarrollar su trabajo, por lo que suelen disponer de acceso a potentes plataformas hardware, que habitualmente son de naturaleza paralela. De cara a obtener buenas prestaciones con este software en este tipo de plataformas se necesita tomar una serie de decisiones importantes a la hora de ejecutar cada rutina [Wu99]. Algunas de estas decisiones serían: cuántos procesadores utilizar de entre todos los disponibles; qué topología lógica escoger para estos procesadores, de cara a minimizar las comunicaciones entre ellos maximizando su utilización; cómo distribuir los datos entre ellos, para conseguir equilibrar sus cargas de trabajo; qué tipo de agrupaciones de datos utilizar para realizar los cálculos, de cara a minimizar los accesos a memoria; qué librería básica utilizar para cada una de las operaciones elementales de las rutinas; qué algoritmo seleccionar entre varios equivalentes para resolver cada problema; etc. Frente a esta serie de cuestiones los usuarios, que no suelen contar con conocimientos de arquitectura de computadores, se ven obligados a tomar decisiones aleatorias, o bien, a seleccionar las opciones que vienen por defecto en el software de que disponen.

- El desarrollo de hardware de altas prestaciones se sigue manteniendo fiel a la ley de Moore [Moo65], duplicando la velocidad de cómputo cada año y medio, con lo que, teóricamente, aumenta la capacidad de cálculo de los usuarios. Sin embargo, la realidad indica que, con frecuencia, esta continua evolución del hardware obliga a una continua adaptación del software existente de cara a acercarse a las prestaciones máximas que ofrecen las nuevas plataformas [RD02].

- Para estas nuevas plataformas, el desarrollo de código altamente eficiente y la adaptación del ya existente consume una gran cantidad de recursos humanos y técnicos. Además, debido a la naturaleza de este software, los desarrolladores tienen que ser programadores expertos, con amplios conocimientos tanto de análisis numérico como de arquitectura de computadores, de cara a conseguir una buena compenetración entre el software que implementan y el hardware destino [GH01].

- Las capacidades de cómputo de los sistemas cambian frecuentemente, bien de manera estática, mediante modificaciones en sus arquitecturas o cambios en el software básico instalado, o bien de manera dinámica, como consecuencia de altibajos en la carga de trabajo que soportan. Esta variabilidad en el sistema influye notablemente en las prestaciones del software de computación científica, a no ser que éste cuente con capacidad de adaptarse a las nuevas situaciones que se encuentre.

En esta tesis nos hemos centrado en la optimización automática de software de álgebra lineal densa[+] para plataformas paralelas genéricas. Se ha propuesto la aplicación a este software de distintos tipos de ajuste automático: elección apropiada de una serie de parámetros ajustables como son el número de procesadores a utilizar, la topología lógica de éstos, el tamaño del bloque de cálculo, distribución del trabajo a realizar entre los procesadores de la plataforma, selección de la librería básica de donde tomar la rutina para llevar a cabo cada operación (polilibrerías) y elección del mejor algoritmo con el que resolver un problema de entre varios equivalentes (polialgoritmos). En un principio, estos tipos de ajustes del software podrían parecer de naturaleza bien distinta, sin embargo, se han abordado desde una manera unificada, utilizando un modelo analítico del tiempo de ejecución de cada rutina como pilar básico de la metodología propuesta. De esta manera se consigue que el método de ajuste automático realice una búsqueda global en todo el espacio de posibles optimizaciones. Además, se plantea un sistema de trabajo cooperativo entre las diferentes rutinas y librerías a fin de minimizar el tiempo de ajuste de todas ellas.

[+] El software de álgebra linear es el núcleo sobre el que se construye, hoy en día, la mayor parte del software utilizado en el campo de la computación científica [DDS+98].

## 1.2 Visión general del sistema propuesto

El punto de partida de esta tesis lo constituye un modelo propuesto para sistemas paralelos de computación numérica con $P$ procesadores de similares características físicas, conectados entre sí de manera homogénea. Este modelo está constituido por un conjunto de parámetros que caracterizan al sistema, divididos en dos grandes grupos que configuran el submodelo de cómputo y el submodelo de comunicaciones. En el submodelo de cómputo, cada parámetro representará el tiempo precisado por un procesador para llevar a cabo una operación aritmética concreta, utilizando una librería determinada. En el submodelo de comunicaciones se recogerán las capacidades de intercomunicación de los distintos nodos de la plataforma, en función de la capacidad física de la red y del tamaño de los mensajes.

A partir del modelo del sistema, se plantea un modelo analítico del tiempo de ejecución de cada rutina. Este modelo de la rutina contiene dos conjuntos de parámetros. Por un lado, tenemos los parámetros del sistema que determinan la influencia de la plataforma en el tiempo de ejecución de la rutina y, por otro, tenemos un juego de parámetros algorítmicos, cuyos valores se eligen de manera automática de cara a minimizar el tiempo de ejecución de dicha rutina. Este modelo analítico tiene una naturaleza teórico-experimental, partiendo de un diseño inicial en base a la complejidad del algoritmo, al que se le confiere una naturaleza más realista mediante el cálculo experimental del valor de los parámetros del sistema que en él aparecen. Además, a este modelo se le dota de una naturaleza dinámica gracias a que los parámetros del sistema pueden ajustarse teniendo en cuenta la carga de trabajo que soporta la plataforma en cada momento. Consecuentemente, el modelo mantiene en todo momento una imagen actual del comportamiento de la rutina sobre la plataforma, permitiendo tomar las oportunas decisiones de optimización.

Utilizando este modelo analítico como eje central se diseña un sistema software que engloba a cada rutina, infiriéndole capacidad de ajuste automático. El ciclo de vida de esta rutina consta de tres fases bien diferenciadas (Figura 1.1):

- En la fase de diseño se implementa la rutina si es de nueva creación y, a continuación, se traza su modelo analítico del tiempo de ejecución y se programa el gestor de toda la información de optimización que acompañará a esta rutina.

- En la fase de instalación se realiza la recogida de información acerca de cada uno de los parámetros del sistema, mediante experimentación directa, si la rutina pertenece a los niveles inferiores de la jerarquía de librerías de álgebra lineal, o bien solicitando esta información a las rutinas de niveles inferiores que invoca en su código.

- En la fase de ejecución, en primer lugar, se obtiene información sobre el estado de carga de trabajo de la plataforma, a continuación se ajustan los parámetros del sistema de acuerdo a esa información, tras ello, se utiliza el modelo para escoger los valores de los parámetros ajustables y, finalmente, se invoca la rutina original.

Un ejemplo de la mejora que se consigue en las prestaciones de las rutinas gracias a la aplicación de nuestra metodología se muestra en la Figura 1.2 para la rutina paralela de factorización QR de la librería ScaLAPACK [BCC+97][wScaLA]. Los datos mostrados se han obtenido tras ejecutar esta rutina para distintos tamaños de problema, probando para cada tamaño con diferentes combinaciones de parámetros algorítmicos. Por un lado tenemos el tiempo de ejecución óptimo, que únicamente puede conocerse a posteriori tras todas estas ejecuciones; el tiempo de ejecución promedio, que representa la elección que un usuario no experto podría haber hecho de los diferentes parámetros ajustables de la rutina; y el tiempo que se habría obtenido con los parámetros elegidos automáticamente por nuestra rutina. Como se puede apreciar, en general, se consiguen mejoras importantes respecto al caso promedio, así como un acercamiento significativo al óptimo.

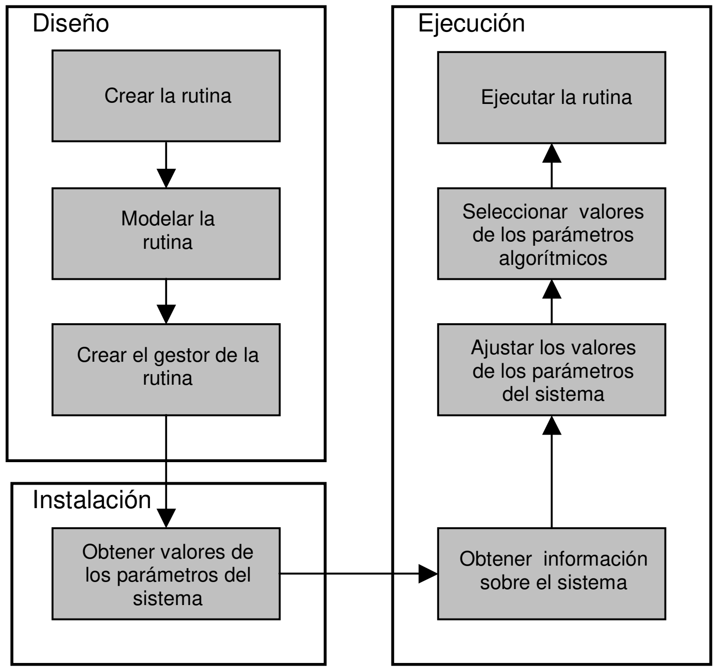


Figura 1.1. Visión general del ciclo de vida de una rutina con capacidad de ajuste automático.

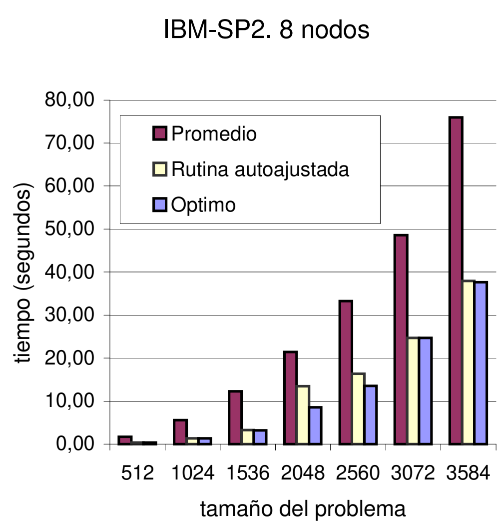


Figura 1.2. Ejemplo de comparación de las prestaciones óptimas, en promedio y las obtenidas con nuestra metodología. Para la rutina paralela de factorización QR, en un IBM SP2.

## 1.3 Aportaciones de la tesis

Las aportaciones principales de esta tesis se enmarcan dentro del campo de investigación en modelado y optimización automática de software de álgebra lineal en base a la arquitectura de la plataforma donde se ejecuta. Las más destacadas son:

- Se propone un modelo para sistemas paralelos de computación numérica. Estos sistemas están formados por la unión de la arquitectura hardware junto al software básico de álgebra lineal instalado.

- Tomando como arquitectura base el modelo del sistema propuesto anteriormente, se plantea un modelo teórico-práctico del tiempo de ejecución de cada rutina de álgebra lineal.

- Se plantea una arquitectura software que confiere a cada rutina de álgebra lineal la capacidad de adaptarse automáticamente a las condiciones de trabajo. La rutina original se mantiene intacta, por lo que la aplicación de esta metodología a todo el software de álgebra lineal existente no supone ninguna recodificación de éste. El resultado es la interposición de una capa *middleware* entre el usuario y el software de cálculo, que realiza el proceso de ajuste de este último de manera transparente al primero.

- Se plantea un sistema de intercomunicación entre rutinas de diferentes librerías de la jerarquía de librerías de álgebra lineal de cara a compartir la información necesaria para su optimización. Esta comunicación se estructura siguiendo el patrón marcado por la reutilización de código que viene implícita cuando una rutina de un nivel de la jerarquía invoca a otras de niveles inferiores.

- De cara a resolver un problema dado o llevar a cabo una operación básica dentro de una rutina compleja, se propone un sistema de selección automática de la versión de rutina a utilizar entre varias de igual funcionalidad que pertenecen a diferentes librerías equivalentes (polibrerías). Este mismo sistema de selección se puede aplicar a seleccionar el mejor algoritmo para resolver un problema (polialgoritmos).

Por otro lado, en el campo de investigación de desarrollo de algoritmos de álgebra lineal de altas prestaciones, se presenta un algoritmo de Jacobi unilateral por bloques para resolver el problema de valores propios, tanto en secuencial como en paralelo. Este nuevo algoritmo, gracias a un mayor aprovechamiento de la localidad en el acceso a los datos, obtiene mejores prestaciones que las versiones bilaterales y que las unilaterales sin bloques que existían previamente.

## 1.4 Estructura de la Memoria

En el capítulo 2 se realiza un repaso a una serie de proyectos que comparten con el nuestro el objetivo genérico de desarrollar software con capacidad de adaptarse por sí mismo a las condiciones de ejecución. Por lo general, estos proyectos considerarán tanto las condiciones de la plataforma de ejecución como del problema a resolver. La mayor parte de las propuestas vistas están centradas en el estudio de rutinas paralelas de álgebra lineal, que es donde está enmarcado este trabajo, pero también se muestran algunos proyectos que actúan sobre otro tipo de software, tanto paralelo como secuencial. La comparación entre las diferentes aproximaciones se realiza en base a una serie de descriptores que caracterizan a cada uno de estos trabajos: sobre qué software actúa, en qué tipo de plataformas desempeñan su función, qué método de trabajo emplean, qué tipo de optimización persiguen y en qué momento del ciclo de vida de las rutinas actúan. Al final del capítulo se muestra un resumen comparativo de todos estos trabajos con nuestra propuesta.

En el capítulo 3 se presenta el marco de trabajo en donde se ha desarrollado este proyecto, esto es, plataformas paralelas de propósito general y software general de álgebra lineal. Se mostrará una visión general de las arquitecturas paralelas actuales, detallándose las características concretas de las plataformas utilizadas. En la segunda parte de este capítulo, se dará un repaso por el software utilizado habitualmente en la computación numérica paralela: el software básico para la compartición y comunicación de datos entre procesadores, las librerías clásicas de software de álgebra lineal y algunas herramientas software auxiliares.

En el capítulo 4 se muestra el modelo analítico propuesto para el tiempo de ejecución de software numérico sobre una plataforma paralela genérica. Este modelo es la base sobre la que se sustenta la arquitectura software de ajuste automático de rutinas de álgebra lineal que se irá mostrando a lo largo de los siguientes capítulos. En primer lugar, como punto de partida, se repasan los principales antecedentes en el campo del modelado de computación paralela. Tras ello, se describe nuestra propuesta de modelo para un sistema paralelo de computación numérica. Sobre este modelo, que caracteriza a la arquitectura destino, se diseña el modelo analítico del tiempo de ejecución de rutinas de álgebra lineal. En la segunda parte de este capítulo, se describen las distintas rutinas que se han utilizado en los experimentos, así como sus respectivos modelos analíticos, mostrándose de una manera más detallada la rutina paralela de Jacobi unilateral por bloques para la resolución del problema de valores propios, por ser ésta una de las aportaciones de la tesis. Finalmente, se muestra una comparación, en diversas plataformas hardware, entre los tiempos teóricos que predice el modelo analítico de cada rutina frente al tiempo experimental obtenido.

En el capítulo 5 se describe detalladamente la arquitectura propuesta para una rutina de álgebra lineal con capacidad de ajuste automático. El código original de la rutina pasa a formar parte de un sistema software llamado *SOLAR* (*Self-Optimised Linear Algebra Routine*) que le confiere la capacidad de adaptarse a las condiciones del sistema. Esta adaptación se manifestará mediante la elección de valores apropiados para una serie de parámetros configurables de la rutina. Se presentan dos propuestas de *SOLAR* en este capítulo. Una primera versión de *SOLAR* orientada a plataformas donde las condiciones de carga de trabajo no sufran grandes variaciones, y una segunda propuesta de *SOLAR* válida para cualquier tipo de plataformas, incluidas aquellas donde la carga de trabajo sufre grandes variaciones. Para ambas propuestas, se describe la estructura y funcionamiento del sistema software construido, mostrándose algunos resultados experimentales obtenidos con diversas rutinas sobre plataformas de diferentes características.

En el capítulo 6 se aborda el ajuste automático del software de álgebra lineal desde una perspectiva más amplia, a nivel de librería y de las relaciones existentes entre rutinas de librerías diferentes. El objetivo de este capítulo es justificar la conveniencia de incluir un sistema de información de auto-optimización en la estructura jerárquica de las librerías clásicas de álgebra lineal. En la primera sección se muestra una propuesta para reducir el grado de experimentación que se realiza durante la instalación de una *SOLAR*. Esta reducción se consigue compartiendo la información de optimización entre diferentes rutinas. En la segunda sección se muestra cómo introducir la posibilidad de seleccionar qué librería básica utilizar para llevar a cabo cada operación dentro de una rutina de nivel superior, escogiendo entre aquellas que tengan funcionalidad semejante y que estén instaladas previamente en la plataforma.

En el capítulo 7, en primer lugar, se destacan las principales conclusiones obtenidas con la realización de esta tesis. A continuación, se comentan cuáles han sido los proyectos, estancias en otros centros y publicaciones llevados a cabo durante el desarrollo de este trabajo. Finalmente, se relacionan algunas posibles líneas futuras de trabajo.

# Capítulo 2. Trabajos Relacionados

## 2.1 Introducción

En la actualidad existe un amplio conjunto de trabajos que plantean metodologías para el diseño y construcción de software capaz de adaptarse automáticamente a las condiciones en que se encuentre la plataforma de ejecución. Aunque se mostrarán algunos proyectos que actúan sobre software diverso, nos centraremos en aquellos cuyo campo trabajo es el software de álgebra lineal. Por otro lado, algunos de estos trabajos están orientados únicamente a plataformas secuenciales, sin embargo, se ha hecho más hincapié en propuestas más generales, válidas para cualquier tipo de plataformas paralelas. Por último, también se han descrito algunas aproximaciones para plataformas de naturaleza heterogénea e incluso plataformas distribuidas. Se verá que, en general, es posible utilizar dos técnicas para ajustar las rutinas de cara a optimizar sus prestaciones. La primera técnica consistiría en el modelado analítico de las rutinas y la posterior aplicación a estos modelos de técnicas de optimización teóricas. La segunda se basaría en la realización de ejecuciones de la rutina, probando sus diferentes opciones, de cara a descubrir la óptima. En cuanto al momento en que se lleva a cabo el ajuste de una rutina, una primera opción sería cuando ésta se esté instalando y otra posibilidad es ajustar la rutina cuando se ejecuta. La primera opción se basará únicamente en una visión estática de la plataforma de ejecución. Por el contrario, un ajuste en el momento de la ejecución permitirá obtener una visión del estado real de la plataforma justo cuando se tienen que tomar las decisiones de cómo ejecutar la rutina.

En la siguiente sección se enumerarán una serie de descriptores que hemos utilizado como ficha de identidad de cada trabajo (sobre qué software actúa, en qué tipo de plataformas, ...). En la tercera sección del capítulo, se detallarán las características de los diferentes trabajos estudiados, empezando por aquellos que actúan sobre software de tipo general y siguiendo por aquellos cuyo software de actuación son rutinas de álgebra lineal, clasificados por el tipo de plataformas donde actúan. Por cada proyecto descrito se mostrará su ficha de descriptores, un resumen de su propuesta y algunos comentarios comparativos con nuestro trabajo. Por último, al final del capítulo se mostrará una comparativa general de todos estos trabajos con nuestra propuesta, así como una serie de conclusiones.

## 2.2 Descriptores de los trabajos

### 2.2.1 Software sobre el que se actúa

Nuestro estudio se ha realizado sobre rutinas de álgebra lineal (A. L.), por esa razón, nos centraremos en los que trabajan sobre este mismo software, pero también veremos algunas aproximaciones que optimizan otro tipo de rutinas. De cualquier manera, en la mayor parte de los casos, el tipo de software sobre el que actúa un proyecto no es determinante para la metodología que se sigue. Es decir, la mayoría de los proyectos, con sus respectivas metodologías, podrían aplicarse a otros tipos de software sin apenas modificaciones en sus planteamientos.

### 2.2.2 Plataformas de ejecución

En cuanto al tipo de plataformas sobre el que actúan, podemos clasificar los proyectos en:

- **Proyectos para plataformas secuenciales**: Su principal objetivo es optimizar el acceso a los diferentes niveles de la jerarquía de memoria. En algunos casos, amplían su estudio a plataformas paralelas de memoria compartida.
- **Proyectos para plataformas paralelas de memoria distribuida:** Plantean metodologías válidas, además de para secuenciales, para cualquier tipo de plataformas paralelas que puedan ser programadas mediante paso de mensajes. Podemos encontrar dos orientaciones:
  - **Proyectos para plataformas homogéneas**: Consideran la plataforma de ejecución como un sistema formado por procesadores con igual potencia de cómputo conectados por una red de interconexión que mantiene valores de latencia y ancho de banda constantes para cualquier par de nodos.
  - **Proyectos para plataformas heterogéneas**: Realizan una labor de equilibrado de la carga de trabajo de las rutinas teniendo en cuenta la heterogeneidad de las características de los procesadores y/o de la red de interconexión del sistema.
- **Proyectos para plataformas distribuidas**: Los sistemas distribuidos están formados por una serie de nodos de diferente naturaleza (ordenadores personales, estaciones de trabajo, *clusters*, ...) conectados entre sí a través de diferentes redes de área extendida (*grid computing*). Los proyectos que actúan sobre este tipo de plataformas se marcan como objetivo prioritario la búsqueda de la mejor distribución de la carga de trabajo teniendo en cuenta la heterogeneidad del sistema y, sobre todo, el alto coste de las comunicaciones entre los nodos.

### 2.2.3 Método de trabajo

Dependiendo de que la metodología que se siga para optimizar las prestaciones de las rutinas tenga una orientación teórica o experimental, podemos clasificar los trabajos en:

- **Proyectos teóricos**: Realizan la labor de ajuste de las rutinas diseñando modelos analíticos más o menos detallados que describen las rutinas para, posteriormente, aplicar técnicas de optimización teóricas a estos modelos.

- **Proyectos experimentales**: Su método de trabajo suele consistir en realizar un repertorio de pruebas de ejecución de la rutina que quieren optimizar, probando diferentes posibilidades hasta comprobar cuál de éstas es la más conveniente.
- **Proyectos teórico-experimentales:** Son trabajos que aúnan las técnicas del modelado analítico y el estudio experimental. Su objetivo es aprovechar las mejores características de cada una de ellas, es decir, el grado de abstracción y portabilidad del modelado frente a la concreción de los datos reales obtenidos experimentalmente.

### 2.2.4 Tipo de optimización

Existen diferentes objetivos de optimización que se pueden perseguir al ajustar una rutina:

- **Parametrización**: Buscar los mejores valores para una serie de parámetros ajustables del algoritmo, como el tamaño de bloque de cálculo, tamaño de los mensajes, etc.
- **Distribución del trabajo**: Elegir qué procesadores utilizar entre todos los que se tengan disponibles, y buscar la mejor distribución de los procesos y/o de los datos entre los procesadores seleccionados.
- **Polialgoritmos**: Elegir el algoritmo concreto a utilizar, entre varios posibles que se tengan disponibles, para resolver el problema concreto que el usuario plantee.
- **Polilibrerías**: Escoger de entre varias librerías, que estén instaladas y disponibles en la plataforma, la versión de la rutina más apropiada para ser ejecutada.

Como veremos a lo largo de la siguiente sección, la mayoría de los trabajos comparados utilizan a la vez más de uno de estos tipos de optimización.

### 2.2.5 Momento de actuación

Existen tres posibles momentos en los que llevar a cabo el ajuste de una rutina:

- **Instalación**: Cuando la rutina se esté instalando se lleva cabo un estudio, con mayor o menor profundidad, de las características estáticas de la plataforma, es decir, de la capacidad de cómputo de los procesadores y de la capacidad de comunicación de la red de interconexión.
- **Al iniciar la ejecución**: En el momento en que el usuario invoca la rutina se lanza el proceso de ajuste de la misma. De esta manera, al poderse recoger valores de la carga de trabajo que en ese momento está soportando cada nodo, así como la carga en la red de interconexión, se consigue una visión más actual de la situación de la plataforma. Esta información del estado del sistema permite realizar un ajuste mejor que si sólo se maneja la información obtenida en tiempo de instalación. Encontramos dos posibilidades en cuanto al proceso de testeo de la plataforma:
    - Este proceso lo realiza el propio sistema de autoajuste mediante la ejecución de algunas subrutinas propias a modo de cargas de prueba (*benchmarks*). El problema de este ajuste es que se puede introducir una sobrecarga en el tiempo de ejecución de la rutina.
    - Este proceso lo realiza alguna herramienta especializada en tal labor, como puede ser NWS [WSH99], que en cualquier momento podrá aportar información actualizada sobre el estado del sistema (disponibilidad de los distintos procesadores, disponibilidad de la red, o redes, de interconexión, ...). Estas herramientas realizan su labor de seguimiento del sistema de manera continua, actuando en un segundo plano (*background*) y no suponiendo una sobrecarga

significativa al sistema, siendo mínimo el coste de leer la información que aporta este tipo de herramientas.

- **Durante la ejecución**: Se monitoriza el comportamiento de la rutina y, en base a éste, se modifican convenientemente los valores de algunos de sus parámetros. Usando este modo de optimización se tiene en todo momento una visión del estado la plataforma muy realista, pero tiene el inconveniente de que la sobrecarga introducida en el tiempo de ejecución puede llegar a ser importante.

Algunos de los proyectos que describiremos realizan un ajuste mixto, dividido en distintos subprocesos que se reparten por cada uno de estos momentos, buscando las ventajas que ofrece realizar su labor en cada uno de ellos y, obviamente, intentando minimizar los inconvenientes.

## 2.3 Proyectos de software general

En esta sección se mostrará una relación de trabajos sobre ajuste automático de software variado (FFTs, rutinas de comunicaciones colectivas, rutinas segmentadas o *pipeline*,...), frente a los que lo hacen exclusivamente sobre software de álgebra lineal, que serán descritos en las siguientes secciones. Se muestran estos primeros proyectos debido a que plantean metodologías de trabajo semejantes a la nuestra, independientemente del software sobre el que actúan.

### 2.3.1 Proyecto de E. A. Brewer

| | |
|---|---|
| **Software sobre el que se actúa** | Algoritmos de ordenación<br>Resolución de ecuaciones diferenciales |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Parametrización<br>Polialgoritmos |
| **Momento de actuación** | Instalación |

En el proyecto de E. A. Brewer [Bre94][Bre95] se presenta una arquitectura de un sistema de gestión de una librería de alto nivel donde, automáticamente, se selecciona la mejor implementación entre varias disponibles en esta librería, para resolver un problema dado. Además, el sistema puede determinar el valor óptimo de una serie de parámetros específicos de la implementación escogida.

La labor del diseñador de las diferentes rutinas de la librería es proveer al sistema de las diferentes implementaciones y de los esquemas de los modelos de coste de éstas. Será el propio sistema, con su herramienta de calibración de modelos (*auto-calibration toolkit*), el que obtenga los modelos en su versión definitiva. Esta labor automática de calibración de modelos se llevará a cabo en el proceso de instalación y cada vez que cambien las condiciones de la plataforma. Para llevar a cabo la calibración, el sistema experimenta con las rutinas completas y, tras ello, se ajustan los valores de los parámetros que aparecen en el modelo, descartándose los parámetros no significativos, gracias a la aplicación de métodos estadísticos a las medidas experimentales.

Esta metodología ha sido aplicada a algunas rutinas de ordenación y otras de resolución de ecuaciones diferenciales sobre diferentes plataformas paralelas (un CM-5, un Alewife, un Intel Paragon y un *cluster* de estaciones de trabajo). Con posterioridad, otros autores [GLR+02][VDB01] han utilizado una metodología semejante sobre otro tipo de rutinas.

### Comentarios

- Los parámetros ajustables son específicos de cada rutina. No se plantea una visión general del proceso de parametrización de software, sino que se tiene que realizar una labor concreta por cada rutina de la librería.
- El método que se propone para adaptarse a posibles cambios en las condiciones de la plataforma es poner en marcha el proceso de recalibrado completo de las rutinas. Este proceso, equivalente a reinstalar la librería, es demasiado costoso como para realizarlo con demasiada frecuencia. Como consecuencia de esto, no se tienen en cuenta posibles cambios en la carga de trabajo de la plataforma desde el último proceso de recalibrado hasta el momento de ejecución de las rutinas, por lo que las decisiones que se tomen para resolver un problema (qué algoritmo utilizar y qué valor darle a los parámetros) pueden estar equivocadas.

## 2.3.2 Proyecto de S. Juhász y H. Charaf

| | |
|---|---|
| **Software sobre el que se actúa** | Algoritmos de ordenación |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Distribución del trabajo |
| **Momento de actuación** | Instalación |

En la propuesta de S. Juhász y H. Charaf [JC02] se realiza un estudio de modelado de rutinas paralelas de cara a estimar el número óptimo de procesadores a utilizar en un *cluster* homogéneo de procesadores. Con este objetivo se realizan tres submodelos analíticos por separado para el cómputo aritmético, las comunicaciones y las operaciones de E/S. El modelo de la rutina completa se forma como resultado de aunar estos tres submodelos. En este modelo aparecen una serie de parámetros del sistema cuyos valores se miden en tiempo de instalación.

### Comentarios

- Cuando la plataforma es heterogénea se considera virtualmente homogénea, considerando al nodo más lento como el que va a marcar el ritmo de cómputo de la rutina paralela, debido a las sincronizaciones explícitas o implícitas entre los procesadores. Esta técnica se asemeja a la que utilizamos a la hora de modelar el tiempo de ejecución de rutinas que tienen una distribución homogénea de trabajo cuando se ejecutan sobre plataformas heterogéneas. En nuestro proyecto sacamos mayor partido de esta aproximación, utilizándola para decidir qué procesadores utilizar y cuáles no.
- Se realiza todo el proceso en momento de instalación, no se tiene en cuenta la situación de la plataforma en el momento de ejecución.

### 2.3.3 Proyecto SANS

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra lineal<br>Comunicaciones colectivas<br>Procesamiento de señales |
| **Plataformas de ejecución** | Plataformas secuenciales<br>Plataformas paralelas de memoria distribuida<br>Plataformas distribuidas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Parametrización<br>Polialgoritmos<br>Polilibrerías<br>Distribución del trabajo |
| **Momento de actuación** | Instalación + Ejecución |

SANS (*Self-adapting Numerical Software*) es, más que un proyecto concreto, una filosofía o marco de trabajo que han planteado los investigadores del laboratorio ICL de la Universidad de Tennessee [wSANS] en el campo de ajuste automático de software. Esta filosofía se está aplicando en la actualidad a varios proyectos del propio ICL y de otros centros de investigación. Su principal objetivo es la construcción de software numérico de altas prestaciones, destinado tanto a *clusters* locales como a sistemas distribuidos, que pueda ser utilizado por usuarios no expertos con gran facilidad. Dado un problema a resolver, se basa en el manejo inteligente de conocimiento sobre la estructura de los datos de entrada al problema, de los posibles algoritmos para resolverlo y de las plataformas de ejecución. De manera general, los componentes de un sistema SANS serían:

1. Un *agente SANS*, que a su vez consta de:
   - 1.1. Un *componente inteligente*, que básicamente estudia la estructura de los datos de entrada.
   - 1.2. Un *componente de sistema*, encargo de recopilar información sobre los diferentes recursos la plataforma de ejecución: recursos hardware (diferentes procesadores disponibles, red de interconexión, ...) y los recursos software (librerías disponibles en cada procesador).
   - 1.3. Una *base de datos histórica*, que almacena, para cada problema que se resuelve, un serie de metadatos que describen la estructura de los datos de entrada y, además, los datos sobre las prestaciones obtenidas al resolver ese problema con unos recursos hardware y software determinados.
2. Un *lenguaje script* para comunicarse con el usuario y con las diferentes librerías de software.
3. Un *vocabulario de metadatos*.
4. Un *conjunto de librerías* compiladas de manera óptima en las diferentes plataformas y preparadas para ser invocadas por un *agente SANS*.

Mediante el diálogo entre el *componente inteligente* y el *componente de sistema*, y utilizando la información de la *base de datos histórica* se seleccionan los componentes de las librerías más adecuados a los datos de entrada, características del problema y situación de la plataforma.

De manera genérica y con diferentes aproximaciones, la filosofía de desarrollo de software autoajustable planteada en el proyecto SANS está siendo actualmente aplicada a diferentes campos de trabajo, algunos de manera directa en el propio ICL:

- Software de álgebra lineal para matrices escasas, en el proyecto SALSA [DE02].
- Software de álgebra lineal para matrices densas en *clusters*, en el proyecto LFC [RD02].
- Optimización automática de comunicaciones colectivas [VFD00].
- Núcleos básicos de álgebra linear de altas prestaciones, en el proyecto ATLAS [WPD01].

También está siendo aplicada en otros proyectos mediante colaboraciones del ICL con otros centros de investigación:

- Software de álgebra lineal para matrices escasas en el proyecto BeBOP [VDY+02][VKH+02] con la Universidad de Berkeley.
- Software de procesamiento de señales en los proyectos SPIRAL [MJJ+00], UHFFT [MMJ00] y SALT [FFF+01] con las Universidades de Houston y de Viena.

Las características más importantes de estos proyectos serán descritas a lo largo de diferentes secciones de este capítulo.

**Comentarios**

Se plantea una metodología de trabajo, donde hay que tomar las decisiones de ajuste del software en base a la información aportada por tres componentes principales: uno que estudia el problema a resolver, otro encargado de analizar la situación del sistema hardware y un tercero formado por la información histórica de ejecuciones anteriores de las rutinas. En nuestra propuesta existe un cuarto componente, el modelo analítico del tiempo de ejecución de cada rutina, que muestra una visión teórica de su comportamiento sobre una plataforma dada. Este modelo es una herramienta muy útil para guiar la labor de ajuste, reduciendo el tiempo de este proceso.

### 2.3.4 Proyecto SPIRAL

| | |
|---|---|
| **Software sobre el que se actúa** | Procesamiento de señales |
| **Plataformas de ejecución** | Plataformas secuenciales |
| | Plataformas paralelas de memoria compartida |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Polialgoritmos |
| **Momento de actuación** | Instalación |

Gran parte de los problemas de procesamiento de señales pueden ser descritos usando fórmulas matemáticas, de tal forma que una amplia variedad de algoritmos matemáticamente equivalentes se pueden generar fácilmente. Sin embargo, estos algoritmos equivalentes pueden tener diferentes prestaciones según el sistema donde se ejecuten.

El objetivo general del proyecto SPIRAL [MJJ+00] es desarrollar una librería optimizada y portable que cuente con diferentes algoritmos de procesamiento de señales.

Los elementos de su arquitectura son: un lenguaje especializado para estos problemas, un generador de algoritmos a partir de las fórmulas, un generador de código a partir de los algoritmos y, finalmente, un evaluador de prestaciones que analiza el comportamiento de los diferentes algoritmos e implementaciones.

Dada una plataforma, el evaluador experimenta con una implementación de un algoritmo, tras ello, retroalimenta al generador de algoritmos y al generador de código con la información de las prestaciones obtenidas. Después, estos generadores producirán las correspondientes variantes del algoritmo y de su implementación, que tendrá que volver a ser evaluada. Este proceso se repetirá sucesivamente en busca de una versión óptima del algoritmo y su correspondiente implementación.

**Comentarios**

Al realizar todo su proceso en el momento de la instalación, esta propuesta asume un entorno de ejecución estable. Sin embargo, si hay variaciones en la carga de trabajo de la plataforma, puede ocurrir que las implementaciones y algoritmos seleccionados en el proceso de instalación ya no lo sean cuando se va a ejecutar el software.

### 2.3.5 Proyecto FFTW

| | |
|---|---|
| **Software sobre el que se actúa** | Transformada rápida de Fourier (FFT) |
| **Plataformas de ejecución** | Plataformas secuenciales |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Polialgoritmos |
| **Momento de actuación** | Instalación + Ejecución |

En el proyecto FFTW [FJ97][FJ98][FK01] el principal objetivo es la minimización automática del tiempo de resolución de transformadas rápidas de Fourier (FFT, *fast Fourier transforms*). La arquitectura de esta propuesta incluye como principales elementos:

- *Codelet generator:* Un generador de código que, durante la instalación, genera automáticamente *codelets* (trozos de código especializados en resolver FFT de un determinado tamaño fijo). Estos *codelets* son teóricamente óptimos según un modelo analítico del tiempo de ejecución donde queda recogida la utilización de la memoria de la plataforma así como la cantidad de cálculo a realizar.
- *Planner*: Se encarga de seleccionar, en tiempo de ejecución, el mejor plan (combinación de *codelets*) para un problema dado en una plataforma concreta. Para ello realiza ejecuciones a fin de escoger el mejor plan de entre un conjunto preestablecido (conjunto ya preseleccionado por un algoritmo de programación dinámica que escoge a priori, sin necesidad de experimentación, los mejores planes).
- *Executor*: Pone a funcionar el plan seleccionado por el *planner*.

**Comentarios**

El proceso del *planner* puede producir una importante sobrecarga en el tiempo total de ejecución. Posteriores extensiones, como UHFFT [MMJ00] han buscado resolver este problema a base de realizar parte de este proceso durante la instalación de la librería.

### 2.3.6 Proyecto SALT

| | |
|---|---|
| **Software sobre el que se actúa** | Procesamiento de señales |
| **Plataformas de ejecución** | Plataformas paralelas heterogéneas<br>Plataformas distribuidas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Polialgoritmos<br>Distribución del trabajo |
| **Momento de actuación** | Instalación + Ejecución |

Tomando como punto de partida las implementaciones FFTW y SPIRAL, la propuesta del proyecto SALT [FFF+01] tiene por objetivo el desarrollo de prototipos específicos de algoritmos de procesamiento de señales para sistemas heterogéneos.

En la metodología que propone SALT se estudia la situación de la plataforma para seleccionar los recursos a utilizar y el reparto del trabajo entre ellos. Como trabajo futuro tienen previsto adaptar SALT a la ejecución en sistemas distribuidos en la red (*grid environment*).

#### Comentarios

Para conocer la situación de la plataforma se utiliza la herramienta NWS. Varios proyectos para plataformas heterogéneas y/o distribuidas han tomado esa misma decisión, entre ellos el nuestro.

### 2.3.7 Proyecto de Vadishar *et al.*

| | |
|---|---|
| **Software sobre el que se actúa** | Rutinas para comunicaciones colectivas |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas |
| **Método de trabajo** | Experimental |
| **Tipo de optimización** | Parametrización |
| | Polialgoritmos |
| **Momento de actuación** | Instalación + Ejecución |

En [VFD00] se propone una metodología para ajustar automáticamente ciertos parámetros de las operaciones de comunicación colectivas. Dada un plataforma, con un número de procesadores concreto, para cada operación colectiva se propone un método de búsqueda automática del algoritmo de comunicación óptimo (árbol secuencial, árbol en anillo, árbol binario, etc.) y del tamaño óptimo de segmentación de los mensajes (se ajustará lo mejor posible al tamaño de los *buffers* del sistema de comunicación). Este proceso se ayuda de diversos métodos heurísticos con los que reducir el espacio de búsqueda.

#### Comentarios

- No se apoya en ningún modelo analítico que le pudiera ayudar en el proceso de búsqueda de parámetros óptimos.
- Algunas de las técnicas utilizadas en esta aproximación para reducir el tiempo de experimentación son semejantes a las que hemos utilizado en nuestra propuesta.

### 2.3.8 Proyecto GTP de Rodríguez *et al.*

| | |
|---|---|
| **Software sobre el que se actúa** | Software diverso: Problema de la Mochila, FFT, ... |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Parametrización |
| | Distribución del trabajo |
| | Polialgoritmos |
| **Momento de actuación** | Ejecución |

En el grupo de investigación de Casiano Rodríguez *et al*. de la Universidad de La Laguna [wULL] se plantea una formalización general para el ajuste de programas paralelos (*General Tuning Problem*, GTP). En su planteamiento se unifican dos técnicas generales de mejora de las prestaciones de un programa: el estudio teórico del tiempo de ejecución (modelado analítico) y el seguimiento de la prestaciones reales durante su ejecución (*profiling*). Las fases que plantean para ajustar las prestaciones de un programa son:

1. El usuario realiza un modelo analítico del tiempo de ejecución del programa, donde quedan reflejadas sus prestaciones en función de las características del sistema y del tamaño del problema a resolver.
2. La herramienta de *profiling* calcula los parámetros del sistema que aparecen en el modelo mediante instrumentación del programa y seguimiento de su ejecución en sus primeras iteraciones o fases.
3. Conocidos los parámetros del sistema, se buscan los valores para parámetros ajustables del modelo que minimicen el tiempo de ejecución y entonces se utilizan éstos para el resto de la ejecución del programa.

Dentro de este proyecto encontramos dos aproximaciones:

En [MAG+01][GAM+03], utilizando un modelo analítico tradicional, aplican esta metodología a algoritmos con funcionamiento de tipo *segmentado* (*pipeline*). Para ello han introducido capacidad de realizar *profiling* en su herramienta LLP [MAG+99] (una herramienta de propósito general para el paradigma de programación en *pipeline*). Los parámetros cuyos valores se ajustan son: el número de procesadores a utilizar, el número de procesos asignados por procesador y el tamaño del *buffer* de las comunicaciones. Este planteamiento está siendo modificado y ampliado de cara a ser aplicado a otros paradigmas, como el maestro-esclavo, en plataformas heterogéneas [AGM+03].

En [GLR+02][BGL+04] usan el modelo analítico el OBSP* (una modificación del *Oblivious* BSP, OBSP [GLP+99]), introduciendo la característica de que los parámetros del sistema no son considerados como constantes sino que dependen también de las características del algoritmo. Se presenta en este trabajo un prototipo de arquitectura de ajuste de programas formada por tres componentes principales: la herramienta CALL, que genera automáticamente código de instrumentación a introducir en la rutina para realizar su seguimiento en ejecución, la librería CLL y la herramienta LLAC que analiza los datos generados con la instrumentación de cara a proporcionar valores para los parámetros del sistema.

## Comentarios

El planteamiento general es semejante al nuestro, aunque encontramos dos diferencias significativas:

- La información sobre el comportamiento de la rutina a ajustar se obtiene en tiempo de ejecución mediante *profiling* sobre la propia ejecución de dicha rutina. La inclusión de este código de monitorización puede suponer una sobrecarga al tiempo de ejecución .
- Este planteamiento está más bien dirigido a esquemas (*skeletons*) algorítmicos en lugar de a rutinas independientes y al desarrollo de librerías.

## 2.4 Proyectos de A. L. para plataformas secuenciales

### 2.4.1 Proyecto LAWRA

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra lineal |
| **Plataformas de ejecución** | Plataformas secuenciales |
| | Plataformas paralelas de memoria compartida |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Parametrización |
| **Momento de actuación** | Ejecución |

En el proyecto LAWRA (*Linear Algebra With Recursive Algorithms*) [AGK+00][wLAWRA] el estudio se centra en mejorar el uso de la jerarquía de memoria de las plataformas actuales cuando ejecutan rutinas de álgebra lineal.

Por un lado, en [Gus97] se plantea modificar las rutinas de la librería secuencial de álgebra lineal LAPACK [ABB+90] transformándolas en rutinas recursivas, pero sin cambiar su funcionalidad. El motivo principal de este cambio es que se consigue que el tamaño de bloque, que va variando a lo largo de la ejecución del algoritmo, sea implícita y automáticamente escogido por el mecanismo de recursividad, de manera que se ajuste lo mejor posible al tamaño de la memoria caché de la plataforma.

Por otro lado, para hacer uso de las rutinas de BLAS* [DDD+90] es necesario cambiar el formato de almacenamiento de los datos con los que operan a un formato de bloques para poder conseguir buenas prestaciones de ejecución. Los autores de LAWRA plantean en [GHJ+98] usar ese formato por bloques de BLAS como formato general de almacenamiento para los datos de matrices y vectores densos. De esta manera las rutinas de BLAS llamadas recibirían sus operandos en el formato que precisan, ahorrándose el cambio de éste.

En [KLV98] plantean una nueva versión de la librería BLAS de nivel 3, *GEMM-based Level 3 BLAS*, donde sus rutinas tienen como parte dominante de la computación llamadas a la rutina *GEMM*. Se reducen al mínimo las llamadas a rutinas de los niveles inferiores de BLAS ya que éstas no permiten una utilización eficiente de la memoria caché de la plataforma. Con esto, además, se consigue que todas las mejoras que se realicen a posteriori en la rutina *GEMM* se trasladen automáticamente al resto de las rutinas de BLAS-3.

En [GHJ+98a] se introducen en su librería BLAS-3 basada en *GEMM* algunas técnicas más avanzadas de cara a plataformas con varios niveles de caché. Algunas de estas técnicas son: diferentes niveles de división por bloques, desenrollado de bucles y precarga de datos.

En [EG00][GJ00][AWG01] todo este planteamiento general de un uso eficiente de la jerarquía de memoria en monoprocesadores se extiende a sistemas paralelos de memoria compartida al incluir procesamiento *multithreading*.

* Las rutinas de BLAS son utilizadas masivamente tanto por las de LAPACK como por rutinas de más alto nivel.

### Comentarios

- La transformación de las rutinas de LAPACK a versiones recursivas puede requerir un gran esfuerzo de programación.
- El parámetro fundamental de las rutinas (tamaño de problema base donde la recursividad debe detenerse para que se resuelva el subproblema de manera directa) se ajusta en base a condiciones estáticas de la plataforma (básicamente el tamaño de caché). No se tiene en cuenta posibles cambios en la carga de trabajo del sistema desde el momento de instalación hasta el momento de ejecución.
- La selección de un tamaño de bloque de manera implícita mediante la recursión se puede combinar con técnicas de elección explícita de tamaño de bloque (como la que se realiza en nuestra propuesta) para obtener mejores prestaciones [Gus97].
- En general, las mejoras en la codificación de rutinas secuenciales, como las llevadas a cabo en este proyecto, son totalmente compatibles y complementarias con propuestas de ajuste automático de rutinas donde, dado un problema a resolver, exista un proceso de selección de la rutina concreta a utilizar de entre las librerías instaladas en el sistema (polilibrerías). Son varios los proyectos, entre ellos el nuestro, que han incorporado el uso de polilibrerías.

## 2.4.2 Proyectos de auto-ajuste de núcleos de A. L.

| | |
|---|---|
| **Software sobre el que se actúa** | Multiplicación matriz-matriz |
| **Plataformas de ejecución** | Plataformas secuenciales |
| **Método de trabajo** | Experimental |
| **Tipo de optimización** | Parametrización + Polialgoritmos |
| **Momento de actuación** | Instalación + Ejecución |

Dentro del marco de desarrollo de rutinas autoajustables un campo que está suscitando gran interés en los últimos años es el de creación de núcleos (*kernels*) secuenciales de álgebra lineal, que realizan un profundo trabajo de ajuste a bajo nivel, muy cercano al nivel hardware, durante su instalación en una plataforma dada. Estos núcleos son la base sobre la que se van construyendo los sucesivos niveles superiores de rutinas de álgebra lineal. El principal de estos núcleos, debido a su uso extensivo e intensivo en gran parte de las rutinas de álgebra lineal de nivel medio y alto, es la multiplicación matriz-matriz, que en BLAS viene representada por la rutina *GEMM* (*GEneral Matrix times Matrix*). *GEMM* es la rutina más utilizada de BLAS, incluso hay proyectos que proponen un tercer nivel de BLAS completamente escrito en función de esta rutina [KLV98].

La rutina *GEMM*, a su vez, se puede implementar eficientemente en términos de un núcleo interior (*inner-kernel*) que lleve a cabo multiplicaciones matriz-matriz en bloques de un determinado tamaño de la matriz completa.

### Proyecto PHiPAC

PHiPAC [BAC+97] fue el primer proyecto que apareció siguiendo estas ideas de mejorar la rutina *GEMM*. En esta propuesta, durante la compilación se experimenta con diferentes posibilidades de división de las matrices en bloques, se monitorizan estos experimentos y entonces se determinan los parámetros para obtener una multiplicación de altas prestaciones.

### Proyecto ATLAS

En el proyecto ATLAS [WPD01] las características de la *GEMM* que son propias del hardware de la plataforma se estudian por separado. Para ello, han desarrollado un generador de código que determina experimentalmente, en tiempo de compilación, el tamaño de bloque y el grado de desenrollado de bucles necesarios de cara a obtener un núcleo interno de *GEMM* que ofrezca las máximas prestaciones ejecutándose *on-chip*, es decir, manejando datos contenidos en registros de la CPU y en el nivel primero de la caché de una plataforma dada. Por otro lado, implementa dos posibles algoritmos para el núcleo externo de *GEMM* (*off-chip*) que envuelve al núcleo interno. Esta parte externa sería común para todas las plataformas.

### Proyecto ITXGEMM

En ITXGEMM [GHvG01] se utiliza un modelo matemático para la jerarquía de memoria de la plataforma de cara a obtener una técnica eficiente de división de la matriz en bloques. En tiempo de compilación se generan diferentes versiones de la rutina de multiplicación, cada una con una técnica diferente de separación de datos en diferentes niveles de bloques. En tiempo de ejecución se decide heurísticamente, en función de las dimensiones de las matrices a multiplicar, qué versión de la rutina se usará. En [Hen01] se introduce la idea de utilizar un núcleo interno de *GEMM* que pueda ser auto-modificado en tiempo de ejecución en función de la estructura de los datos de entrada.

### Proyecto LAWRA-GEMM

Dentro del proyecto LAWRA [AGK+00] anteriormente descrito, también ha surgido la propuesta para implementar la propia rutina *GEMM* mediante un algoritmo recursivo, que, a su vez, maneje estructuras de datos que cuenten también con una naturaleza recursiva. Esta iniciativa viene motivada principalmente por la idea de que la partición recursiva de los operandos en bloques da lugar a que los tamaños de éstos últimos se ajusten automáticamente a las dimensiones de los distintos niveles de la jerarquía de memoria de la plataforma.

### Comentarios

- Estos cuatro proyectos consiguen importantes mejoras en las prestaciones de la rutina secuencial de multiplicación matricial *GEMM*.
- En nuestra propuesta, para aquellas rutinas de más alto nivel que tienen llamadas a la rutina *GEMM*, se realiza una selección automática de la versión más óptima de *GEMM* que se encuentre instalada en la plataforma de trabajo. Esta decisión se realiza en función del estado de la plataforma y del problema concreto a resolver. La versión elegida de *GEMM* en cada momento puede ser PHiPAC, ATLAS, ITXGEMM o LAWRA. De igual manera, también se pueden escoger otras versiones de *GEMM* que han sido optimizadas “a mano”, modificando el algoritmo implementado [wGoto][HN02][wHen] y/o haciendo versiones especiales de la implementación para diferentes plataformas concretas [wBLAS].

## 2.4.3 Proyecto BeBOP

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra lineal de matrices escasas |
| **Plataformas de ejecución** | Plataformas secuenciales |
| **Método de trabajo** | Teórico |
| **Tipo de optimización** | Parametrización<br>Polialgoritmos |
| **Momento de actuación** | Ejecución |

En el proyecto BeBOP [VDB01][VDY+02][VKH+02], se estudia la interacción entre las aplicaciones de usuario y los compiladores con el objetivo de ajustar automáticamente las prestaciones de las rutinas. Este trabajo se centra en rutinas de álgebra lineal para matrices escasas en plataformas secuenciales.

Su modo de trabajo consiste en generar, para cada problema a resolver, un conjunto de posibles algoritmos de entre los cuales se elegirá en tiempo de ejecución el que mejor se adapte a los datos de entrada. El proceso de selección de los algoritmos se puede realizar usando varios métodos, que cuentan con diferente grado de acierto a la hora de elegir el algoritmo óptimo, así como con diferente coste de computación.

Además de la selección del algoritmo óptimo, se manejan algunos parámetros referentes al bloque de compresión utilizado para almacenar los elementos de la matriz y a la reorganización de cada matriz escasa como el conjunto de una submatriz densa y otra submatriz escasa. La búsqueda de los valores óptimos para estos parámetros se lleva a cabo mediante diferentes técnicas heurísticas, que buscan maximizar la localidad de acceso a los datos del problema en base a las características de la matriz escasa que se usa y de la plataforma. De cara a la validación de estos métodos heurísticos, se diseña un modelo para obtener la cotas superior e inferior del tiempo de ejecución de una rutina en función de accesos a memoria (según la tasa de aciertos en los accesos a caché).

**Comentarios**

Las decisiones de ajuste automático se toman, únicamente, en base a las características del problema. No se tiene en cuenta la carga de trabajo que soporta el sistema.

## 2.5 Proyectos de A. L. para plataformas paralelas homogéneas

### 2.5.1 Proyecto *Multicomputer Toolbox*

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra Lineal |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas |
| **Método de trabajo** | Teórico |
| **Tipo de optimización** | Polialgoritmos |
| | Polilibrerías |
| **Momento de actuación** | Ejecución |

En el proyecto *Multicomputer Toolbox* [SLS+94] se realiza un planteamiento global de un conjunto unificado de librerías con las siguientes aportaciones:

- Las librerías incluyen algoritmos paralelos, con sus correspondientes rutinas, que están preparados para operar independientemente de la distribución de los datos que manejan. Esto permitiría que se hicieran sucesivas llamadas a diferentes rutinas que trabajan sobre un mismo conjunto de datos sin tener que realizar redistribuciones de éstos entre cada par de llamadas, evitando el consecuente coste de las comunicaciones. Además, para mejorar el acceso a los datos se plantean esquemas de almacenamiento locales que se ajusten mejor al que precisan rutinas de librerías básicas como BLAS.
- Por otro lado, se plantea en este proyecto un esquema de polialgoritmos para resolver cada problema. En tiempo de ejecución se decide heurísticamente en función de las características concretas del problema planteado (por ejemplo la forma de las matrices operandos) y del número de procesadores de la plataforma, qué algoritmo utilizar de cara a minimizar el coste computacional.

Este proyecto ha sido aplicado a algunas rutinas de álgebra lineal como multiplicación de matrices [LSF95], resolución de sistemas de ecuaciones [BSB+93], etc.

### Comentarios

- En la elección de un algoritmo para llevar a cabo la resolución de un problema no se tienen en cuenta características del sistema ni su situación de carga de trabajo, sino únicamente las características del problema y el número de procesadores de la plataforma. Esto produce un alejamiento de la situación real de la plataforma de ejecución y, por tanto, una selección errónea del mejor algoritmo.
- Es necesario recodificar las rutinas originales para que puedan trabajar sobre los datos de entrada independientemente de su distribución.

## 2.5.2 Proyecto ILIB

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra lineal |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas |
| **Método de trabajo** | Experimental |
| **Tipo de optimización** | Parametrización |
| **Momento de actuación** | Instalación |

En el proyecto ILIB [Kat00][KKK01] se plantea una metodología de ajuste automático de algunas rutinas de álgebra lineal: rutinas de resolución de sistemas de ecuaciones con matrices escasas [KKK99], tridiagonalización de matrices en problemas de valores propios [KKK00], etc.

En este planteamiento se realiza un estudio de la parametrización "a medida" de cada rutina de manera aislada (*auto-tuning software for dedicated usage*), en lugar de abordar un método general. Para cada rutina, el conjunto de parámetros a estudiar se obtiene extrayendo de su algoritmo una serie de puntos clave donde se pueda tomar una decisión, como, por ejemplo, qué grado de desenrrollado utilizar en cada bucle o qué tipo de comunicación utilizar para llevar a cabo cada transmisión del algoritmo.

La búsqueda del valor óptimo de cada parámetro se realiza mediante ejecuciones de la rutina completa para los posibles valores de este parámetro, manteniendo el resto del conjunto de parámetros a un valor constante.

### Comentarios

- El método de búsqueda de los valores óptimos de cada parámetro por separado no considera la posibilidad de que haya una dependencia entre parámetros que haría necesario que esta búsqueda fuese global en todo el espacio de valores de estos parámetros.
- La búsqueda de valores óptimos de los parámetros se hace mediante ejecuciones completas de la rutina, lo que puede aumentar el tiempo de este proceso de ajuste. Este tiempo, medido por los propios autores, llega a ser de varias horas por cada rutina de la librería.
- No tiene en cuenta que posibles cambios en el sistema, principalmente en la carga de trabajo que soporta, desde el momento en que se realiza la instalación hasta el momento en que las rutinas son ejecutadas podrían provocar un cambio en el valor óptimo de los parámetros.
- Cada rutina se estudia para su optimización por separado sin considerar la posibilidad de utilizar información conseguida por otra rutina que también haya sido previamente optimizada en el mismo sistema.

### 2.5.3 Proyecto de Zhang *et al.*

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra lineal |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Parametrización |
| **Momento de actuación** | Instalación |

En el planteamiento de Zhang *et al*. [Zha99][Zha99a][Zha00][ZC03] se presenta una metodología para la búsqueda del tamaño de bloque óptimo ($b_{opt}$) en las rutinas paralelas de factorización LU y QR de ScaLAPACK. Dada una de estas rutinas se realizan cuatro tareas por separado:

1. Estudio experimental del $b_{opt}$ para los cálculos de la rutina ejecutados localmente en cada procesador. Se realiza una serie de experimentos con la rutina de multiplicación general matriz por matriz, *GEMM* de la librería BLAS, que constituye la base del cálculo aritmético de estas rutinas de factorización.
2. Estudio teórico-experimental del $b_{opt}$ que minimice el coste de las comunicaciones de la rutina. Se diseña un modelo que describe las necesidades de comunicación de la rutina en función de las características de la red, la longitud de los mensajes, el tamaño de bloque y el número de procesadores. La información sobre la red de interconexión que precisa este modelo se obtiene experimentalmente con rutinas de intercambio de datos entre nodos.
3. Estudio analítico del $b_{opt}$ que minimice los requerimientos de memoria de la rutina. Se diseña un modelo que describe las necesidades de memoria como función del tamaño del problema a resolver, el número de procesadores, la configuración lógica del conjunto de procesadores y el tamaño de bloque.
4. Estudio analítico del $b_{opt}$ para conseguir el mejor balanceo de la carga de trabajo de la rutina. Se diseña un modelo que describe la carga de trabajo de la rutina en función del tamaño del problema, la configuración lógica del conjunto de procesadores y el tamaño de bloque.

Tras completar estos estudios por separado se comprueba si existe un valor óptimo para el tamaño de bloque común en los cuatro campos de optimización. Si no es así, se elige cuál de estos cuatro es el más importante, tomándose el $b_{opt}$ para ese campo. En los resultados presentados es el cuarto factor estudiado (balanceo de la carga) el que resulta más significativo, marcando éste el valor para $b_{opt}$.

#### Comentarios

- Para cada rutina se desarrollan cuatro submodelos que reflejan cómo influyen en la elección del $b_{opt}$ el coste aritmético local, el coste de las comunicaciones, los requerimientos de memoria y el balanceo de la carga. Sin embargo, no se lleva a cabo una unificación de estos submodelos, con las ponderaciones necesarias, para obtener un modelo único que permitiría tomar una mejor decisión para $b_{opt}$.
- No tiene en cuenta posibles cambios en la carga de trabajo del sistema desde el momento de instalación hasta el momento de ejecución. Estas variaciones podrían provocar un cambio en el valor de $b_{opt}$.

## 2.6 Proyectos de A. L. para plataformas paralelas heterogéneas

En los últimos años se ha incrementado notablemente la utilización de conjuntos de ordenadores independientes conectados por una red local como una manera relativamente barata y sencilla de obtener una plataforma paralela. Lo que ocurre es que, a diferencia de los multiprocesadores propiamente dichos, estas redes de ordenadores suelen ser sistemas heterogéneos consistentes en ordenadores con prestaciones diversas. Además, éstos pueden estar conectados por enlaces con diferentes valores de latencia y ancho de banda. Como consecuencia de todo esto, si se utiliza el software paralelo tradicional las prestaciones pueden verse reducidas notablemente debido a que este software no va a tener en cuenta las características heterogéneas de estas plataformas y va a realizar una distribución de trabajo homogénea. Además, en estos sistemas, suele existir el problema añadido de que la carga de trabajo que soportan los distintos procesadores suele ser también heterogénea y variar dinámicamente.

De cara a intentar balancear la carga conforme a las condiciones heterogéneas de estas plataformas se pueden seguir varias estrategias, que serían, en orden creciente de la eficiencia del código final y del costo de llevarlas a cabo:

1. **Selección de procesadores**. Consiste en hacer una selección de aquellos procesadores que ofrezcan mejores prestaciones de cómputo y de comunicaciones, dejando sin usar los demás. Tras ello, se distribuye el trabajo de manera homogénea entre los procesadores escogidos. Esta es la estrategia menos agresiva, en tanto en cuanto no supone ninguna modificación de las rutinas diseñadas para sistemas homogéneos.

2. **Distribución heterogénea de procesos**. Consiste en realizar una distribución heterogénea de los procesos entre los procesadores, de manera que si, por ejemplo, un procesador $A$ tiene el doble de potencia que otro $B$, a $A$ se le asignarán el doble de procesos que a $B$. Esta estrategia requiere una pequeña recodificación de las rutinas, concretamente en la fase previa en donde se generan los procesos y se mapean sobre los procesadores. El cuerpo de la rutina, que corresponde con el algoritmo de cálculo implementado, permanece intacto.

3. **Distribución heterogénea de datos.** Consiste en realizar una distribución heterogénea de los datos entre los procesos. En esta estrategia es necesario un laborioso proceso de recodificación del cuerpo central de cada rutina, para adaptarla a un nuevo algoritmo de naturaleza heterogénea.

A continuación se muestran varios proyectos que utilizan alguna de estas tres estrategias.

### 2.6.1 Proyecto mpC

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra Lineal |
| **Plataformas de ejecución** | Plataformas paralelas heterogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Distribución del trabajo |
| **Momento de actuación** | Instalación + Ejecución |
| **Estrategia heterogénea** | Distribución heterogénea de procesos + Distribución heterogénea de datos |

En el proyecto mpC [KKP01][KKP+02][Las03] se presenta un lenguaje paralelo que, a modo de capa intermedia entre el usuario y el código final de las rutinas paralelas, permite la codificación de software sobre sistemas paralelos heterogéneos sin que el usuario tenga que manejar mucha información de cada plataforma donde se puede ejecutar. Las rutinas escritas se adaptan automáticamente a las condiciones del sistema en el momento en que se ponen a funcionar, realizando un chequeo para ver los procesadores disponibles, sus prestaciones, las prestaciones de sus enlaces con la red y la carga que cada procesador soporta. Entonces se distribuyen los datos del problema en función de esta disponibilidad heterogénea de potencia de cómputo y de comunicación. Además se pueden redistribuir los datos dependiendo de cambios dinámicos de la carga de trabajo que soporten los diferentes procesadores. El método que se utiliza para establecer las prestaciones de los diferentes procesadores de la plataforma, tanto durante la instalación como en tiempo de ejecución, consiste en ejecutar un programa paralelo especial, a modo de *benchmark*, en la plataforma.

El proyecto mpC ha sido aplicado a diversas aplicaciones de negocios y de ingeniería de software, sin embargo, nuestro estudio comparativo se ha centrado en su utilización con software de álgebra lineal [AKL+98][KL01]. En este proyecto se han recodificado varias rutinas paralelas de álgebra lineal que utilizan en su código llamadas a rutinas secuenciales de LAPACK y BLAS, al estilo de ScaLAPACK, pero orientado a sistemas heterogéneos. De igual forma han creado un interfaz entre mpC y ScaLAPACK que permite migrar aplicaciones de alto nivel basadas en ScaLAPACK a software que se ejecuta en una red heterogénea de manera eficiente.

**Comentarios**

- La recodificación de las rutinas puede llevar demasiado coste de personas y tiempo.
- El tener que ejecutar un *benchmark* para detectar variaciones dinámicas de la carga de trabajo del sistema y redistribuir el trabajo si fuese necesario puede producir una sobrecarga importante en el tiempo de ejecución de las rutinas.
- Según los propios autores, las técnicas de modelado de nuestra propuesta podrían integrarse en mpC de cara a mejorar su sistema de toma de decisiones.

## 2.6.2 Proyecto de Yves Robert *et al.*

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra Lineal |
| **Plataformas de ejecución** | Plataformas paralelas heterogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Distribución del trabajo |
| **Momento de actuación** | Ejecución |
| **Estrategia heterogénea** | Distribución heterogénea de datos |

En el proyecto de Yves Robert *et al*. [BPR+99][BBP+01][BBR+01][BLR+01] se están modificando las rutinas de alto nivel de la librería ScaLAPACK referentes a la distribución de los datos, de cara a conferirles capacidad para adaptarse a posibles condiciones heterogéneas de la plataforma. Cuando una rutina va a ser ejecutada, se miden las prestaciones de los diferentes procesadores experimentalmente mediante alguna rutina simple como, por ejemplo, una multiplicación matricial a pequeña escala. Tras ello, se decide la mejor distribución de los datos y, finalmente, se procede a la ejecución de la rutina.

En la actualidad, sobre plataformas heterogéneas, están obteniendo mejores prestaciones con las rutinas que han implementado que las correspondientes de la librería ScaLAPACK tradicional, que obviamente fueron diseñadas para sistemas homogéneos.

### Comentarios

- La recodificación de las rutinas para adaptarse a la distribución heterogénea de los datos puede ser costosa.
- Cuando se quiere ejecutar una rutina se tiene que experimentar a pequeña escala en cada procesador para conocer su capacidad de cómputo antes de hacer la distribución de los datos. Este proceso aumentará el tiempo de ejecución total de la rutina.
- No se tiene en cuenta si las prestaciones de la red de interconexión son también heterogéneas.

## 2.6.3 Proyecto PINCO

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra Lineal |
| **Plataformas de ejecución** | Plataformas paralelas heterogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Distribución del trabajo |
| **Momento de actuación** | Instalación + Ejecución |
| **Estrategia heterogénea** | Distribución heterogénea de procesos |

En el proyecto PINCO [CDG99] se diseña un sistema de distribución del trabajo entre los procesadores de diferente velocidad conectados por una red de características homogéneas. La información sobre la situación del sistema se obtiene en dos fases. En una primera fase, se miden las prestaciones estáticas de los diferentes procesadores mediante un conjunto de *benchmarks*. En una segunda fase, una serie de programas locales en cada procesador, que funcionan a modo de demonios (*local daemons*) que no introducen sobrecarga significativa en el sistema, testean dinámicamente la carga de trabajo en cada procesador y envían esta información periódicamente al *master daemon* situado en el procesador principal de la plataforma. La distribución de los datos se hace homogéneamente entre los procesos y el balanceo de la carga se realiza gracias al mapeado heterogéneo de procesos sobre los procesadores, en función de las prestaciones de los procesadores (información estática) y de la carga de trabajo (información dinámica).

Aunque este proyecto ha sido diseñado para software diverso, nuestro estudio comparativo se ha centrado en su aplicación a software de álgebra lineal. En [CDG00] han aplicado este sistema a varios algoritmos de multiplicación matricial. En [CDG01] plantean que se podrían utilizar las rutinas de la librería ScaLAPACK sobre PINCO, mediante la modificación de la rutina *Blacs_setup* de la librería de comunicaciones BLACS, que inicializa la malla virtual de procesos y los mapea sobre los procesadores, donde habría que incluir una llamada a la rutina de inicialización de PINCO y cambiar la llamada a la distribución homogénea de procesos *pvm_spawn* por la propia de PINCO, *pinco_spawn*, que distribuirá los procesos de acuerdo a las prestaciones de los procesadores.

### Comentarios

- La distribución del testeo del sistema se realiza en dos fases: una estática, general, y otra dinámica, rápida, realizada con demonios locales, sin que se sobrecargue al sistema, para detectar los posibles desbalanceos de la carga de trabajo del sistema. En nuestro planteamiento se incorpora un método semejante.
- No se tiene en cuenta si las prestaciones de la red de interconexión son también heterogéneas.
- Se basa en primitivas PVM [wPVM] para realizar las comunicaciones, en lugar de en un estándar más actual como MPI [wMPI].

### 2.6.4 Proyecto de Mills *et al.*

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra Lineal |
| **Plataformas de ejecución** | Plataformas paralelas heterogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Distribución del trabajo |
| **Momento de actuación** | Ejecución |
| **Estrategia heterogénea** | Distribución heterogénea de datos |

En la aproximación de Mills *et al*. [MSS01][CMS03][CS03] se realiza el ajuste de la carga de trabajo durante la ejecución de cada rutina. En una primera fase se obtienen los valores de las prestaciones de los distintos procesadores en base a la rapidez con que cada procesador ha llevado a cabo el trabajo que le correspondía, entonces se decide si se realiza una redistribución de los datos y, tras ello, se continúa con la ejecución de la rutina. Se han codificado diversas rutinas de resolución del problema de valores propios siguiendo este planteamiento de ajuste a las condiciones de la plataforma.

#### Comentarios

- Es necesario reprogramar las rutinas ya existentes para que sigan la metodología planteada.
- Para conocer la situación de la plataforma, los autores sugieren como vía futura el uso de herramientas como NWS. Varios proyectos han tomado esa misma decisión, entre ellos el nuestro.

### 2.6.5 Proyecto LFC

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra Lineal |
| **Plataformas de ejecución** | Plataformas paralelas homogéneas<br>Plataformas paralelas heterogéneas |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Distribución del trabajo |
| **Momento de actuación** | Ejecución |
| **Estrategia heterogénea** | Selección de procesadores |

En el proyecto LFC (*Lapack For Clusters*) [PBD+01][RD02][CDL+03] se han planteado como objetivo que un usuario no experto pueda utilizar paquetes de software de álgebra lineal para sistemas de memoria distribuida como ScaLAPACK, ejecutándolos eficientemente sobre plataformas tipo *cluster*, pero sin que este usuario tenga que tener conocimientos sobre cómo utilizar eficientemente los recursos del sistema.

El núcleo de este proyecto lo constituye una capa *middleware* situada entre el interfaz con el usuario y el software de álgebra lineal. Cuando el usuario realiza una llamada a una función de tipo LAPACK como si se encontrara en una plataforma monoprocesador, el *middleware* captura esta llamada y estudia la situación de la plataforma paralela en ese momento (de manera similar a como lo hace la herramienta NWS). A continuación, con la información de la situación de la plataforma y los datos del problema a resolver, selecciona los recursos más apropiados, distribuye los datos entre los procesadores seleccionados y pone en marcha la correspondiente rutina paralela de álgebra lineal para resolver el problema.

**Comentarios**

- Todo el proceso de ajuste se realiza cuando la rutina se va a ejecutar, con la consiguiente sobrecarga en el tiempo de ejecución de ésta.
- En cierta medida, LFC es una propuesta complementaria a la nuestra, pues ambas trabajan sobre el mismo campo: rutinas de álgebra lineal por paso de mensajes para plataformas paralelas, buscando el objetivo común de liberar al usuario de cualquier toma de decisión de cómo ejecutar la rutina que necesita. Algunas de estas decisiones son: cuántos y qué procesadores utilizar, cómo distribuir la carga de trabajo entre ellos, qué topología lógica usar (ésta afectará principalmente al coste de comunicación), cómo agrupar las operaciones algebraicas a realizar (bloque de cómputo), qué librerías básicas escoger, etc. En el proyecto LFC se han centrado en las dos primeras (la elección de los procesadores y la distribución del trabajo) y, además, han realizado importantes avances en el interfaz con el usuario. Nuestra propuesta está más orientada a la toma del resto de decisiones (la topología lógica de los procesadores, la librería básica a utilizar y el tamaño de bloque de cálculo). Partiendo de esta situación, hemos realizado un trabajo junto a los investigadores de LFC [CGG+03] en el que se estudia cómo afecta la variación de la carga de trabajo de una plataforma paralela al proceso de optimización automática de rutinas de álgebra lineal.

## 2.7 Proyectos de A. L. para plataformas distribuidas

En el campo de la computación en redes de área extensa (*grid computing*) han surgido en los último años diferentes propuestas que buscan como objetivo un uso efectivo del entorno distribuido que ofrece la red, obteniendo grandes prestaciones y, al mismo tiempo, alejando al usuario de la gran complejidad que supone cualquier acercamiento a la computación en este entorno tan heterogéneo y dinámicamente cambiante.

### 2.7.1 Proyecto GrADS

| | |
|---|---|
| **Software sobre el que se actúa** | Álgebra Lineal |
| **Plataformas de ejecución** | Entornos distribuidos en distintas redes |
| **Método de trabajo** | Teórico + Experimental |
| **Tipo de optimización** | Distribución del trabajo |
| **Momento de actuación** | Instalación + Ejecución |
| **Estrategia heterogénea** | Distribución heterogénea de procesos |

En el proyecto GrADS [PBD+01][BCC01] están trabajando varios centros de investigación de Estados Unidos que se han planteado la necesidad de un nuevo entorno de desarrollo que permita mitigar y controlar los efectos del enorme dinamismo de la red. Este entorno de desarrollo está formado por:

1. Un subsistema de preparación de programas donde se genera, a partir de librerías con capacidad de auto-adaptación (siguiendo la metodología propuesta en el proyecto SANS anteriormente descrito), un objeto donde queda encapsulada la aplicación a ejecutar (*configurable object program*, *COP*). Este objeto se podrá optimizar rápidamente en tiempo de ejecución de acuerdo al conjunto de recursos de la red que se seleccionen.
2. Un subsistema de control del entorno de ejecución encargado de determinar los recursos disponibles en la red (apoyándose en alguna herramienta como NWS), seleccionar los más adecuados y entonces poner en marcha el *COP* en estas condiciones. Además durante la

ejecución se encargará de monitorizar este proceso para observar si se siguen cumpliendo las prestaciones previstas cuando empezó la ejecución (*performance contract*), y en caso de que no se cumplan, se pueda renegociar la situación realizando modificaciones en el *COP* y/o cambios en la selección de recursos de la Red para continuar la ejecución.

**Comentarios**

Para conocer la situación de la plataforma se utiliza la herramienta NWS. Varios proyectos para plataformas heterogéneas y para distribuidas han tomado esa misma decisión, entre ellos el nuestro.

## 2.8 Comparativa de los trabajos

En la Tabla 2.1 se muestran de manera esquemática las características de todos los proyectos anteriormente descritos, junto a las de nuestro propio trabajo. En primer lugar, en relación a las propuestas que actúan, al igual que la nuestra, sobre software paralelo de álgebra lineal, podemos encontrar algunas diferencias:

- Por un lado, encontramos propuestas (PHiPAC, ATLAS, ITXGEMM, ILIB, Zhang *et al.* y Mills *et al.*) que actúan realizando ajustes "a medida" para cada rutina. Mientras que en otras, incluida la nuestra, se plantea un metodología general para todas las rutinas.
- Por otro lado, en algunas propuestas (LAWRA, *Multicomputer Toolbox*, mpC, Robert *et al.* y Mills *et al.*) es necesaria una recodificación total o parcial de las rutinas originales. Sin embargo, en la mayor parte de los proyectos, entre ellos el nuestro, no se tiene que modificar el software ya existente.

En cuanto a la plataforma de destino:

- En las propuestas orientadas a máquinas secuenciales el esfuerzo de ajuste de las rutinas se suele centrar en mejorar el tiempo de acceso a los diferentes niveles de la jerarquía de memoria, analizándose los esquemas de almacenamiento y los patrones de acceso a los datos de las rutinas (bloques de cálculo).
- En aquellas orientadas a plataformas paralelas homogéneas, además de mejorar los accesos a los datos locales, cobra gran importancia optimizar el coste de las comunicaciones entre los procesadores, con lo que los métodos de autoajuste dedican gran parte de su labor a la búsqueda de las topologías lógicas de los procesadores y los patrones de comunicación que disminuyan este coste.
- En los entornos heterogéneos, debido a las diferentes potencias de cómputo y de comunicación de sus nodos y redes de interconexión, la principal labor a realizar es la búsqueda de la distribución óptima del trabajo entre los procesadores. La estrategia para este reparto del trabajo puede incluir una distribución heterogénea de los datos, que supondrá recodificar totalmente las rutinas (mpC, Robert *et al.* y Mills *et al.*). Sin embargo, si el reparto del trabajo se hace mapeando heterogéneamente los procesos sobre los procesadores la recodificación será mínima, como ocurre en PINCO. Por último, si únicamente se realiza una selección de procesadores a utilizar de entre los de mayor capacidad de cómputo y de comunicación, no será necesario ninguna recodificación, como ocurre en LFC y en nuestro proyecto.

En relación a la técnica utilizada para ajustar las rutinas, existen tres posibilidades generales:

- La primera, más teórica, consiste en realizar modelos analíticos detallados que describan las rutinas, para, posteriormente, aplicar técnicas de optimización teóricas a estos modelos (BeBOP y *Multicomputer Toolbox*).
- La segunda, que se basaría en una labor experimental, consiste en la realización de un repertorio de pruebas de ejecución de toda o parte de la rutina que se pretende optimizar, probando con las diferentes posibilidades que dicha rutina admita, hasta comprobar cuál de éstas es la más conveniente (Vadishar *et al.*, PHiPAC, ATLAS, ITXGEMM e ILIB).
- El resto de trabajos, entre ellos el nuestro, aúnan las dos técnicas anteriores (modelado analítico y estudio experimental) aprovechando las mejores características de cada una de ellas.

| **PROYECTO** | **Software** | **Plataformas Hardware** | **Estrategia heterogénea** | **(T)eórico / (E)xperimental** | **Tipo Optimización** | **Momento : (I)nstalación / (E)jecución** |
|---|---|---|---|---|---|---|
| Brewer | General | Paral. Homo. | | T + E | Parametrización Polialgoritmos | I |
| Juhász&Charaf | General | Paral. Homo. | | T + E | Parametrización | I |
| SPIRAL | Proc. Señales | Secuenciales | | T + E | Polialgoritmos | I |
| FFTW | FFT | Secuenciales | | T + E | Polialgoritmos | I + E |
| SALT | Proc. Señales | Paral. Hetero. Distribuidas | Dist. procesos | T + E | Polialgoritmos Dist. del trabajo | I + E |
| Vadishar | MPI | Paral. Homo. | | E | Parametrización Polialgoritmos | I + E |
| GTP | General | Paral. Homo. | | T + E | Parametrización Polialgoritmos Dist. del trabajo | E |
| LAWRA | A. L. | Secuenciales | | T + E | Parametrización | E |
| Núcleos MM: PHiPAC ATLAS ITXGEMM LAW-GEMM | A. L. | Secuenciales | | E | Parametrización Polialgoritmos | I + E |
| BeBOP | A. L. | Secuenciales | | T | Parametrización Polialgoritmos | E |
| *Multicomputer Toolbox* | A. L. | Paral. Homo. | | T | Polialgoritmos Polilibrerías | E |
| ILIB | A. L. | Paral. Homo. | | E | Parametrización | I |
| Zhang | A. L. | Paral. Homo. | | T + E | Parametrización | I |
| mpC | A. L. | Paral. Hetero. | Dist. procesos + Dist. datos | T + E | Dist. del trabajo | I + E |
| Robert | A. L. | Paral. Hetero. | Dist. datos | T + E | Dist. del trabajo | E |
| PINCO | A. L. | Paral. Hetero. | Dist. procesos | T + E | Dist. del trabajo | I + E |
| Mills | A. L. | Paral. Hetero. | Dist. datos | T + E | Dist. del trabajo | E |
| LFC | A. L. | Paral. Hetero. | Sel. procesadores | T + E | Dist. del trabajo | E |
| Distribuidas: GRADS | A. L. | Distribuidas | Dist. procesos | T + E | Dist. del trabajo | I + E |
| Nuestra Propuesta | A. L. | Paral. Hetero. | Sel. procesadores | T + E | Parametrización Polialgoritmos Polilibrerías Dist. del trabajo | I + E |

Tabla 2.1. Comparativa de los diferentes proyectos.

En cuanto al objetivo concreto perseguido en el proceso de autoajuste podemos encontrar tres grupos de proyectos:

- Aquellos con un solo objetivo: obtener los mejores valores para una serie de parámetros ajustables, elegir qué procesadores utilizar, balancear el trabajo entre los nodos, elegir el algoritmo concreto a utilizar entre varios posibles o bien escoger de entre varias librerías disponibles la versión de la rutina más apropiada para ser ejecutada.
- Otro grupo lo forman aquellos proyectos que persiguen varios objetivos de optimización, realizando un estudio por separado para cada uno.
- Por último, tanto en GTP como en nuestra propuesta, todas las optimizaciones perseguidas se materializan mediante una metodología unificada basada en el modelo analítico del tiempo de ejecución de la rutinas, en el que se introducen las características del sistema que le puedan afectar. De esta manera se puede realizar una búsqueda global en todo el espacio de factores ajustables de las rutinas.

Con referencia al momento en que se lleva a cabo el ajuste de las rutinas:

- Algunos proyectos realizan el ajuste durante la instalación del software, basando su labor únicamente en una visión estática de la plataforma de ejecución. Estos proyectos no tienen en cuenta una información tan útil como qué carga de trabajo existe en las diferentes partes del sistema cuando la rutina se va a ejecutar.
- Otros proyectos llevan a cabo el ajuste a la hora de ejecutar cada rutina o incluso a lo largo de la propia ejecución. De esta manera se puede tener en cuenta cualquier cambio que se produzca en el sistema. El problema de estos proyectos es que se introduce una sobrecarga no deseada en el tiempo de ejecución total.
- Algunas propuestas, como la nuestra, realizan el ajuste de una manera mixta, una parte durante la instalación y otra parte en el momento de ejecutar la rutina. En concreto, nuestra propuesta realiza la parte más costosa del proceso (captura de las condiciones estáticas de la plataforma) durante la instalación, dejando para el momento de ejecutar la rutina una labor menos costosa (ajustar la información que se obtuvo durante la instalación conforme a la carga de trabajo que en ese momento esté soportando el sistema). Esta última parte del proceso, que se realiza en tiempo de ejecución, introduce una mínima sobrecarga ya que no se realiza ningún tipo de experimentación para conocer la situación de la plataforma, sino que se consulta la información que aporta la herramienta NWS, de manera semejante a como se realiza en los proyectos de Mills *et al.*, SALT, LFC y GrADS.

## 2.9 Resumen

En este capítulo se han descrito una serie de propuestas que tienen como objetivo general el ajuste automático de software de acuerdo a las condiciones del entorno de ejecución, incluyendo en este entorno tanto las características de la plataforma hardware como del problema a resolver. La mayor parte de las propuestas vistas están centradas en el estudio de rutinas de álgebra lineal que es donde está enmarcado nuestro trabajo, pero también se han mostrado algunos proyectos que actúan sobre otro tipo de software. Aunque se han mostrado algunos trabajos que centran su esfuerzo en rutinas para plataformas secuenciales, se ha hecho más hincapié en propuestas más generales, válidas para cualquier tipo de plataformas paralelas. Por último, también se han descrito algunas aproximaciones para plataformas de naturaleza heterogénea y distribuida. Finalmente, en la última sección del capítulo se ha mostrado una comparativa entre los diferentes trabajos descritos, incluyendo referencias a la labor desarrollada en nuestra propuesta.

Como se describirá en los siguientes capítulos, nuestro proyecto abarca los diferentes tipos de optimizaciones mostrados (parametrización, distribución del trabajo, polilibrerías y polialgoritmos) de una manera unificada, a partir del modelo analítico del tiempo de ejecución de cada rutina, que constituye el pilar básico de toda la metodología. Este modelo tiene una naturaleza teórico-experimental, pues su diseño inicial se hace mediante un estudio de complejidad de la rutina, pero los parámetros del sistema que en él aparecen se calculan de manera empírica en tiempo de instalación. Posteriormente, en tiempo de ejecución, estos parámetros son ajustados teniendo en cuenta la carga de trabajo de los diferentes nodos y de la red de interconexión, con lo que el modelo mantiene en todo momento una imagen realista del comportamiento de la rutina sobre la plataforma, permitiendo tomar las oportunas decisiones de optimización.

# Capítulo 3. Entorno de Trabajo

## 3.1 Introducción

Como ya se ha comentado en los dos primeros capítulos, el principal objetivo de esta tesis es desarrollar una metodología general de ajuste automático de rutinas paralelas de álgebra lineal que estén programadas con paso de mensajes para plataformas paralelas genéricas. En este capítulo se describirá el marco de trabajo en donde se ha llevado cabo esta labor de investigación, esto es, plataformas paralelas de propósito general y software de álgebra lineal densa.

En primer lugar, se mostrará una visión general de las arquitecturas paralelas actuales. A continuación, se detallarán las características concretas de las plataformas que se han utilizado para llevar a cabo los experimentos de este trabajo: un multiprocesador de memoria compartida distribuida, un multiprocesador de memoria distribuida, un *cluster* de PCs y un *cluster* de estaciones de trabajo.

En cuanto al entorno software de trabajo, en la última sección de este capítulo, daremos un repaso al software utilizado en la computación numérica paralela. Se mostrará el software básico para la compartición y comunicación de datos entre procesadores, las librerías clásicas de software de álgebra lineal y, finalmente, algunas herramientas software auxiliares que son de gran utilidad en este marco de trabajo.

## 3.2 Arquitecturas Paralelas

Una clasificación clásica de las plataformas paralelas fue propuesta por Flynn [Fly66]. En esta clasificación, las plataformas monoprocesador son llamadas sistemas SISD (*Single Instruction stream, Single Data stream*), es decir, un flujo único de instrucciones (programa) que actúan sobre una única secuencia o conjunto de datos. Un segundo grupo de plataformas serían los sistemas SIMD (*Single Instruction stream, Multiple Data stream*), donde un único flujo de instrucciones actúa sobre múltiples datos. El tercer tipo de plataformas abarcaría a aquellas que cuentan con un grupo de procesadores independientes, cada uno ejecutando un programa diferente y accediendo a su propio conjunto de datos. Estas últimas, que son llamadas MIMD (*Multiple Instruction stream, Multiple Data stream*), vienen a constituir hoy en día la amplía mayoría de plataformas paralelas de propósito general existentes, pudiendo emular a cualquiera de los otros tipos de plataformas paralelas. Por último, en las plataformas MISD (*Multiple Instruction stream, Single Data stream*) un conjunto único de datos es manipulado por procesadores separados que ejecutan su propio programa. En este trabajo nos centraremos en las plataformas paralelas de propósito general o MIMD, donde, dependiendo de la visión de la memoria de la máquina que perciben los procesadores, podemos encontrar dos tipos genéricos [WA99]: multiprocesadores de memoria compartida y multiprocesadores de memoria distribuida.

En los multiprocesadores de memoria compartida existe un único espacio de direcciones de memoria que es compartido entre todos los procesadores. En esta memoria única existe código ejecutable diferente para cada uno de los procesadores, así como los datos con los que operará cada uno de ellos. Las comunicaciones entre los procesadores se realizan implícitamente haciendo uso de este espacio de memoria compartida. Existen dos posibles arquitecturas generales para estas plataformas: Multiprocesadores Simétricos (*Symetric MultiProcessors*, SMP) y Multiprocesadores de Memoria Compartida Distribuida (*Distributed Shared Memory*, DSM). En los primeros existe un gran espacio de memoria formado por un conjunto de módulos a los que los procesadores pueden acceder a través de una red de interconexión compartida entre todos (Figura 3.1). Todos los procesadores tienen el mismo tiempo de acceso a cualquier posición de la memoria, por lo que estas plataformas se conocen también como multiprocesadores UMA (*Uniform Memory Access*). En los segundos la memoria se encuentra físicamente distribuida entre los diferentes procesadores, con lo que, aunque se mantiene la lógica de un espacio de direcciones único para todos los procesadores, el tiempo de acceso a las diferentes posiciones de la memoria es diferente si las posiciones referenciadas por un procesador se encuentran en su propio nodo o en el de otro (Figura 3.2). Por esta razón, estas últimas plataformas se conocen también como multiprocesadores NUMA (*Non Uniform Memory Access*).

Por otro lado, en los multiprocesadores de memoria distribuida, cada nodo está formado por un procesador completamente independiente, que cuenta con su propio espacio de direcciones localizado físicamente en su memoria local (Figura 3.3). Las comunicaciones entre los procesadores se realizan explícitamente mediante el paso de mensajes, donde, a lo largo de la ejecución de cada programa, los datos se envían de unos procesadores a otros que los requieren para su computación. Por esta razón estas plataformas se conocen también como multiprocesadores de paso de mensajes o, simplemente, multicomputadores.

Además de estas plataformas paralelas concretas y tangibles anteriormente descritas, existen en la actualidad otra serie de plataformas paralelas virtuales como pueden ser los *clusters*, las plataformas heterogéneas y las plataformas distribuidas. Un *cluster* consiste básicamente en un conjunto de ordenadores independientes conectados por una red local dedicada. Si esta red de interconexión proporciona buenas prestaciones, el comportamiento de un *cluster* puede ser bastante similar al de un multiprocesador de memoria distribuida [Wu99]. Por otro lado, las plataformas heterogéneas pueden estar formadas por un conjunto de *clusters*, ordenadores independientes, multiprocesadores, etc, que están conectados a través de diferentes redes de interconexión. En estas plataformas se suele combinar técnicas distintas de programación (paso de mensajes y compartición de variables) acorde a su naturaleza heterogénea. Por último, las plataformas distribuidas vienen a ser una extensión de las heterogéneas, incrementándose notablemente la distancia física y la heterogeneidad entre los diferentes nodos que la forman, con el consiguiente aumento del coste de las comunicaciones. Debido al potencial de cómputo que ofrecen estas plataformas distribuidas, así como a la complejidad de su gestión, programación y utilización, está surgiendo en los últimos años todo un nuevo campo de investigación y desarrollo llamado *grid computing* cuyo objetivo general se podría resumir en construir un conjunto de redes de computación regionales, nacionales e internacionales, que provean poder de computación a los usuarios de la misma manera que las redes eléctricas proveen energía a nuestros hogares [And00].

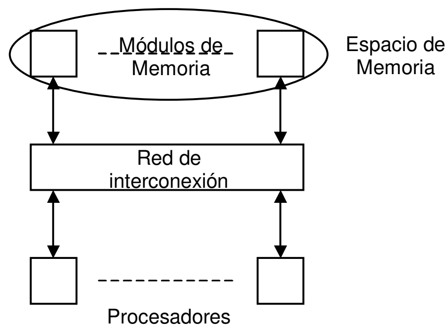


Figura 3.1. Multiprocesador Simétrico (SMP).

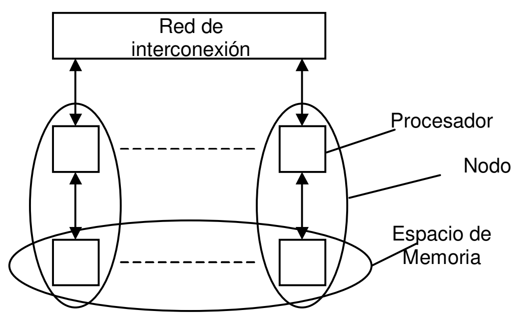


Figura 3.2. Multiprocesador de Memoria Compartida Distribuida (DSM)

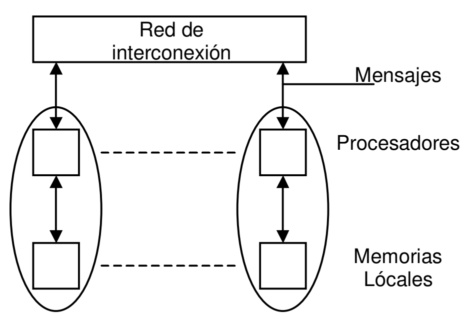


Figura 3.3. Multiprocesador de Memoria Distribuida

Por último, remarcar que en los últimos años las plataformas de memoria distribuida programadas por paso de mensajes (multicomputadores, *clusters* y plataformas distribuidas) se están convirtiendo en las protagonistas dentro del campo de la computación paralela de altas prestaciones, acaparando la práctica totalidad de la capacidad de cómputo global, como podemos observar en la lista TOP500 [wTOP] (Figura 3.4). Además, el software paralelo por paso de mensajes puede utilizarse también en multiprocesadores de memoria compartida de manera eficiente, lo que hace que nuestra propuesta sea válida para la gran mayoría de los sistemas actuales de altas prestaciones.

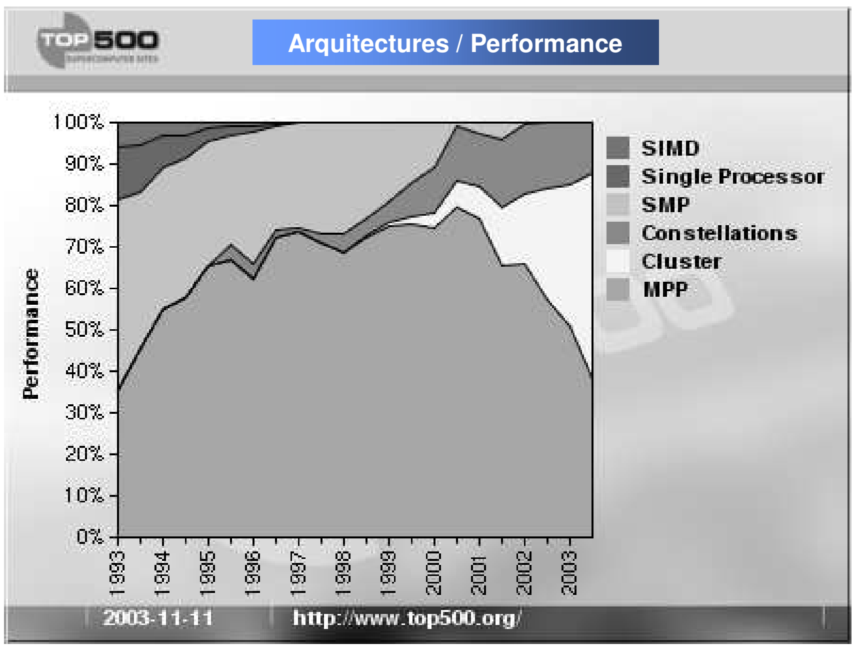


Figura 3.4. Distribución por tipo de arquitectura de la potencia de cómputo de las plataformas incluidas en el TOP500. (http://www.top500.org)

## 3.3 Plataformas utilizadas en nuestros experimentos

Dentro del conjunto de plataformas paralelas genéricas donde pretendemos comprobar la validez de las propuestas de este trabajo existen diferencias significativas en cuanto a:

- Prestaciones de computo, tanto globales como locales en cada uno de sus nodos.
- Características de la red de interconexión: latencia, ancho de banda, grado de interconectividad entre los nodos, etc.
- Ratio entre la capacidad de cómputo de los nodos y capacidad de las comunicaciones de la red de interconexión.

- Características de memoria: tamaño, organización jerárquica, distribución entre los nodos, etc.
- Gestión de los recursos: modos de utilización por parte de los usuarios (multiusuario / monousuario), sistemas de control de la carga de trabajo en la plataforma, etc.

Por esta razón, para llevar a cabo la fase experimental de este trabajo, se ha escogido un conjunto heterogéneo de plataformas según esas características. Las plataformas utilizadas han sido:

- Plataforma Karnak: un multiprocesador SGI Origin 2000, de memoria compartida físicamente distribuida.
- Plataforma Besiberri: un multiprocesador IBM-SP2, de memoria distribuida.
- Plataforma TORC: un *cluster* de PCs.
- Plataforma COCI: un *cluster* de estaciones de trabajo.

Como se muestra en Figura 3.4, esta selección cubre el espectro de sistemas de computación paralela más relevantes de la actualidad. En los siguientes apartados se irán describiendo las características de cada una de estas plataformas.

## 3.3.1 Plataforma Karnak

El SGI Origin 2000 es un multiprocesador con una arquitectura de memoria compartida físicamente distribuida, provisto de un sistema de coherencia de caché (*cache coherence Non Uniform Memory Access*, ccNUMA). Puede tener hasta 128 procesadores, admitiendo programación tanto del modelo de memoria compartida como del modelo de paso de mensajes [wSGI][LL97]. El bloque más elemental de su arquitectura es un nodo SN0 (Figura 3.5). En cada uno de estos nodos encontramos:

- 2 procesadores R10000. Este procesador RISC de 64 bits, 250 MHz, superescalar de 4 vías, cuenta con una caché de primer nivel de datos de 32 KB y otra de instrucciones de 32 KB.
- 2 memorias caché de segundo nivel de 4MB, cada una acoplada a un procesador.
- Memoria RAM de 8 GB.
- Directorios para el sistema de coherencia de caché, de 4 GB.
- 1 *hub* que controla el tráfico entre los procesadores, la memoria RAM y la E/S del nodo.

Para cada dos nodos de la arquitectura existe un *router* cuya misión es encaminar los mensajes entre nodos diferentes. Cada *router* está conectado a través de la red *CrayLink Interconnect* de 160 GB/s con el resto de *routers* del sistema, formando una estructura de hipercubo (Figura 3.6). Los puertos para la memoria y los procesadores son bidireccionales, mientras que los de entrada/salida y conexión con el *CrayLink* son *half-duplex*, operando todos los puertos a 780 MB/s.

La plataforma sobre la que se han realizado los experimentos de este trabajo es Karnak (karnak.cepba.upc.es), perteneciente al CEPBA (Centro Europeo de Paralelismo de Barcelona) [wCEPBA] de la Universidad Politécnica de Cataluña. Esta máquina consta de 64 procesadores, 360 GB de capacidad de almacenamiento en disco y con un pico máximo de prestaciones de 32 GFlops. El sistema operativo instalado es IRIX 64.

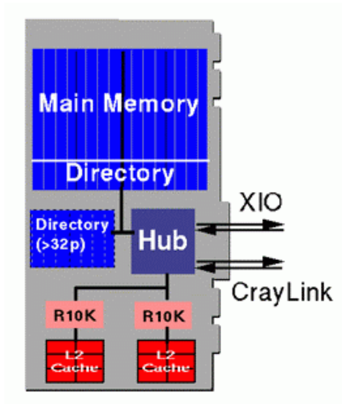


Figura 3.5. Esquema de un nodo SN0 de la arquitectura del SGI Origin 2000.
(http://www.sgi.com/)

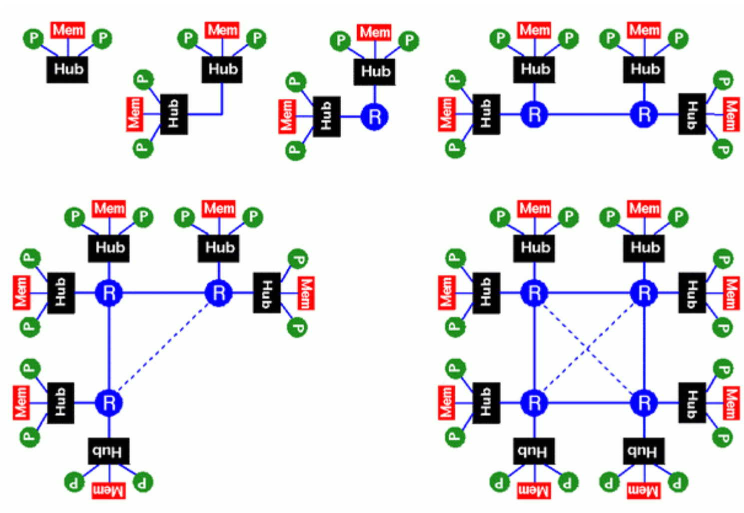


Figura 3.6. Esquema de posibles arquitecturas del SGI Origin 2000.
(http://www.sgi.com/)

### 3.3.2 Plataforma Besiberri

El IBM-SP2 [AMM+95] es un sistema multiprocesador con alto grado de escalabilidad pues está basado en una arquitectura de memoria distribuida siguiendo el modelo de paso de mensajes. Normalmente los sistemas SP2 pueden tener de 2 a 128 nodos, aunque hay sistemas de hasta 512 nodos. Para los nodos se utiliza el procesador superescalar POWER2 de tecnología RISC. En cada nodo existirá una caché de primer nivel de hasta 256 KB, una caché de segundo nivel de hasta 512 KB y una memoria principal local al nodo de hasta 2 GB (Figura 3.7).

Las comunicaciones entre nodos se realizan a través de una red de interconexión de conmutación de paquetes: *High-Performance Switch*, diseñada para ser escalable, tener baja latencia y un alto ancho de banda, y para disminuir la sobrecarga del procesador a la hora de realizar una comunicación con otro nodo. La topología del *switch* es una red indirecta multietapa de conmutación de paquetes que permite una conexión entre cualquier par de nodos. Gracias a esta topología, el ancho de banda entre un par de nodos permanece constante independientemente de la situación física de éstos, y además, el ancho de banda biseccional escala linealmente [GH01]. Los elementos básicos de esta red son chips *crossbar* de 8 puertos bidireccionables. Mediante 4 de estos chips configurados en 2 etapas se obtienen tarjetas de 32 puertos bidireccionables. En la Figura 3.8 se muestra cómo se conectarían 64 nodos mediante cuatro de estas tarjetas. Cada puerto permite un ancho de banda máximo de 40 MB/s en cada dirección.

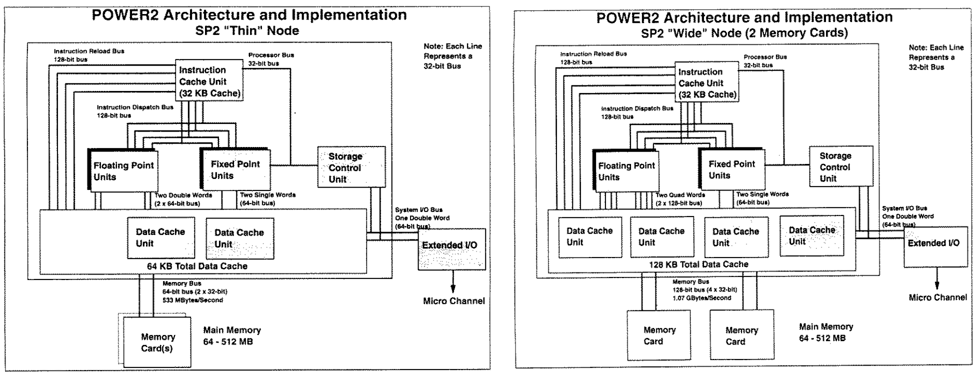


Figura 3.7. Estructura de un nodo thin y nodo wide del IBM-SP2.
(http://www.research.ibm.com/journal/)

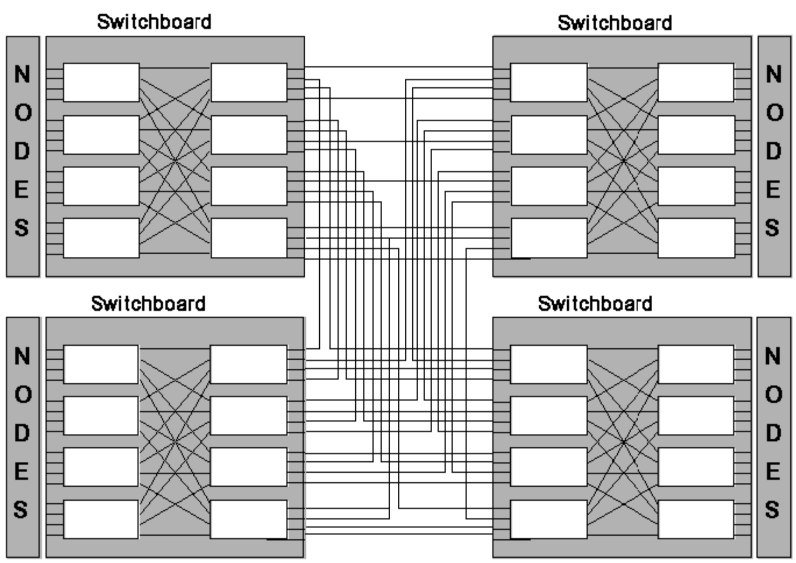


Figura 3.8. Estructura del switch para un sistema SP2 de 64 nodos.
(http://www.research.ibm.com/journal/)

El sistema donde se han realizado los experimentos de este trabajo es la máquina Besiberri (besiberri.sp.cesca.es) del CESCA (Centro de Supercomputación de Cataluña) [wCESCA] de la Universidad Politécnica de Cataluña. Sus características son:

- 42 nodos thin160, a 160 MHz, con 128 KB de caché de primer nivel y con 256 MB de memoria principal. Con unas prestaciones pico de 640 MFLOPS por nodo.
- 2 nodos wide, a 66 MHz, con 256 KB de caché de primer nivel, uno con 512 MB de memoria principal y otro de 1024 MB. Con unas prestaciones pico de 266 MFLOPS por nodo.

Las características globales de esta máquina son: potencia punta de 27.41 GFLOPS, memoria principal de 12 GB y disco de 494 GB.

### 3.3.3 Plataforma TORC

La plataforma TORC (*Tennessee Oak Ridge Cluster*) es un *cluster* de PCs perteneciente al Laboratorio ICL [wICL] de la Universidad de Tennessee y al *Oak Ridge National Laboratory*. Las características más relevantes de la parte de esta plataforma que hemos utilizado en nuestros experimentos son:

- Nodos de computación:
  - 10 CPUs Pentium III dual, a 550 MHz.
  - Memoria caché de segundo nivel de 512 KB.
  - Memoria Principal de 512 MB.
  - Sistema operativo Linux Redhat 7.2.
- Red de interconexión: Fast-Ethernet (100 mb/s) de uso dedicado, con un switch de 2×16 puertos.

Para algunos experimentos puntuales también hemos utilizado otra parte de esta plataforma: 2 nodos Pentium 4 a 1.5 GHz y una red Myrinet con un switch de 8 puertos.

### 3.3.4 Plataforma COCI

La plataforma COCI está formada por cinco estaciones de trabajo Sun Ultra 1 [wSun] y una estación Sun Ultra 5, conectadas con una red local Ethernet (10 mb/s) de uso dedicado. Esta plataforma pertenece al Grupo de Investigación de Computación Científica [wCOCI] de la Universidad de Murcia.

Cada estación Sun Ultra 1 tiene las siguientes características:

- CPU: 143MHz UltraSPARC.
- Caché de primer nivel de 32 KB.
- Caché de segundo nivel de 1 MB.
- Memoria Principal de 64 MB.
- Sistema Operativo Solaris 5.5.1.

La estación Sun Ultra 5 tiene las siguientes características:

- CPU: 360 MHz UltraSPARC.
- Caché de primer nivel de 32 KB.

- Caché de segundo nivel de 2 MB.
- Memoria Principal de 128 MB.
- Sistema Operativo Solaris 5.7.

### 3.3.5 Comparativa de las plataformas

De cara a tener una visión inicial de las diferentes capacidades de estos sistemas, en la Tabla 3.1 se muestran las prestaciones obtenidas experimentalmente en cada plataforma, con una librería básica de cálculo y con MPI como librería de comunicaciones. En primer lugar, se muestra la capacidad máxima de cómputo obtenida en cada procesador con la rutina de multiplicación matricial *GEMM* de la librería básica instalada, que puede ser una versión de BLAS implementada para la plataforma (mac-BLAS) o la librería auto-optimizada ATLAS. A continuación, se expone la capacidad máxima de comunicaciones punto a punto, obtenida con una rutina *ping-pong* implementada con MPI.

| Sistema | | CPU (Mflops) | Ancho de banda (Mbytes/s) | Latencia (μs) |
|---|---|---|---|---|
| Karnak | mac-BLAS | 400 | 80 | 20 |
| Besiberri | mac-BLAS | 555 | 26 | 75 |
| TORC<br>PIII + FastEthernet | ATLAS | 333 | 12 | 60 |
| COCI<br>Sun1 | mac-BLAS | 90 | 1.2 | 170 |

Tabla 3.1. Prestaciones obtenidas experimentalmente en cada plataforma.

Observando estas características de las prestaciones de computo y comunicaciones, vemos que Karnak es la plataforma que tiene el menor ratio comunicación-computación de entre las cuatro usadas para la experimentación, es decir, es la que tiene las comunicaciones más rápidas en relación a la computación. Por su parte, Besiberri es la plataforma que tiene la mayor capacidad de cómputo, pero con menor velocidad de comunicación que Karnak. En cuanto a los *clusters*, TORC tiene un ratio comunicación-computación similar a Besiberri, aunque con valores mucho menores de estas capacidades. Finalmente, COCI es la que tiene la menor capacidad de cómputo pero, sobre todo, de comunicaciones, ya que tiene un ancho de banda que es tan solo la décima parte de la que se alcanza en la plataforma TORC.

En cuanto al control de la carga de trabajo, tanto Karnak como Besiberri cuentan con un sistema de colas de trabajos de cara a optimizar el uso de sus recursos y balancear la carga de trabajo que la máquina soporte en cada momento. En el extremo opuesto tenemos a TORC, que no cuenta con ningún control sobre el número de usuarios, de procesos o de la carga de trabajo que se soporta en un momento dado. Por esta razón, en TORC suelen darse grandes variaciones de la carga de trabajo y, además, de manera heterogénea. Finalmente, la plataforma COCI, al igual que le ocurre a TORC, no cuenta con ningún control sobre el número de usuarios, de procesos o la carga de trabajo que se soporta un momento dado, pero, a diferencia de ésta, el número de usuarios es muy reducido, con lo que se puede llevar un control directo de la carga de trabajo del sistema, siendo habitual que el usuario que lo precise pueda tener un uso exclusivo de la plataforma.

## 3.4 Software para Cálculo Numérico Paralelo

### 3.4.1 Software de Comunicaciones

De cara a comunicar o compartir información entre procesadores de plataformas paralelas existen dos metodologías generales que se pueden seguir: el paradigma de paso de mensajes y el paradigma de compartición de variables. Habitualmente se dice que el paradigma de programación mediante paso de mensajes es menos atractivo para los programadores que el paradigma de compartición de variables, típico de los multiprocesadores de memoria compartida. Esto se debe a que la introducción de código para enviar y recibir mensajes en los programas no es una labor sencilla y suele conllevar muchos errores durante el desarrollo de este software. Sin embargo, el paradigma de paso de mensajes tiene la ventaja de que no se necesitan mecanismos especiales para controlar accesos simultáneos a los datos. Estos mecanismos de acceso exclusivo pueden provocar un aumento significativo en el tiempo de ejecución de los programas [SFK97]. Otra ventaja del paradigma de paso de mensajes es que mantiene un alto grado de eficacia en cualquier tipo de plataforma [GGK+03] (en multiprocesadores de memoria distribuida, en multiprocesadores de memoria compartida usando partes de la memoria compartida para mantener los datos que son enviados de un procesador a otro, en *clusters*, en plataformas heterogéneas e, incluso, en plataformas distribuidas). El estándar de programación más extendido dentro del paradigma de compartición de variables es OpenMP [DM98][wOMP], mientras que los dos estándares más conocidos y usados en el paradigma de paso de mensajes son PVM [wPVM][GBD+96] y MPI [SOH+96][wMPI].

OpenMP (*Open Multi Processing*) está formado por un conjunto de directivas de compilación y por una librería de rutinas. Con estas directivas y rutinas se extienden lenguajes tradicionales como Fortran, C o C++, incorporándoles la capacidad para expresar paralelismo en memoria compartida desde un punto de vista más a alto nivel que mediante el uso directo de APIs de *threads* como, por ejemplo, el estándar de *threads* POSIX, Pthreads [wPthre]. En la actualidad un gran número de productores de software tienen desarrolladas diversas herramientas para OpenMP, incluyendo compiladores, herramientas de desarrollo, herramientas de análisis de prestaciones, etc. [And00].

PVM (*Parallel Virtual Machine*) es un sistema software diseñado para que se pueda usar un conjunto de computadores heterogéneos como un único recurso computacional, es decir, como si fuese un multiprocesador. Los distintos computadores, que pueden ser de muy diferente naturaleza, estarán conectados por una red cualquiera.

MPI (*Message Passing Interface*) es una especificación estándar para librerías de comunicaciones con paso de mensajes. Este estándar define el interfaz con el usuario y la funcionalidad de un amplio rango de capacidades de paso de mensajes. El principal objetivo de MPI es la portabilidad del código de paso de mensajes entre plataformas de muy diferente naturaleza (multiprocesadores de memoria distribuida, multiprocesadores de memoria compartida, redes de computadores, ...), manteniendo un alto grado de eficiencia en todas ellas. Se pueden encontrar multitud de implementaciones de MPI, siendo las más difundidas MPICH [wMPICH] del Laboratorio Nacional de Argonne y la Universidad Estatal de Mississippi, LAM [wLAM] de la Universidad de Ohio, así como las realizadas explícitamente para plataformas concretas [wMPIi].

Tanto en PVM como en MPI no se detallan cómo son realizadas las diferentes operaciones de comunicación, tan sólo se especifican de manera lógica. Ambos entornos incluyen, además de las operaciones de comunicación punto a punto, un conjunto de operaciones colectivas: barreras de sincronización, difusión (*broadcast*), distribución (*scatter*), recolección (*gather*), así como diversas operaciones de reducción globales (suma, mínimo, máximo, ...). MPI además incluye operaciones de multidifusión (*multicast*). Algunas de las principales diferencias entre PVM y MPI serían [wZha02][GL97]:

- **Control de Procesos**: PVM tiene la capacidad de iniciar y finalizar procesos en cualquier momento de la ejecución de un programa. Esta capacidad no existía en el estándar de MPI, pero se ha incorporado a MPI-2 [wMPI2]. En PVM se realiza este proceso directamente, mientras que en MPI-2 se le encarga esta labor al sistema operativo distribuido que exista en la plataforma.
- **Control de recursos**: En PVM se pueden añadir o eliminar un recurso en cualquier momento de la ejecución, mientras que en MPI no se puede hacer ninguna de estas modificaciones.
- **Topología**: En MPI se pueden agrupar tareas en una topología lógica de interconexión específica, realizándose la comunicación entre estas tareas dentro de este subgrupo. En PVM esta labor de agrupación de tareas la tiene que realizar el programador manualmente.
- **Tolerancia a fallos**: PVM incluye un esquema básico de notificación de fallos a las tareas, mientras que en el estándar MPI no existe ningún tipo de mecanismo de este tipo, dejándose la decisión de incluirlos a los implementadores de MPI para cada plataforma [wFTM].
- **Prestaciones**: Las prestaciones que se pueden obtener con implementaciones de MPI suelen ser mucho mejores que con PVM.

## 3.4.2 Software de Álgebra Lineal

El software de álgebra linear es el núcleo sobre el que se construye, hoy en día, la mayor parte del software utilizado en el campo de la computación científica [DDS+98]. En los últimos 20 años, el trabajo en el desarrollo de software de álgebra lineal ha venido motivado por la necesidad de resolver problemas de gran tamaño y usando los computadores más rápidos de los que se disponga. Con esta motivación, se vio la necesidad de identificar las operaciones básicas necesarias y unificarlas en un estándar *de facto*. Este estándar lo constituye la librería BLAS (*Basic Linear Algebra Subprograms*) [wBLAS], de la que existen implementaciones de gran eficiencia para la mayoría de los computadores de arquitecturas avanzadas existentes. BLAS está dividida en tres niveles:

- Nivel 1 de BLAS [LHK+79]: operaciones vectoriales, es decir, O($n$) operaciones sobre O($n$) datos.
- Nivel 2 de BLAS [DDH+88]: operaciones matriz-vector, es decir, O($n^2$) operaciones sobre O($n^2$) datos.
- Nivel 3 de BLAS [DDD+90]: operaciones matriz-matriz, es decir, O($n^3$) operaciones sobre O($n^2$) datos.

Usando BLAS como base, se diseño una librería de un nivel superior, LAPACK [Dem89][ABB+99][wLAPAC], que incluye resolución de sistemas de ecuaciones, problemas de mínimos cuadrados, problemas de valores propios y problemas de valores singulares. Además esta librería incorpora las factorizaciones de matrices LU, QR, Cholesky, SVD, Schur y Schur generalizada. Las matrices operando que manejan estas rutinas pueden ser matrices densas o matrices banda, aunque, en general, no tienen capacidad para operar con matrices escasas. En todas estas áreas, LAPACK proporciona una funcionalidad similar para matrices reales y complejas, en simple y doble precisión.

LAPACK surgió con el propósito de unificar y hacer más eficientes, robustas, precisas y funcionales dos librerías anteriormente existentes: EISPACK [wEISP] y LINPACK[wLINP]. El principal problema de estas librerías era el patrón de acceso a los datos que seguían sus rutinas, ya que estaban basadas en operaciones vectoriales del nivel 1 de BLAS. LAPACK mejoró esta ineficiencia reorganizando los algoritmos de cara a usar operaciones orientadas a bloques de matrices, con lo que aumenta la localidad de acceso a los datos, utilizando operaciones del nivel 3 de BLAS. Estas operaciones por bloques pueden ser optimizadas para cada arquitectura si se tienen en cuenta las dimensiones de los distintos niveles de su jerarquía de memoria.

Las rutinas de LAPACK se clasifican en:

- **Rutinas *Drivers***: Resuelven un problema completo, por ejemplo, un sistema linear de ecuaciones o el cálculo de valores propios de una matriz simétrica.
- **Rutinas Computacionales**: Resuelven diferentes tareas computacionales concretas, como, por ejemplo, una factorización LU. Cada rutina *driver* estará formada por una secuencia de llamadas a rutinas computacionales.
- **Rutinas Auxiliares**:
    - Rutinas que realizan subtareas dentro de rutinas por bloques. Principalmente son implementaciones de las versiones sin bloques de estas mismas rutinas.
    - Rutinas que realizan computaciones de bajo nivel requeridas frecuentemente como, por ejemplo, escalar una matriz. De cara a una mejor organización de las diferentes librerías, algunas de ellas podrían ser incluidas en futuras versiones de BLAS.
    - Unas pocas extensiones de BLAS tales como operaciones matriz-vector con matrices complejas. Al igual que las anteriores, también podrían incorporarse a futuras versiones de BLAS.

Por otro lado, a partir de las librerías básicas de comunicaciones (PVM, MPI, ...) se creó la librería BLACS [DW95][wBLACS]. BLACS es un paquete preparado para usarse en programas de álgebra lineal paralelos implementados según el paradigma de paso de mensajes. Este paquete soporta, de manera eficiente, comunicaciones punto a punto y colectivas, entre procesadores configurados siguiendo una topología lógica de malla bidimensional.

Usando BLAS y BLACS como librerías básicas se diseñó PBLAS [CDW92][wPBLAS]. PBLAS está formado por la extensión de un subconjunto de rutinas de BLAS para computadores de memoria distribuida, operando con matrices distribuidas de acuerdo al esquema cíclico por bloques. Siguiendo este esquema, bloques de datos consecutivos son distribuidos cíclicamente entre los procesadores (Figura 3.9).

A partir de los módulos BLAS, PBLAS y BLACS se construyó ScaLAPACK [BCC+97][wScaLA]. PBLAS se usa en ScaLAPACK al estilo que se utiliza BLAS en LAPACK, de manera que la mayoría de las llamadas a rutinas de nivel 2 y 3 de BLAS que aparecen en rutinas de LAPACK pueden ser reemplazadas en ScaLAPACK por llamadas a las rutinas correspondientes de PBLAS. En la Figura 3.10 se puede ver la jerarquía de librerías sobre la que se construye ScaLAPACK. Las rutinas principales de ScaLAPACK normalmente sólo llaman a rutinas de PBLAS, pero las rutinas auxiliares pueden necesitar llamar a BLAS directamente o a LAPACK para algunos cálculos locales, así como a BLACS para alguna comunicación entre procesos.

Además de estas librerías descritas, habría que destacar algunas otras que también podrían formar parte de esta jerarquía, como ATLAS [wATL] o PLAPACK [vdG97][wPLAP]. ATLAS, como ya se comentó en el capítulo anterior, es una librería equivalente a BLAS que además cuenta con capacidad de adaptarse automáticamente a las condiciones de la plataforma donde se instala. Por su parte, PLAPACK es una librería de similar funcionalidad que ScaLAPACK, pero desarrollada con un nivel más alto de abstracción, de cara a que se puedan implementar algoritmos más sofisticados. Por otro lado, como describiremos en el capítulo 6, esta jerarquía puede extenderse a lo ancho en los diferentes niveles, así como con nuevas librerías en niveles superiores formadas por rutinas de alto nivel que utilizan el software descrito en este apartado.

| | | | | | | | | | **D** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***p,q*** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| | **0** | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 |
| | **1** | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 |
| | **2** | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 |
| | **3** | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 |
| | **4** | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 |
| ***B*** | **5** | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 |
| | **6** | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 |
| | **7** | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 |
| | **8** | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 |
| | **9** | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 | 0,0 | 0,1 | 0,2 | 0,3 |
| | **10** | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 | 1,0 | 1,1 | 1,2 | 1,3 |
| | **11** | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 | 2,0 | 2,1 | 2,2 | 2,3 |

(a) Asignación de los índice globales de bloques (*B*,*D*), a los procesos (*p*,*q*)

| | | | | | | | | | ***q*** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***B,D*** | **0** | | | | **1** | | | | **2** | | | | **3** | | | |
| | **0** | 0,0 | 0,4 | 0,8 | 0,12 | 0,1 | 0,5 | 0,9 | 0,13 | 0,2 | 0,6 | 0,10 | 0,14 | 0,3 | 0,7 | 0,11 | 0,15 |
| | | 3,0 | 3,4 | 3,8 | 3,12 | 3,1 | 3,5 | 3,9 | 3,13 | 3,2 | 3,6 | 3,10 | 3,14 | 3,3 | 3,7 | 3,11 | 3,15 |
| | | 6,0 | 6,4 | 6,8 | 6,12 | 6,1 | 6,5 | 6,9 | 6,13 | 6,2 | 6,6 | 6,10 | 6,14 | 6,3 | 6,7 | 6,11 | 6,15 |
| | | 9,0 | 9,4 | 9,8 | 9,12 | 9,1 | 9,5 | 9,9 | 9,13 | 9,2 | 9,6 | 9,10 | 9,14 | 9,3 | 9,7 | 9,11 | 9,15 |
| ***p*** | **1** | 1,0 | 1,4 | 1,8 | 1,12 | 1,1 | 1,5 | 1,9 | 1,13 | 1,2 | 1,6 | 1,10 | 1,14 | 1,3 | 1,7 | 1,11 | 1,15 |
| | | 4,0 | 4,4 | 4,8 | 4,12 | 4,1 | 4,5 | 4,9 | 4,13 | 4,2 | 4,6 | 4,10 | 4,14 | 4,3 | 4,7 | 4,11 | 4,15 |
| | | 7,0 | 7,4 | 7,8 | 7,12 | 7,1 | 7,5 | 7,9 | 7,13 | 7,2 | 7,6 | 7,10 | 7,14 | 7,3 | 7,7 | 7,11 | 7,15 |
| | | 10,0 | 10,4 | 10,8 | 10,12 | 10,1 | 10,5 | 10,9 | 10,13 | 10,2 | 10,6 | 10,10 | 10,14 | 10,3 | 10,7 | 10,11 | 10,15 |
| | **2** | 2,0 | 2,4 | 2,8 | 2,12 | 2,1 | 2,5 | 2,9 | 2,13 | 2,2 | 2,6 | 2,10 | 2,14 | 2,3 | 2,7 | 2,11 | 2,15 |
| | | 5,0 | 5,4 | 5,8 | 5,12 | 5,1 | 5,5 | 5,9 | 5,13 | 5,2 | 5,6 | 5,10 | 5,14 | 5,3 | 5,7 | 5,11 | 5,15 |
| | | 8,0 | 8,4 | 8,8 | 8,12 | 8,1 | 8,5 | 8,9 | 8,13 | 8,2 | 8,6 | 8,10 | 8,14 | 8,3 | 8,7 | 8,11 | 8,15 |
| | | 11,0 | 11,4 | 11,8 | 11,12 | 11,1 | 11,5 | 11,9 | 11,13 | 11,2 | 11,6 | 11,10 | 11,14 | 11,3 | 11,7 | 11,11 | 11,15 |

(b) Bloques globales (*B*,*D*) en cada proceso (*p*,*q*)

Figura 3.9. Descomposición cíclica por bloques de una matriz de dimensiones 48×80, con un tamaño de bloque 4×5, sobre una malla de 3×4 procesos.

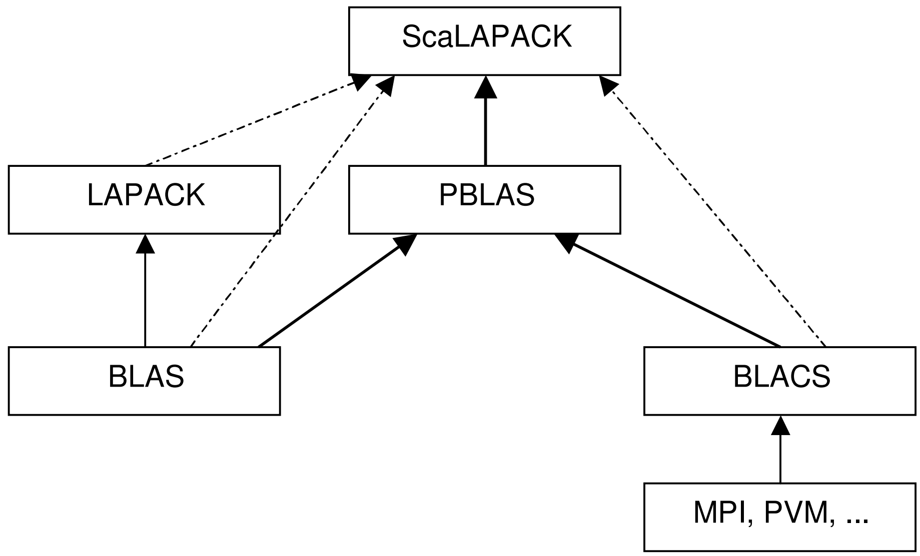


Figura 3.10. Jerarquía de librerías de álgebra lineal bajo ScaLAPACK.

### 3.4.3 Software auxiliar

De cara a desarrollar software paralelo eficiente y escalable, existe una serie de herramientas auxiliares que ayudan al programador en esta labor [wTools][MCL+01][BDT01]. Las funciones generales que realizan estas herramientas son: ayudar en el desarrollo de software, recopilar información sobre las prestaciones que se obtienen y analizar esta información. A continuación se resumirán las características más relevantes de algunas de estas herramientas.

DEEP (*Development Environment of Parallel Programs*) y DEEP/MPI [wDEEP] son herramientas de análisis de programas paralelos. Proporcionan un interfaz gráfico integrado para análisis de prestaciones de programas, tanto de paso de mensajes como de compartición de variables. Incluye un visualizador del árbol de rutinas llamadas y diversas herramientas para extraer y manipular datos de prestaciones de la ejecución de los programas como, por ejemplo, el tiempo de ejecución de cada procedimiento. Además, puede mostrar el balanceo de la carga de trabajo y de mensajes enviados de los distintos nodos de la plataforma.

MPE (*Multi-Processing Environment*) [wMPE] es una librería distribuida junto a la implementación MPICH, pero que también se puede usar con otras implementaciones de MPI. MPE incluye herramientas para depurar, realizar trazas de seguimientos (con la utilidad *logging*) y mostrar análisis gráficos de estas trazas (con la utilidad *jumpshot*).

Pablo [wPablo] es un entorno de análisis de prestaciones diseñado para capturar, analizar y presentar información sobre las prestaciones de rutinas construidas con MPI. Además, con su utilidad *Autopilot* se puede monitorizar en tiempo real las prestaciones de aplicaciones paralelas y distribuidas. Por otra parte, cuenta con *SvPablo* como un interfaz gráfico que introduce código para instrumentalizar los eventos que decida el usuario.

Paradyn [wPard] es una herramienta para medir las prestaciones de programas paralelos y distribuidos. Paradyn inserta dinámicamente instrumentación en rutinas que se estén ejecutando y muestra en tiempo real las prestaciones que se van obteniendo.

TAU (*Tuning and Analysis Utilities*) [wTAU] es una herramienta para realizar seguimiento de las prestaciones de rutinas paralelas de paso de mensajes con MPI, o de memoria compartida con OpenMP o PThreads. Los datos que se obtienen pueden ser visualizados con su propia utilidad gráfica, *Racy*, o bien usando la herramienta de visualización de VAMPIR.

PGPROF [wPGPR], que es parte del entorno de desarrollo de software PGI (*Portland Group Inc.*), es una herramienta de análisis y visualización de prestaciones de programas de memoria compartida que utilicen *threads*. Estos programas paralelos pueden haber sido generados mediante autoparalelización o mediante OpenMP.

Dimemas [wDIME] es una herramienta de predicción de prestaciones en programas de paso de mensajes. Permite al usuario ir desarrollando y optimizando sus programas usando un único procesador, mientras proporciona predicciones, mediante simulación, de sus prestaciones en la plataforma paralela de destino. Soporta MPI, PVM y PARMACS. Los ficheros de trazas que generan pueden ser visualizados y analizados con PARAVER y VAMPIR.

VAMPIR [wVAMP] es una herramienta de análisis de prestaciones para programas paralelos que usan MPI. Tiene su propio interfaz gráfico para mostrar al usuario los datos obtenidos.

PARAVER [wPARA] cuenta con una herramienta de visualización y con un conjunto de mecanismos para el análisis de aplicaciones de paso de mensajes y OpenMP, para contadores de prestaciones hardware y para predicciones realizadas con el simulador DIMEMAS.

Por otro lado, existe una serie de herramientas que pueden ser utilizadas para conocer el estado de las plataformas de ejecución, tanto de sus condiciones estáticas (capacidad de cómputo de los distintos procesadores, capacidad de comunicación entre los diferentes nodos, capacidad de memoria, etc), como de sus condiciones dinámicas (carga de trabajo que soportan las distintos procesadores y la plataforma en su conjunto). Estas herramientas son especialmente útiles para desarrollar sistemas software de ajuste automático de las prestaciones de rutinas en base a las condiciones de la plataforma de ejecución, como el que planteamos en este trabajo. A continuación se resumen las características de algunas de estas herramientas:

PAPI (*Performance API*) [wPAPI] es un interfaz que, independientemente de la plataforma en que se encuentre, ofrece información local sobre las prestaciones del procesador, memoria, paginamiento, etc. PAPI puede usarse, a su vez, por varias las herramientas (DEEP, Pablo, ...) para conocer esta información, que PAPI obtiene accediendo a los contadores hardware que tenga disponibles en el sistema.

NWS (*Network Weather Service*) [wNWS][WSH99] es una herramienta que puede funcionar sobre un sistema distribuido monitorizándolo periódicamente y haciendo predicciones de sus prestaciones, tanto de los recursos computacionales como de comunicaciones. Esta herramienta está formada por cuatro componentes principales:

- Un conjunto distribuido de procesos sensores encargados de recoger las diferentes prestaciones del sistema:
    - Disponibilidad de los diferentes procesadores, tanto para los procesos ya funcionando como para un proceso nuevo.

  - Disponibilidad de la red, midiendo la latencia, el ancho de banda y el tiempo de conexión entre cada par de nodos.
  - Memoria libre que existe en los diferentes nodos.
  - Espacio libre de los diferentes discos del sistema.
- Un servidor de nombres que liga nombres de procesadores y de datos con información de contacto a bajo nivel, por ejemplo, el número puerto TCP/IP.
- Un conjunto de procesos encargados de guardar la información recogida por los sensores.
- Un proceso predictor que, a petición de alguna aplicación cliente que requiera sus servicios, realiza estimaciones estadísticas, en base a los datos recogidos por los sensores, sobre las prestaciones que van a ofrecer distintas partes del sistema.

REMOS (*REsource MOnitoring System*) [wREMO][DGK+01] es una herramienta que pueden utilizar las aplicaciones distribuidas para obtener información sobre la situación de la red donde se están ejecutando. Esta herramienta está formada por:

- Colectores: procesos encargados de recoger información sobre la situación de la red o estructura de redes del sistema. Los colectores envían esta información a los modeladores.
- Modeladores: Recogen la información de los colectores y la procesan adecuadamente para proveer un interfaz a las aplicaciones clientes, proporcionándoles la información concreta que necesitan.
- Predictores: Con la información histórica recogida por los colectores, realizan predicciones sobre la situación de la red en un futuro próximo.

De todas estas herramientas descritas, durante la realización de este trabajo se han utilizado MPE y PAPI durante el desarrollo de algunas rutinas y NWS para la monitorización del estado del sistema.

## 3.5 Resumen

En este capítulo se ha mostrado una visión general del marco de trabajo en el que se han llevado a cabo las diferentes labores de nuestra propuesta, esto es, software de computación numérica en plataformas paralelas.

Para enmarcar el entorno hardware de este trabajo, se ha empezado dando una visión general de las diferentes plataformas paralelas existentes en la actualidad. Se incluyen entre estas plataformas multiprocesadores de memoria compartida, multiprocesadores de memoria distribuida, conjuntos de máquinas conectadas por una red local formando un *cluster,* así como otras combinaciones heterogéneas de máquinas y redes formando plataformas distribuidas. Todas estas plataformas tienen, obviamente, características físicas bien diferentes. Sin embargo, cualquiera de ellas puede ser vista como una plataforma paralela genérica, donde se cumple que:

- La plataforma está formada por un conjunto de procesadores que pueden funcionar de manera independiente.
- Cada procesador puede acceder localmente a una porción de la memoria total de la plataforma.
- El conjunto de procesadores de la plataforma puede adquirir diferentes topologías lógicas (malla, anillo, hipercubo, ...).
- La plataforma se puede programar, obteniéndose un alto grado de eficiencia, siguiendo el paradigma de paso de mensajes.

A continuación se han descrito detalladamente las características concretas de las plataformas paralelas que se han utilizado en la parte experimental de este trabajo: un multicomputador de memoria distribuida, un multicomputador de memoria compartida físicamente distribuida, un *cluster* de PCs y un *cluster* de estaciones de trabajo.

En cuanto al entorno software, en primer lugar se han resumido las características básicas de los estándares de compartición de datos (OpenMP) y de comunicaciones (PVM y MPI) más extendidos en la actualidad. A continuación, se ha dado una visión del conjunto jerárquico de librerías más usadas en la computación de álgebra lineal. Esta jerarquía está formada, en su base, por BLAS como librería básica de cálculo en plataformas secuenciales, así como la librería LAPACK, desarrollada a partir de BLAS. Por otro lado tenemos BLACS como librería de comunicaciones, formada a partir de primitivas de paso de mensajes (PVM, MPI, ...). Con la ayuda de BLACS se construye el equivalente de BLAS para plataformas paralelas, PBLAS. Por último, a partir de las anteriores, se desarrolla ScaLAPACK, como el equivalente de LAPACK para plataformas paralelas de memoria distribuida. Finalmente se han descrito una serie de herramientas software auxiliares para el desarrollo y análisis de programas (DEEP, Pablo, Paradyn, ...), así como para testear el estado en que se encuentra la plataforma hardware (NWS, PAPI, REMOS).

# Capítulo 4. Propuesta de Modelado de la Computación Numérica en Sistemas Paralelos

## 4.1 Introducción

En este capítulo se describirá nuestra propuesta de un modelo analítico para rutinas de álgebra lineal, donde quede reflejado su tiempo de ejecución sobre una plataforma paralela genérica. Este modelo constituye el punto de partida sobre el que se construye la arquitectura software de ajuste automático de rutinas de álgebra lineal que se planteará a lo largo de los siguientes capítulos.

En primer lugar, se repasarán las principales aproximaciones que se han realizado en los últimos años sobre modelado de computación paralela. Los diferentes modelos planteados se han utilizado históricamente con tres objetivos principales: realizar estudios meramente teóricos, realizar comparaciones entre diferentes plataformas y realizar estimaciones del tiempo de ejecución de rutinas sobre una plataforma dada. Tras esta revisión, y en base a estos antecedentes históricos, se establecerá como punto de partida de nuestra aproximación los requisitos y necesidades que debe cubrir un modelo. En la sección cuarta de este capítulo, se describirá nuestra propuesta de modelo para un sistema paralelo de computación numérica. Este sistema será la unión de una plataforma paralela genérica junto al software de comunicaciones y al software de cálculo aritmético que tenga instalados. A continuación, tomando como arquitectura destino el sistema paralelo de computación numérica modelado anteriormente, se propondrá un modelo analítico para el tiempo de ejecución de rutinas de álgebra lineal. Se describirán las distintas rutinas que se han utilizado en los experimentos llevados a cabo en este trabajo, y para cada una de ellas se detallará el diseño de su modelo analítico siguiendo nuestra propuesta.

Para finalizar el capítulo, con el objetivo de validar nuestra propuesta, se mostrará una comparación entre los tiempos teóricos que predice el modelo analítico de cada rutina estudiada frente al tiempo experimental obtenido en diversas plataformas hardware.

## 4.2 Antecedentes

Desde finales de los años 80 se han realizado un gran número de estudios que plantean modelos de arquitecturas paralelas desde multitud de enfoques, obteniéndose modelos con diferentes grados de abstracción [Cam97][JW98][Gon03]. Algunos de estos modelos han permanecido únicamente en el terreno teórico, mientras que otros han sido utilizados para intentar predecir el comportamiento real de la ejecución de programas, realizándose estudios comparativos de los tiempos de ejecución teóricos predichos por estos modelos con los tiempos de ejecución medidos al ejecutar los programas.

En esta sección veremos algunos de estos planteamientos, con sus características más relevantes y destacando cuáles han sido sus aportaciones al estudio del modelado de plataformas y rutinas paralelas. En primer lugar se verán algunos planteamientos generales propuestos en los últimos años. A continuación, motivados por la importancia del coste de las comunicaciones en el tiempo total de ejecución del software paralelo, se verá un conjunto de propuestas que han centrado su estudio en el modelado de dicho coste.

Como veremos en las siguientes secciones, este conjunto planteamientos históricos nos han servido de punto de partida, extrayendo diversas ideas que hemos considerado interesantes incorporar a nuestra aproximación.

### 4.2.1 Modelos generales

#### El modelo PRAM

El modelo PRAM, introducido en [FW78], supuso un punto de partida en el modelado de plataformas paralelas y un referente para los planteamientos surgidos en los siguientes años. Se basa en una visión con un alto grado de abstracción de las arquitecturas paralelas, lo que lo convirtió desde su creación en un interesante modelo para realizar estudios teóricos.

Se considera una plataforma de ejecución multiprocesador con una memoria compartida que tiene un tiempo de acceso uniforme por parte de todos los procesadores. Además de esta memoria compartida, cada procesador cuenta con un conjunto de registros para almacenamientos locales. Por último, se considera un unidad de control centralizada que, de manera síncrona, va marcando la ejecución de cada instrucción en todos los procesadores. Para este modelo, cada una de estas instrucciones tiene un coste de ejecución fijo de una unidad de tiempo, independientemente de que se trate de una instrucción de cálculo o de un acceso a la memoria compartida (única forma de comunicación que aparece en esta propuesta). Basándose en cómo se gestionan los accesos simultáneos de más de un procesador a las mismas posiciones de memoria, el modelo PRAM cuenta con cuatro subclases [KGG+94]:

- Lectura exclusiva, escritura exclusiva (EREW PRAM). En esta subclase los accesos a cualquier posición de memoria son exclusivos, no permitiéndose lecturas o escrituras concurrentes.
- Lectura concurrente, escritura exclusiva (CREW PRAM). En esta subclase se permiten múltiples accesos simultáneos de lecturas a una misma posición de memoria.
- Lectura exclusiva, escritura concurrente (ERCW PRAM). En esta subclase se permiten múltiples accesos simultáneos de escrituras a una misma posición de memoria.
- Lectura concurrente, escritura concurrente (CRCW PRAM). En esta subclase se permiten múltiples accesos simultáneos tanto de lecturas como de escrituras a una misma posición de memoria.

Para las escrituras concurrentes a una misma posición es necesario algún protocolo de arbitraje: que todas las escrituras escriban el mismo valor, que se deje escribir arbitrariamente a cualquier procesador, o bien que exista algún sistema de prioridades entre los procesadores.

En el modelo PRAM no queda reflejado que, normalmente, en una plataforma paralela cualquier tipo de comunicación o acceso a memoria compartida es mucho más lento que los cómputos locales. Con el objetivo de tener una visión más realista de las plataformas paralelas han ido surgiendo numerosas extensiones del modelo PRAM. Algunas de estas extensiones son descritas a continuación.

El modelo APRAM [CZ89] es una variante asíncrona del PRAM. En este esquema el tiempo de computación es medido en *rounds*, donde un *round* es el tiempo requerido por cada procesador para ejecutar al menos una instrucción. Por tanto, el tiempo de ejecución se mide de acuerdo al reloj más lento.

El modelo BPRAM [ACS90] es una variante del PRAM donde las comunicaciones con la memoria son en bloques de datos en lugar de escrituras/lecturas de datos individuales. Esta nueva característica permite modelar la localidad espacial y temporal de los datos, pues cuando un procesador trae un bloque desde la memoria global, éste se guarda en su memoria local. El tiempo de traer un bloque de $b$ datos es modelado como ($l$+$b$), siendo $l$ la latencia con la memoria global ($l$ será, normalmente, función del número de procesadores). El tiempo de acceso a cualquier dato de la memoria local es sólo de una unidad de tiempo.

En la variante LPRAM [ACS90] cada procesador cuenta con una memoria local que puede usar de igual manera que usa la memoria global. En este modelo hay dos latencias diferentes: una será el tiempo de acceder a la memoria local (se considera de una unidad de tiempo) y la otra será el tiempo de acceder a la memoria global. Por tanto, el modelo de tiempo incluye dos tipos de instrucciones con diferentes costes: instrucciones de comunicación (acceso a memoria global) consideradas de un tiempo fijo e instrucciones de cálculo (el procesador opera con datos de la memoria local) consideradas de un tiempo unidad.

La variante WPRAM [NDD93] considera una memoria compartida de dos tipos: una memoria compartida global, con un tiempo de acceso uniforme para todos los nodos, y una memoria compartida local donde los datos compartidos se encuentran en un nodo concreto. El tiempo de acceso a los datos compartidos que están distribuidos físicamente será no uniforme.

La variante YPRAM [dlTK91] plantea un modelo descomponible recursivamente en 2 grupos de máquinas, siendo cada grupo equivalente a una submáquina. Dentro de cada submáquina los procesos tienen periodos de computación y periodos de accesos a la memoria perteneciente a dicha submáquina. Esta descomposición permite el modelado de una jerarquía de memoria multinivel, donde cada nivel corresponde a una submáquina diferente en la jerarquía.

La variante HPRAM [HR92] consiste en una jerarquía dinámicamente configurable de PRAMs síncronas que operan asíncronamente entre ellas. Al igual que en la variante YPRAM, se modela una memoria multinivel. Además tiene en cuenta los diferentes costes de latencia y de sincronización entre las diferentes sub-PRAMs.

## El modelo BSP

El modelo BSP [Val90] plantea un modelo de computador paralelo llamado BSPC que consiste en un número fijo de $p$ procesadores con memoria local, una red de interconexión con un ancho de banda limitado y un sistema de barreras (*barriers)* de sincronización de coste fijo. En el BSPC, la ejecución de un programa se realizará en una serie de etapas (*supersteps*). La computación en cada *superstep* es local, de manera que los procesadores trabajan, de manera independiente, con sus datos locales en su memoria. Las comunicaciones en cada *superstep* consisten en que cada procesador envía como máximo $h$ mensajes de pequeño tamaño a $h$ procesadores destino , donde estarán disponibles en el siguiente *superstep*.

Los parámetros para definir el coste en este modelo son:

- $p$: Número de procesadores.
- $g$: El coste por cada comunicación.
- $L$: El mínimo tiempo entre sincronizaciones, es decir, la mínima longitud de un *superstep* impuesta por el hardware.

Tomando $m$ como el tiempo dedicado a computación existen dos versiones de este modelo:

- Versión 1: En cada *superstep* hay dos fases bien diferenciadas, una primera de computación y una segunda donde se envían los mensajes. El coste total de un *superstep* será: $T = \max(m+gh, L)$.
- Versión 2: Los mensajes se envían en cualquier momento del *superstep*, solapándose computación y comunicación. El coste total de un *superstep* será: $T = \max(m, gh, L)$.

Este modelo permite, dada una plataforma, determinar entre varios algoritmos cuál es el más apropiado, teniendo un alto grado de acierto. Si embargo, la precisión para calcular el tiempo real de ejecución no es muy alta debido principalmente a dos razones surgidas de su estricta estructura de sincronización. La primera es que no tiene en cuenta los retrasos que se pueden producir debido a saturaciones en la red de interconexión cuando muchos procesadores están enviando mensajes a la vez. La segunda es el alto coste de la sincronización mediante una barrera global al final de cada *superstep*, además de su poca escalabilidad. A continuación se describen algunas variantes de este modelo.

El modelo D-BSP [dlTK96] es una variante del BSP, en el que la plataforma modelada es un conjunto de procesadores particionado en $2^i$ *clusters*, con $i>0$, donde cada *cluster* se comporta como un BSPC independiente. Posteriormente apareció el D-BSP* [DMP02] donde a cada procesador se le introdujeron dos niveles de memoria jerárquica: RAM y el disco de entrada/salida.

El modelo E-BSP [JW96] extiende el modelo básico BSP de cara a tener en cuenta los patrones de comunicación desbalanceados, es decir, aquellos en los que los procesadores envían y reciben diferentes cantidades de datos.

El modelo OBSP [GLP+99] relaja el estricto proceso de sincronización del BSP original, permitiendo que en un momento dado diferentes procesadores puedan encontrarse en distintos *supersteps*, pero manteniendo las características del BSP de que el número total de *supersteps* llevados a cabo por todos los procesadores es el mismo y que las comunicaciones entre procesadores se hacen efectivas cuando el receptor alcanza el final del *superstep i* en el que el remitente se encontraba cuando lanzó el mensaje.

El modelo CCM [RSL+99] plantea una nueva clase de *superstep*, el *division superstep*, que divide los procesadores en grupos, los datos son distribuidos entre estos grupos, se realizan tareas de cómputo en cada grupo y finalmente se redistribuyen los resultados. Además el número de posibles patrones de comunicación es limitado a un conjunto reducido donde se incluyen todos los esquemas típicos de comunicaciones colectivas. Esto último tiene como objetivo obtener previsiones del tiempo de ejecución que tengan una mayor precisión.

### El modelo LogP

El modelo LogP [CKP+93] surgió como un planteamiento para superar las limitaciones de los modelos PRAM y BSP, pretendiendo ser un modelo más realista del comportamiento de plataformas paralelas. El modelo de computador que plantea es, de manera similar al del BSP, un conjunto de procesadores con memoria local unidos a través de una red caracterizada por unos pocos parámetros. Dos conceptos introducidos en este modelo que lo diferencian del BSP son:

- No existe ningún mecanismo de sincronización, excepto el que conlleva implícitamente cada instrucción de paso de mensajes incluida en los programas.
- La limitación del tráfico en la red no viene marcada por un número máximo de mensajes a enviar (parámetro $h$ del BSP), sino que el límite lo marca la velocidad de los puertos de la red de interconexión, modelada por unos pocos parámetros.

El modelo de coste incluye los siguientes parámetros:

- $L$ (*latency*): cota superior del tiempo de comunicación de un mensaje pequeño (normalmente se considera un mensaje pequeño el formado por una única palabra).
- $o$ (*overhead*): tiempo durante el que un procesador que inicia un envío o una recepción de un mensaje se encuentra ocupado en ese proceso, no pudiendo realizar ninguna otra tarea.
- $g$ (*gap*): intervalo de tiempo entre la recepción/transmisión de dos mensajes consecutivos en un procesador.
- $P$: número de módulos (procesadores, memoria).

La capacidad de la red de interconexión está limitada por estos parámetros, de manera que a lo sumo $\lceil L/g \rceil$ mensajes pueden encontrarse en tránsito desde un procesador a cualquier otro en un momento dado. El coste de comunicar un mensaje corto entre dos procesadores será igual a:

$$T_{COM} = 2o + L \tag{4.1}$$

y el coste de comunicar $n$ mensajes cortos de manera consecutiva entre dos procesadores será igual a:

$$T_{COM} = 2o + L + (n-1)\max(o, g) \tag{4.2}$$

A continuación se mostrarán algunas extensiones del modelo LogP que han ido surgiendo con el objetivo de mejorar el modelado de interfaces de comunicación actuales más complejos, como pueden ser PVM o MPI.

El modelo LogGP [AIS+95] tiene en cuenta en su estudio los mensajes de gran longitud de manera diferenciada. Cuenta con un nuevo parámetro *G* (*gap* entre bytes consecutivos de un mensaje largo), el cual captura el ancho de banda obtenido para estos mensajes de mayor longitud. De esta manera el coste de comunicar un mensaje formado por *k* bytes entre dos procesadores será:

$$T_{COM}(k) = 2o + L + kG \qquad 4.3$$

El modelo P-LogP [KBG00] es una extensión jerárquica del LogP que trata mensajes de longitud variable y comunicaciones colectivas. Mantiene los mismos parámetros que LogP, pero los valores de *g* y *o* son considerados como funciones del tamaño de los mensajes que se envíen.

El modelo LogGPS [IFH01] maneja un *overhead* variable para simular la sincronización implícita entre procesadores antes de la transmisión de mensajes largos en interfaces de paso de mensaje como MPI.

El modelo LogGPC [MF01] tiene en cuenta la contención en el tráfico de la red de interconexión añadiendo este coste al modelo de coste total.

El modelo P-3PC [SK02] es una variante del LogGP de similar complejidad pero especializado en las comunicaciones punto a punto estándares de MPI, considerando valores diferentes para el tiempo de emisión y el tiempo de recepción de un mensaje.

### Otros Modelos Generales

En este apartado se enumerarán algunas otras aproximaciones que también nos han aportado algunas ideas interesantes a la hora de plantear nuestro modelo de computación paralela.

Norton y Pfister [NP85] plantean submodelos de colas probabilísticos para estudiar por separado cada componente del sistema (procesador/caché, red de interconexión y memoria). Finalmente se plantea un modelo global que agrupa a todos estos submodelos.

Flatt y Kennedy [FK89] describen un modelo analítico del tiempo de ejecución de una rutina sobre una plataforma paralela donde se incluye la sobrecarga (*overhead*) de tiempo debida a las comunicaciones y sincronizaciones entre los diferentes procesos. Esta sobrecarga es considerada como una función del número de procesadores. Teniendo todo esto en cuenta, se puede extraer de manera analítica el número óptimo de procesadores a utilizar.

Ammar *et al.* [AIA+90] desarrollan dos modelos para el estudio de las prestaciones de programas sobre multiprocesadores de memoria compartida. En primer lugar, la estructura del algoritmo es modelada mediante Redes de Petri Estocásticas Generalizadas (*Generalized Stochastic Petri Nets*, *GSPNs*), donde se definen las diferentes tareas junto con sus restricciones de dependencia y sincronización. En segundo lugar, mediante Redes de Colas (*Queuing Networks*, *QNs*) se modelan los accesos a los diferentes recursos de la plataforma.

Willebeek-LeMair *et al.* [WRN90] plantean un modelo para caracterizar las prestaciones de una plataforma multiprocesador mediante una medida, el *transfer ratio*, en la que quedan reflejadas las capacidades de la plataforma. El *transfer ratio* normaliza el coste de operaciones complejas (comunicaciones básicamente) que es calculado como el cociente entre el coste medio de estas operaciones complejas y el coste medio de operaciones aritméticas simples (sumas y multiplicaciones escalares).

Liu y Peir [LP90] describen un modelo analítico detallado para multiprocesadores de memoria compartida. Este modelo supone que los programas están formados por regiones de computación y regiones de comunicación separadas, por lo que está formado por dos submodelos, uno encargado de caracterizar la computación y otro para modelar los accesos a la memoria compartida. El submodelo de cómputo es totalmente experimental. Para obtenerlo, en primer lugar se realizan experimentos secuenciales de las diferentes partes computacionales de la rutina para algunos tamaños de problema, tras ello, se extrapolan los resultados para el resto de tamaños. Para el submodelo de accesos a memoria se hace un estudio teórico de cada rutina, obteniendo un modelo analítico detallado del esquema de acceso a datos que ésta tiene. En este submodelo aparecen parámetros del sistema como la latencia y el ancho de banda cuyos valores se toman de las especificaciones de la plataforma.

Gallivan *et al.* plantean el modelo *load/store* [GJM+91] para programas escritos en Fortran destinados a multiprocesadores de memoria compartida. Dado un programa, de su código original se extrae el esquema de accesos a memoria (*loads* y *stores*) que realiza. A continuación, gracias a una base de datos de estos esquemas con sus tiempos de ejecución en una plataforma, son capaces de predecir el tiempo de ejecución de este programa en esa plataforma.

Crovella *et al.* [CBL+92] proponen un modelo simple basado en el ratio comunicación-computación (*C/C ratio*) que se puede usar para predecir las prestaciones de un programa en una plataforma paralela incluso cuando el programa aún no está acabado. Esto lo convierte en una útil herramienta para la programación, ayudando a tomar decisiones que mejoren las prestaciones de los programas cuando aún están en desarrollo.

Clemment y Quinn [CQ93] plantean un modelo analítico en el que se tienen en cuenta el coste de la computación aritmética, el coste de las comunicaciones, el coste de los accesos a memoria y los efectos del compilador usado (número de instrucciones máquina generadas por cada operación lógica del programa). Los parámetros que caracterizan estos costes son obtenidos de manera directa, sin apenas experimentación, a partir del código original del programa, de las especificaciones estándares de la plataforma y de las características del compilador.

Crovella y LeBlanc [CL94][Cro94] realizan un amplio estudio de cómo medir y modelar todas las fuentes de sobrecarga (*overhead*) que aparecen en un programa al ejecutarse sobre una plataforma paralela usando el método *Lost Cycles Analysis*. Usando este método, las fuentes de *overhead* se dividen en varias categorías: desbalanceo de la carga de trabajo, paralelismo insuficiente, coste de sincronización, coste de comunicaciones y tiempo de espera para obtener recursos hardware. Para cada una de estas medidas se crea un modelo analítico a partir de experimentación. El modelo final del tiempo de ejecución de un programa estará formado por la unión de todos estos submodelos y el tiempo de computación aritmética.

Zaki *et al.* [ZLP95] desarrollan un modelo válido para la toma de decisiones de balanceo de la carga de trabajo de programas entre los diferentes procesadores de una plataforma paralela. En este modelo se manejan dos tipos de parámetros, los estáticos y los dinámicos. Los parámetros estáticos, cuyos valores se obtienen en tiempo de instalación, recogen las características del programa, del procesador y de la red de interconexión. Por otro lado, los parámetros dinámicos, cuyos valores se obtienen en la ejecución de cada rutina, caracterizan la carga de trabajo que tienen los procesadores en un momento dado.

Xu *et al.* [XZS96] plantean un modelo jerárquico de dos niveles. Un modelo en forma de grafo de tareas se usa para describir el comportamiento de los programas, teniendo en cuenta la sobrecarga introducida por las comunicaciones y las sincronizaciones. Un modelo analítico se usa para caracterizar la plataforma paralela, estimándose los parámetros implícitos de la arquitectura mediante experimentación.

Zhang [Zha99] propone un modelo donde se incorpora el acceso a los diferentes niveles de la jerarquía de memoria al coste computacional de una rutina. Además considera, de cara a su propuesta de modelo, que el acceso por parte de un procesador a datos que se encuentren en la memoria local de otro nodo es equivalente a acceder a un nuevo nivel virtual en la jerarquía de memoria de ese procesador.

Rauber y Rünger [RR01] describen un modelo unificado del tiempo de acceso a los diferentes niveles jerárquicos de memoria en multiprocesadores de memoria compartida distribuida, pero igualmente válido para plataformas de memoria distribuida y para plataformas de memoria compartida físicamente.

### 4.2.2 Modelado de las comunicaciones

Desde que se iniciaron los estudios en el modelado de plataformas paralelas y del coste de ejecución de programas sobre esas plataformas se ha destacado la importancia que supone analizar correctamente el coste de las comunicaciones [FK89][CL94], principalmente cuando se realiza un programa siguiendo el paradigma de paso de mensajes. Por esta razón, se han realizado multitud de estudios centrados en modelar el coste de las comunicaciones de un programa al ejecutarse en una plataforma paralela.

El modelo de Hockney [Hoc82][HJ88] puede considerarse como el principal punto de referencia en el modelado del coste de comunicaciones de programas paralelos. Según este modelo el tiempo para comunicar un mensaje de *n* bytes entre dos procesadores es:

$$T_{COM}(n) = \frac{n + n_{1/2}}{r_\infty} \tag{4.4}$$

siendo $r_\infty$ el ancho de banda asintótico de una comunicación, que se alcanza cuando el tamaño del mensaje tiende a infinito, y $n_{1/2}$ el tamaño de mensaje necesario para alcanzar la mitad de ese valor asintótico. La ecuación (4.4) puede verse de una manera simplificada [RG87] como:

$$T_{COM}(n) = t_0 + nt_{trans} \tag{4.5}$$

siendo $t_0$ el tiempo de inicio de una comunicación (*startup time*) y $t_{trans}$ el tiempo de transmisión por byte (*byte sending time*), considerando que:

$$t_0 = \frac{n_{1/2}}{r_\infty}, \qquad t_{trans} = \frac{1}{r_\infty} \tag{4.6}$$

Posteriormente, se han llevado a cabo numerosos trabajos para estudiar la aplicabilidad del modelo clásico de Hockney a los sistemas de comunicaciones actuales (nuevas redes de interconexión y nuevos interfaces de comunicación), a las comunicaciones colectivas, a redes de interconexión heterogéneas, a sistemas distribuidos, etc. A continuación se enumeran algunas de estas extensiones o modificaciones.

En el *modelo postal* [BK92] se considera que si un procesador inicia la comunicación de un mensaje atómico en el instante *t,* estará ocupado hasta el instante ($t$+1), mientras que el receptor está ocupado recibiendo este mensaje desde el instante ($t$+λ-1) al instante ($t$+λ), siendo λ la latencia del sistema. Este modelo puede considerarse equivalente al de Hockney, normalizando $t_{trans}$= 1 y tomando $\lambda = t_0$.

En el modelo LogP, visto anteriormente, se caracteriza el tiempo de comunicación entre dos procesadores de manera equivalente al modelo clásico de Hockney si se toma $t_0 = (L+2o)$ y $t_{trans} = g$.

Kumar *et al.* [KGG+94] y Barnett *et al.* [BGP+94] muestran el modelo clásico extendido a todo el repertorio de comunicaciones colectivas, para distintas configuraciones lógicas (anillo, malla bidimensional e hipercubo) y diferentes esquemas de enrutamiento de la red de interconexión.

Foster [Fos95] refina el modelo clásico de cara a tener en cuenta las propiedades físicas de la red de interconexión. El principal factor introducido, el cuál tiene un importante impacto en las prestaciones de un red de comunicaciones, es la competición por el ancho de banda disponible. Para introducir este factor, se modifica la ecuación (4.5) introduciendo el parámetro $S$, que es el número de procesadores que necesitan enviar concurrentemente sobre el mismo canal de comunicaciones:

$$T_{COM}(n) = t_0 + n t_{trans} S \qquad 4.7$$

Arruabarrena *et al.* [AAB+96] aplican el modelo clásico al subsistema de comunicaciones del IBM-SP2, comprobando su validez, con algunas modificaciones. La principal de estas modificaciones es que el parámetro $t_0$ no se considera constante sino función del tamaño de mensaje enviado.

Getov *et al.* [GHH97] plantean que, para calcular el tiempo real de las comunicaciones de un programa, el modelo clásico debe ser ampliado para tener en cuenta otros factores, como el tipo de los datos transmitidos y el esquema de almacenamiento en memoria de estos datos.

Rauber y Rünger [RR97] desarrollan, a partir del modelo clásico, un modelo para cada una de las operaciones de comunicación de PVM y MPI. Estos modelos son validados mediante la comparación de los tiempos que predicen con tiempos experimentales medidos en un IBM-SP2.

Banikazemi *et al*. [BSP+99] y Bhat *et al*. [BRP99] realizan diversas modificaciones en el modelo clásico para adaptarlo al estudio de comunicaciones en redes heterogéneas en área local y en sistemas distribuidos, respectivamente.

Tessera y Dubey [TD01] adaptan el modelo de Hockney para analizar las causas de las diferencias en las prestaciones de las estrategias disponibles en MPI para llevar a cabo las comunicaciones entre procesadores (punto a punto vs. colectivas, protocolos bloqueantes vs. no bloqueantes, sobrecarga debido al uso de búferes y a la contención de la red, ...).

## 4.3 Punto de partida: Requisitos de un modelo

A partir de los diferentes estudios expuestos en la sección anterior, se pueden extraer algunas ideas o requisitos que deben tenerse en cuenta a la hora de plantear un modelo de computación paralela que sea útil para predecir el tiempo de ejecución de las rutinas:

- Los modelos con un alto grado de abstracción nos permitirán estudiar el comportamiento de los programas con mayor independencia de las plataformas de ejecución, mientras que, conforme se concretice más, el modelo resultante captará mejor las características de la máquina, pero a costa de perder portabilidad. Por tanto, es necesario llegar a un compromiso si queremos tener un modelo con suficiente abstracción como para que sea portable entre plataformas paralelas de muy diferentes características (memoria compartida, memoria distribuida, *clusters*, …), pero que a la vez tenga suficiente grado de concreción como para que las previsiones de tiempos de ejecución sean suficientemente realistas, y de esta manera el modelo pueda ser una herramienta útil para la tarea de ajustar adecuadamente los programas mejorando sus prestaciones [Wu99].

- Si se quiere obtener un modelo para predecir el tiempo de ejecución real de un programa será necesario, en primer lugar, un buen estudio teórico del programa para obtener un modelo analítico de éste. En segundo lugar, este estudio teórico deberá acompañarse de un conjunto de experimentos para cuantificar los valores de algunos parámetros en la plataforma donde se quiera predecir tiempos reales de ejecución del programa (Crovella y Leblanc [CL94], Zaki *et al.* [ZLP95], Xu *et al.* [XZS96], Arruabarrena *et al.* [AAB+96]).

- Para facilitar el trabajo de modelado, conviene dividir el proceso de modelado en dos submodelos, uno para la capacidad de cómputo y otro para las comunicaciones (Norton y Pfister [NP85], Liu y Peir [LP90], Crovella y Leblanc [CL94]).

- Otro punto importante a tener en cuenta es la estructura jerárquica de las memorias de las plataformas (LPRAM [ACS90], WPRAM [NDD93], Gallivan *et al.* [GJM+91], Zhang [Zha99], Rauber y Rünger [RR01]). En primer lugar, este factor influye en el tiempo de cálculo, pues en el tiempo de realizar una operación básica hay que incluir el coste de acceder a los operandos y de guardar el resultado. También influye en el tiempo de comunicación, donde se debe incluir el coste de acceder a los datos que se envían y el coste de guardar los que se reciben. Todos estos costes dependen del nivel de la jerarquía al que haya que acceder en cada caso y, por tanto, de la localidad espacial y temporal del esquema de acceso a los datos que se siga. Actualmente se utilizan en el software numérico diferentes técnicas para agrupar los accesos a los datos en forma de bloques (*bloking*) [GKU99] con el fin de aumentar la localidad espacial y temporal de los accesos a memoria.

- Es importante modelar correctamente el coste de las comunicaciones debido al gran peso que suelen tener en el coste total de ejecución del programa paralelo en las plataformas actuales (LogP [CKP+93], Flatt y Kennedy [FK89], Willebeek-LeMair *et al.* [WRN90], Crovella y LeBlanc [CL94][Cro94]). El modelo clásico ([Hoc82][HJ88]) parece un buen punto de partida, pero puede ser conveniente incluir en éste algunas variaciones de cara a tener en cuenta las características de los datos que se envían, principalmente su esquema de almacenamiento (Getov *et al.* [GHH97]). Además, hay que prestar especial atención a las comunicaciones colectivas, teniendo en cuenta cómo están implementadas en cada plataforma (Kumar *et al.* [KGG+94], Barnett *et al*. [BGP+94]).

# 4.4 Modelo propuesto para el sistema: DLAM+

De cara a tener un modelo analítico del tiempo de ejecución de rutinas de álgebra lineal que sea aplicable para cualquier máquina, debemos tener la referencia de un modelo para la arquitectura destino que sea válido para una amplía variedad de plataformas hardware. A la hora de diseñar este modelo de la arquitectura destino, nuestro objetivo no es modelar exactamente las características físicas de la plataforma, sino reflejar mediante un conjunto de parámetros los efectos que las características de la máquina pueden tener en el tiempo de ejecución del software. Es decir, más que un modelo de la arquitectura paralela, el objetivo será diseñar un modelo del **sistema paralelo de computación numérica**, entendiendo este sistema como la unión de la plataforma hardware junto al software básico de cálculo numérico y al software básico de comunicaciones que tenga instalados.

En [DK96] Dackland y Kågström plantearon el modelo DLAM (*Distributed Linear Algebra Machine*) para sistemas paralelos de computación numérica. Según este modelo, una plataforma paralela para ejecutar rutinas de álgebra lineal estaría formada por $P$ procesadores BLAS comunicados a través de una red BLACS en forma de malla bidimensional $r\times c$, con $r\times c\leq P$. Estos procesadores solo pueden realizar operaciones aritméticas de tipo BLAS y operaciones de comunicación de tipo BLACS [DW95]. En cada uno de los procesadores, el tiempo para realizar cualquier operación aritmética de nivel $i$ de BLAS sería $k_i$, para $i$=1,2,3; y el tiempo para transferir $n$ palabras entre dos procesadores sería:

$$T_{COM}(n) = (t_s + nt_w)f \tag{4.8}$$

es decir, el modelo clásico para las comunicaciones, ponderado por un factor $f$ cuyo valor dependerá de la plataforma y del tipo de comunicación. Por ejemplo, si la red de interconexión de la plataforma permite una configuración en hipercubo, el valor de $f$ será 1 para comunicaciones punto a punto, $\log_2(p)$ para *broadcast*, etc, ignorándose posibles colisiones en la red. Los valores de los parámetros aritméticos ($k_1$, $k_2$, $k_3$) y de comunicaciones ($t_s$, $t_w$) son considerados como valores constantes propios de cada plataforma, y sus valores se obtienen experimentalmente. En cuanto a la distribución de los datos entre los procesadores, en este modelo se considera que las matrices de tamaño $m\times n$ son distribuidas sobre la malla bidimensional $r\times c$ siguiendo una distribución cíclica por bloques al estilo de PBLAS y ScaLAPACK.

Partiendo de este modelo DLAM para un sistema software de álgebra lineal sobre plataformas paralelas, junto a algunas ideas aparecidas en los modelos de arquitecturas paralelas planteados en la sección anterior, así como otras características totalmente novedosas, hemos planteado nuestra propuesta de modelo, que hemos llamado DLAM+.

Según nuestro modelo DLAM+, un sistema paralelo de computación numérica consiste en $P$ procesadores de similares características físicas, conectados entre sí de manera homogénea, es decir, manteniéndose una latencia y un ancho de banda constante para comunicaciones entre cualquier par de procesadores. A la hora de ejecutar una rutina sobre esta plataforma, la primera labor a realizar será escoger los $p$ procesadores a utilizar, con $p\leq P$. La topología lógica de estos $p$ procesadores será, en un principio, una malla bidimensional de $r$ filas y $c$ columnas, con $r\times c=p$, aunque se podría tomar cualquier otra topología sin tener que modificar el modelo planteado.

En cada procesador, las operaciones aritméticas se podrán realizar mediante rutinas de cualquier librería de álgebra lineal que esté instalada en la plataforma. De igual manera, las comunicaciones entre estos procesadores se podrán realizar mediante rutinas de cualquier librería estándar de paso de mensajes instalada. Si Llamamos **subsistema de cómputo** a la unión de la plataforma paralela y de las librerías básicas de álgebra lineal instaladas; y **subsistema de comunicaciones** a la unión de la plataforma paralela y de las librerías de comunicaciones instaladas, entonces el modelo DLAM+ está formado por dos submodelos, uno para cada uno de estos subsistemas. A continuación se describirán detalladamente estos dos submodelos.

### Modelo del subsistema de cómputo

En las plataformas de hoy en día, con memorias altamente jerarquizadas, cobra mucha importancia la consideración del tiempo de acceso a los datos con los que se opera a la hora de modelizar el coste completo de llevar a cabo una operación aritmética, como ya plantearon Rauber y Rünger [RR01] y Zhang [Zha99] en sus respectivas propuestas de modelado. Este tiempo de acceso a los datos va a depender del nivel de la jerarquía de memoria donde se encuentren los mismos. Cuanto mayor sea la localidad que tienen los datos con los que se está operando, mayor será la posibilidad de encontrar esos datos en los niveles superiores de la jerarquía de memoria, lo que conllevará menor tiempo de acceso y, por tanto, un coste total inferior. Por esta razón, al igual que en el modelo de DLAM, las operaciones de cálculo que se pueden realizar en cada procesador están clasificadas en tres niveles por orden creciente de localidad de datos:

- Nivel 1: operaciones vectoriales, es decir, O($n$) operaciones sobre O($n$) datos.
- Nivel 2: operaciones matriz-vector, es decir, O($n^2$) operaciones sobre O($n^2$) datos.
- Nivel 3: operaciones matriz-matriz, es decir, O($n^3$) operaciones sobre O($n^2$) datos.

Estos niveles corresponden a los tres niveles de la librería BLAS o de cualquier librería que tenga una estructura similar. Los parámetros $k_1$, $k_2$ y $k_3$ representan los tiempos de llevar a cabo una operación de coma flotante de cada uno estos niveles, siendo, habitualmente, $k_1$>$k_2$>$k_3$. Esta separación del coste computacional en tres valores diferentes, frente al clásico enfoque de considerarlo como un único valor, trae consigo un modelo del sistema en el que queda mejor reflejado el verdadero coste total de realizar una operación de cómputo (acceso a los operandos, realización del cálculo y almacenamiento del resultado). Como ya se describió anteriormente, en el modelo DLAM los valores de estos tres parámetros se consideran constantes para una plataforma dada. En nuestra propuesta, de cara a mejorar este planteamiento del modelado del subsistema de cómputo, hemos realizado las siguientes variaciones respecto a DLAM:

- Cada parámetro $k_i$ será considerado realmente como un conjunto de parámetros diferentes $k_{i\_operacion}$ para las distintas operaciones del nivel $i$, con $i$ = 1, 2, 3. Esta modificación viene motivada por el hecho de que el patrón de acceso a los datos de operaciones diferentes no tiene por qué ser el mismo aunque éstas pertenezcan a un mismo nivel de los anteriormente señalados. Un cambio en el patrón de acceso a los datos provocará variaciones en la localidad de los datos operados y, por tanto, dará lugar a un coste distinto por operación realizada.

- Las operaciones de cómputo en cada procesador de una plataforma no tendrán que realizarse expresamente con una única versión de la librería BLAS, sino que podrán usarse distintas versiones genéricas de la librería BLAS, versiones de BLAS optimizadas para esta plataforma [wHen][wGoto] o incluso otras librerías que tengan una estructuración semejante a BLAS como, por ejemplo, ATLAS. Con esta variación buscamos dos objetivos principales. En primer lugar, conseguimos un modelo más general y portable al no estar restringido a una librería de cálculo concreta. En segundo lugar, como se describirá en los capítulos siguientes, podremos utilizar este modelo como herramienta para decidir qué librería utilizar en aquellas plataformas que cuenten con más de una instalada.

- Como consecuencia del punto anterior, dada una *operación* de un nivel $i$, el parámetro $k_{i\_operacion}$ tendrá un valor diferente dependiendo de la librería básica de álgebra linear que se utilice para llevar a cabo la operación.

- Dada una *operación* de un nivel $i$ de una de las librerías de álgebra linear instaladas, el parámetro $k_{i\_operacion}$ se considerará como un valor variable, que será función del esquema concreto de acceso a los datos que realice la rutina de alto nivel que llama a *operación*, a la que le pasa esos datos como parámetros. Este esquema de acceso viene marcado por el algoritmo que siga la rutina, la dimensión del problema a resolver y la forma en que estén almacenados los datos.

El modelo del subsistema de cómputo aritmético estará formado por el conjunto de estos parámetros $k_{i\_operacion}$, con $i$=1,2,3; para cada operación aritmética de nivel $i$ de cada librería básica de álgebra lineal instalada. Además, estos parámetros serán función del esquema concreto de acceso a los datos que se lleve a cabo en cada rutina. Por tanto, este modelo caracteriza las diferentes capacidades de este subsistema de cómputo formado por la unión de la plataforma hardware y de las librerías de álgebra linear instaladas.

## Modelo del subsistema de comunicaciones

Respecto a las comunicaciones entre diferentes procesadores, al igual que en DLAM, en nuestro planteamiento se considera que el tiempo para transferir $n$ palabras entre dos procesadores sería:

$$T_{COM}\left(n,t_s,t_w\right)=(t_s+nt_w)f \qquad 4.9$$

Es decir, el modelo clásico para las comunicaciones, ponderado por un factor $f$ cuyo valor dependerá de la plataforma y del tipo de comunicación. Sin embargo, en nuestra propuesta, de cara a mejorar este planteamiento del modelado de las comunicaciones, hemos realizado las siguientes variaciones respecto a DLAM:

- En el planteamiento de DLAM, los parámetros $t_s$ (tiempo de inicio de una comunicación, *startup time*) y $t_w$ (tiempo para enviar un mensaje de tamaño de una palabra, *word sending time*), se consideran, tal como se ha venido haciendo históricamente, como valores constantes para una plataforma dada. En el valor $t_w$ se está incluyendo el tiempo de acceder a cada dato unitario o palabra que se envía en el mensaje y este tiempo no tiene por qué ser una constante, pues ese dato puede encontrarse en diferentes niveles de la jerarquía de memoria de la plataforma. Por tanto, en DLAM+, este parámetro se considera que puede ser una variable que dependa del tamaño de datos enviados y del esquema de almacenamiento que estos datos tengan, como ya se planteaba en la aproximación de Getov *et al.* [GHH97] y en el modelo P-LogP [KBG00]. Es decir, como punto de partida, no consideramos que el tiempo de comunicaciones tenga que ser simplemente una función lineal del tamaño del mensaje, $n$, sino que contemplamos la posibilidad de que siga una función más compleja que tenga a $t_w$ como valor variable. De igual manera se puede tomar también a $t_s$ como variable, como ya se planteaba en la aproximación de Arruabarrena *et al.* [AAB+96].

- A diferencia de DLAM, en nuestra propuesta no restringimos las comunicaciones entre los procesadores únicamente a operaciones con BLACS, sino que, en general, consideramos que las comunicaciones se podrán llevar a cabo mediante cualquier librería de paso de mensajes como PVM, MPI o la propia BLACS. Con esta variación buscamos dos objetivos principales. En primer lugar, conseguimos un modelo más general y portable al no estar restringido a una librería de paso de mensajes concreta. En segundo lugar, podremos utilizar este modelo como herramienta para decidir qué librería de paso de mensajes utilizar en una plataforma que cuente con más de una instalada.

- El parámetro $f$ tiene en nuestro planteamiento una visión algo diferente de la que tenía originalmente en DLAM. Como anteriormente se ha comentado, en DLAM, para una operación de comunicación dada, este factor depende la topología física de la red de interconexión, ignorándose posibles colisiones. En DLAM+, dada una operación de comunicación colectiva, este parámetro va a caracterizar las prestaciones que ofrece la red de interconexión de la plataforma cuando adquiere una topología lógica determinada para llevar a cabo esa operación, incluyendo las colisiones que se generan al usarse. Por ejemplo, si la plataforma es un *cluster* de $p$ procesadores unidos por una red local, según el modelo DLAM el valor de $f$ para una operación de *broadcast* implementada con un algoritmo de distribución en forma de árbol binario, $f_{broad}(p)$, sería igual a $\log_2(p)$, ya que esta red de interconexión tiene capacidad de comportarse como un hipercubo lógico. Por el contrario, en nuestra propuesta, no se fija $f_{broad}(p)$, ya que éste va a depender de la capacidad dinámica que tenga cada red específica para gestionar esta operación de *broadcast*. Con ese algoritmo de distribución concreto, el valor de este parámetro se encontrará en el intervalo $[\log_2(p),\ p]$, como veremos en los resultados experimentales obtenidos en este trabajo. Incluso puede ocurrir que $f$ dependa también del tamaño de mensaje enviado. Por estas razones, será conveniente medir experimentalmente el valor de $f_{broad}(p, n)$ en cada sistema.

El modelo del subsistema de comunicaciones estará formado por el conjunto de estos parámetros, $t_s$, $t_w$, $f$; para cada librería de paso de mensajes instalada. Este modelo caracteriza, por tanto, la capacidad del subsistema de comunicaciones formado por la unión de la plataforma hardware y de las librerías de comunicaciones instaladas.

**Modelo del sistema completo**

Nuestro modelo del sistema completo estará formado por la unión de los modelos del subsistema de cómputo y de comunicaciones planteados. El conjunto de todos los parámetros del modelo caracterizarán, por tanto, la capacidad de cómputo y de comunicaciones del sistema completo, entendiendo al sistema como la unión de la plataforma hardware y de las librerías de cómputo aritmético y de comunicaciones instaladas. Los valores de estos parámetros podrán ser calculados mediante experimentación sobre el sistema durante la instalación del software correspondiente.

## 4.5 Modelo propuesto para rutinas de álgebra lineal

A la hora de construir un modelo analítico del tiempo de ejecución de una rutina, nuestro principal objetivo ha sido que éste sea una herramienta útil para decidir los valores de los parámetros ajustables de esta rutinas, y con ello minimizar su tiempo de ejecución. Por esta razón, en este modelo deben quedar reflejadas las características de cómputo y de comunicaciones de los algoritmos y las características del sistema sobre el que se ejecutará (hardware y software básico instalado) [CGG01]. Por tanto, el modelo analítico del tiempo de ejecución de una rutina de álgebra lineal consistirá en:

$$T_{EXEC} = f(n, SP, AP) \qquad 4.10$$

donde distinguimos:

- ***n***: el tamaño del problema a resolver.
- ***SP***: el conjunto de parámetros que caracterizan las capacidades de cómputo y de comunicaciones del sistema que influyen en el tiempo de ejecución de la rutina. Corresponderán a los parámetros que forman el modelo del sistema descritos en la sección anterior.
- ***AP***: una serie de parámetros algorítmicos (tamaño de bloque, número de procesadores a utilizar, configuración lógica de los procesadores, ...) cuyo valor podrá ser escogido a la hora de ejecutar la rutina.

Además, como ya se comentó en la sección anterior, el valor de cada parámetros del sistema no se considerará constante, sino que dependerá del tamaño del problema y de los valores de los *AP* que se tomen a la hora de ejecutar la rutina:

$$SP = f(n, AP) \tag{4.11}$$

Por otro lado, gracias a la estructura jerárquica de las librerías de álgebra lineal, el modelo del tiempo de ejecución de una rutina perteneciente a una librería de alto nivel, por ejemplo ScaLAPACK [BCC+97], se construirá en base a los diferentes modelos de las rutinas a las que llama en su código, mientras que para las rutinas de los niveles inferiores se modelizará su tiempo de ejecución directamente a partir de los parámetros básicos del sistema. Se formará, por tanto, una jerarquía de modelos en paralelo a la propia estructura jerárquica de las librerías de álgebra lineal ya existente.

A continuación se describirán las diferentes rutinas de álgebra lineal a las que se ha aplicado la metodología de ajuste automático de prestaciones que se propone en este trabajo. Para cada rutina, se verá la base teórica sobre la que se sustenta, el algoritmo que sigue y, finalmente, se detallará el modelo analítico de su tiempo de ejecución. Las rutinas utilizadas pertenecen a distintos niveles de la jerarquía clásica de librerías de álgebra lineal:

- Rutinas secuenciales del nivel de LAPACK y sus versiones paralelas del nivel de ScaLAPACK:
  - Factorización LU.
  - Factorización QR.
  - Factorización de Cholesky.
  - Métodos de Jacobi unilaterales para el problema de valores propios.
- Rutinas de alto nivel que estarían por encima de ScaLAPACK:
  - Resolución del problema de mínimos cuadrados en matrices Toeplitz.
  - Método "Elevación y Proyección" para la resolución del problema inverso aditivo de valores propios.

## 4.5.1 Factorización LU

La factorización LU es una descripción algebraica de alto nivel de la eliminación de Gauss [GVL96]. La factorización LU de una matriz $A \in \mathbf{R}^{m \times n}$ viene dada por $A = LU$, donde $L \in \mathbf{R}^{m \times n}$ es triangular inferior unitaria (con unos en la diagonal principal) y $U \in \mathbf{R}^{n \times n}$ es triangular superior. Esta rutina se utiliza como paso inicial en la resolución de un sistema de ecuaciones $Ax=b$, descomponiendo este problema en dos subproblemas de resolución de sendos sistemas de ecuaciones triangulares $Ly = b$ y $Ux = y$, de manera que $Ax = LUx = Ly = b$.

## Factorización LU secuencial por bloques

En esta sección se describirá la versión por bloques de la rutina de factorización LU que hemos implementado siguiendo los algoritmos presentados en [GVL96] y [DW95a].

Partiendo de una matriz $A \in \mathbf{R}^{n\times n}$, ésta se particiona siguiendo el diagrama de la Figura 4.1, donde $A_{00}$ es una submatriz $b\times b$, $A_{01}$ es una submatriz $b\times(n-b)$, $A_{10}$ es una submatriz $(n-b)\times b$ y $A_{11}$ es una submatriz $(n-b)\times(n-b)$. $L_{00}$ y $L_{11}$ son submatrices triangulares inferiores y unitarias, mientras que $U_{00}$ y $U_{11}$ son submatrices triangulares superiores. De manera que se puede escribir:

$$L_{00}U_{00} = A_{00} \qquad 4.12$$

$$L_{10}U_{00} = A_{10} \qquad 4.13$$

$$L_{00}U_{01} = A_{01} \qquad 4.14$$

$$L_{10}U_{01} + L_{11}U_{11} = A_{11} \qquad 4.15$$

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 \\ L_{10} & L_{11} \end{bmatrix} \times \begin{bmatrix} U_{00} & U_{01} \\ 0 & U_{11} \end{bmatrix}$$

Figura 4.1. Factorización LU por bloques de la matriz particionada *A*.

La ecuación 4.13 supone realizar una factorización LU en el bloque $A_{00}$ de dimensiones $b\times b$. Una vez realizado, se conocerán las matrices $L_{00}$ y $U_{00}$, con lo que se podrá resolver el sistema triangular superior de la ecuación 4.14 para obtener $L_{10}$ y el sistema triangular inferior de la ecuación 4.15 para obtener $U_{01}$. Finalmente la ecuación 4.16 se podría reformular de esta manera:

$$A_{11}^{'} = A_{11} - L_{10}U_{01} \qquad 4.16$$

con lo que ahora el problema de encontrar $L_{11}$ y $U_{11}$ se reduciría a realizar la factorización LU sobre la matriz $A'_{11}$ de dimensiones $(n-b)\times(n-b)$, en lugar de sobre *A*, realizando el mismo proceso que anteriormente se aplicó sobre *A* (Figura 4.2). La factorización completa de *A* se obtendrá repitiendo este proceso un total de *n*/*b* veces, sobrescribiéndose la matriz *A* por *L* y *U*.

Esta rutina que hemos programado (Figura 4.3) utilizaría, al igual que su equivalente *DGETRF* de la librería LAPACK, una serie de rutinas del propio LAPACK y de BLAS. Para realizar la factorización LU en el bloque $A_{00}$ se usaría la rutina *DGETF2* del nivel 2 de LAPACK. Para resolver los sistemas de ecuaciones 4.13 y 4.14 se usaría la rutina *DTRSM* del nivel 3 de BLAS. Finalmente para resolver la ecuación 4.15 (reformulada como la 4.16) se usaría *DGEMM* del nivel 3 de BLAS. El modelo del coste de este algoritmo será:

$$T_{EXEC} = \frac{2}{3}n^3 k_{3\_DGEMM} + bn^2 k_{3\_DTRSM} + \frac{1}{3}b^2 n k_{2\_DGETF2} \qquad 4.17$$

donde aparecen tres *SP* ($k_{3\_DGEMM}$, $k_{3\_DTRSM}$, $k_{2\_DGETF2}$) que corresponden al coste computacional de una operación básica realizada por cada una las tres rutinas de nivel inferior que son utilizadas (*DGEMM*, *DTRSM*, *DGETF2*) y cuyos valores dependerán del tamaño de problema, $n$, y del bloque de cálculo que se utilice, $b$, que es el único parámetro algorítmico de esta rutina.

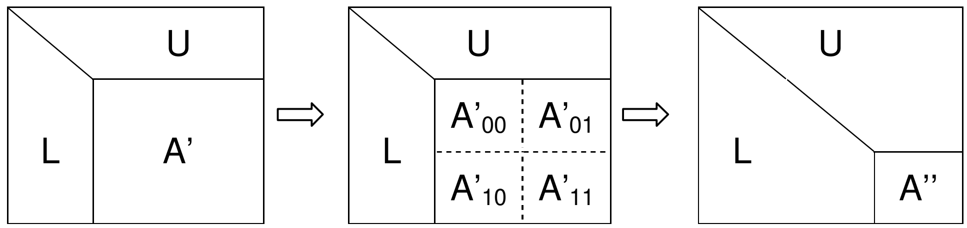


Figura 4.2. Avance en el proceso de factorización LU por bloques de la matriz particionada $A$.

REPETIR $n/b$ veces:
- Se particiona la parte de la matriz $A$ aún no resuelta en:
  - $A_{00}$, una submatriz $b\times b$
  - $A_{01}$, una submatriz $b\times(n-b)$
  - $A_{10}$, una submatriz $(n-b)\times b$
  - $A_{11}$, una submatriz $(n-b)\times(n-b)$
- *DGETF2*. Aplicar factorización LU del nivel 2 de LAPACK a la submatriz $A_{00}$:
  $$L_{00}\,U_{00} = A_{00}$$
- *DTRSM*. Resolver los sistemas de ecuaciones:
  $$L_{10}\,U_{00} = A_{10}$$
  $$L_{00}\,U_{01} = A_{01}$$
- *DGEMM*. Actualizar el resto de la matriz:
  $$A'_{11} = A_{11} - L_{10}\,U_{01}$$

FIN_REPETIR

Figura 4.3. Algoritmo de factorización LU secuencial por bloques.

## Factorización LU paralela por bloques

En este apartado se describirá la versión paralela por bloques de la rutina de factorización LU. Esta rutina la hemos implementado siguiendo los algoritmos presentados en [GVL96][DW95a] y [CDO+96], de manera que sería equivalente a la *PDGETRF* de la librería ScaLAPACK.

Antes de iniciar la factorización de la matriz $A$, ésta debe estar distribuida siguiendo una distribución cíclica por bloques entre los $r\times c$ procesadores de la plataforma, configurados lógicamente como una malla 2D, con $d$=max($r$,$c$). En esta rutina se realizan llamadas a una serie de rutinas de ScaLAPACK, PBLAS y LAPACK para cálculos aritméticos y de MPI para las comunicaciones. En el primer paso, para realizar la factorización LU en el bloque $A_{00}$ se usaría la rutina *PDGETF2* del nivel 2 de ScaLAPACK en el proceso (0,0). Para resolver los sistemas de ecuaciones 4.13 y 4.14 se usaría la rutina *PDTRSM* del nivel 3 de PBLAS en los procesos de la fila 0 y de la columna 0 de la malla, respectivamente. Finalmente para resolver la ecuación 4.15 se usaría *PDGEMM* del nivel 3 de PBLAS en todos los procesos de la malla, ya que $A_{11}$ estaría totalmente distribuida (Figura 4.4). En los siguientes pasos se haría la distribución de trabajo mostrada en la Figura 4.5, siguiendo el esquema completo del algoritmo (Figura 4.6).

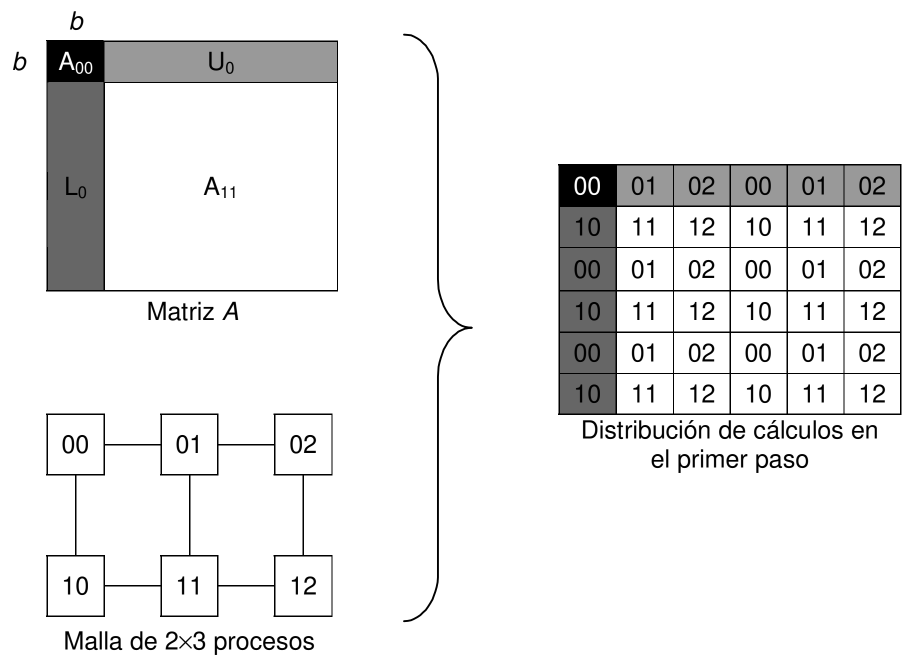


Figura 4.4. Distribución de los cálculos a realizar en un primer paso de la rutina LU paralela por bloques de tamaño $b \times b$, en una malla de 2×3 procesos, sobre la matriz $A \in \mathbf{R}^{n \times n}$, con $n$=6$b$.

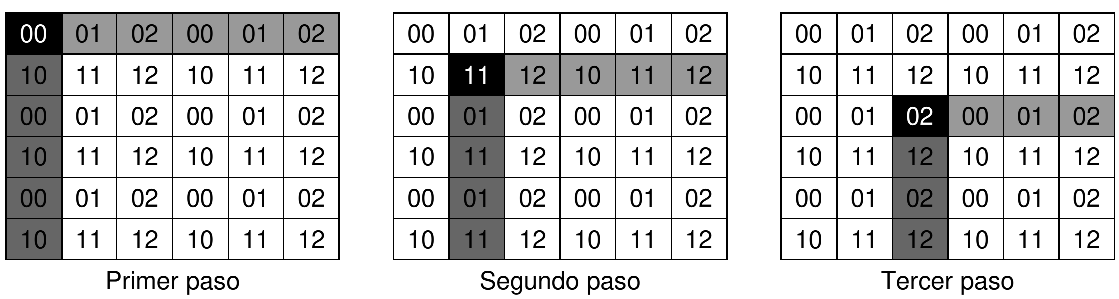


Figura 4.5. Distribución de los cálculos a realizar en los 3 primeros pasos de la rutina LU paralela por bloques de tamaño $b \times b$, en una malla de 2×3 procesos, sobre la matriz $A_{n \times n}$, con $n$=6$b$.

Distribuir la matriz $A$ entre los $r{\times}c$ procesadores de la plataforma, configurados lógicamente como una malla 2D
REPETIR $n/b$ veces:
  Se particiona la parte de la matriz $A$ aun no resuelta en:
    $A_{00}$, una submatriz $b{\times}b$
    $A_{01}$, una submatriz $b{\times}(n\text{–}b)$
    $A_{10}$, una submatriz $(n\text{–}b){\times}b$
    $A_{11}$, una submatriz $(n\text{–}b){\times}(n\text{–}b)$
  *PDGETF2*. En el proceso actual, aplicar factorización LU del nivel 2 de LAPACK a la submatriz $A_{00}$:
    $L_{00}\ U_{00} = A_{00}$
  *PDTRSM*. En la fila y columna de procesos actuales, resolver los sistemas de ecuaciones:
    $L_{10}\ U_{00} = A_{10}$
    $L_{00}\ U_{01} = A_{01}$
  *PDGEMM*. Distribuir $L_{10}$ y $U_{01}$ a todos los procesos, actualizar el resto de la matriz:
    $A'_{11}= A_{11} - L_{10}\, U_{01}$
FIN_REPETIR

Figura 4.6. Algoritmo de factorización LU paralela por bloques.

El modelo analítico del tiempo de ejecución de este algoritmo será:

$$T_{ARI} = \frac{2}{3}k_{3\_DGEMM}\frac{n^3}{p} + \frac{r+c}{p}bk_{3\_DTRSM}n^2 + \frac{1}{3}b^2k_{2\_DGETF2}n$$

$$T_{COM} = t_s\frac{2nd}{b} + t_w\frac{2n^2d}{p} \qquad 4.18$$

donde aparecen tres *SP* computacionales ($k_{3\_DGEMM}$, $k_{3\_DTRSM}$, $k_{2\_DGETF2}$) que, al igual que en la versión secuencial, corresponden al coste computacional de una operación básica realizada por cada una las rutinas de niveles inferiores que se utilizan. Además, aparecen dos *SP* de comunicaciones ($t_s$, $t_w$). El valor de los diferentes *SP* dependerán del tamaño de problema, $n$, y de los diferentes valores que se escojan para los *AP* que aparecen (bloque de cálculo, $b$, número de procesadores, $p$, configuración lógica de esos procesadores en una malla 2D de $r$ filas y $c$ columnas, siendo $d$ el máximo de $r$ y $c$).

## 4.5.2 Factorización QR

La factorización QR de una matriz $A \in \mathbf{R}^{m\times n}$ viene dada por $A = QR$, donde $Q \in \mathbf{R}^{m\times n}$ es ortogonal y $R \in \mathbf{R}^{n\times n}$ es triangular superior. Esta factorización se puede llevar a cabo mediante el uso de transformaciones de Householder o de Givens [GVL96][CDO+96]. Si se utilizan transformaciones de Householder el método consistiría en ir calculando las sucesivas matrices Householder $H_1, ..., H_n$, con $H_i = (I - \tau_i v_i v_i^T)$, donde $i = 1,..., n$. El vector de Householder $v_i$ es de longitud $m$, con valor 0 en las primeras $(i - 1)$ entradas y con valor 1 en la entrada $i$-ésima, y $\tau_i = 2/(v_i v_i^T)$. De manera que si $Q = H_1 \cdot\cdot\cdot H_n$, entonces $Q^TA = R$ es triangular superior.

La parte triangular superior de $A$ se sobrescribe por la parte triangular superior de $R$, los sucesivos vectores $v_i$ sobrescriben la parte triangular inferior de $A$ y $\tau_i$ se guarda en un vector.

## Factorización QR secuencial por bloques

En esta sección describiremos la versión por bloques de la rutina de factorización QR siguiendo los algoritmos presentados en [GVL96] y [CDO+96].

Partiendo de una matriz $A$ de dimensiones $n \times n$, ésta se particiona siguiendo el diagrama de la Figura 4.7, donde la submatriz $A_0$ es de tamaño $n \times b$ y estaría formada por los bloques $A_{00}$ y $A_{10}$, mientras que la submatriz $A_1$ es de tamaño $n \times (n-b)$ y estaría formada por los bloques $A_{01}$ y $A_{11}$. El bloque $A_{00}$ es de tamaño $b \times b$, $A_{01}$ es de tamaño $b \times (n-b)$, $A_{10}$ es de tamaño $(n-b) \times b$ y $A_{11}$ es de tamaño $(n-b) \times (n-b)$.

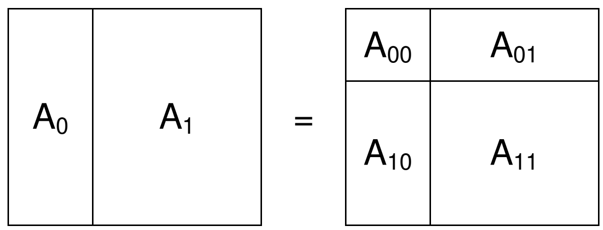


Figura 4.7. Particionamiento de la matriz $A$ para la factorización QR por bloques.

En primer lugar, se realiza una factorización QR sobre $A_0$. Entonces se puede calcular la matriz $Q$ que acumula las transformaciones realizadas en $A_0$ como $Q = H_1 H_2 \cdot\ \cdot\ \cdot\ H_b = I - VTV^T$, donde $T$ es una matriz triangular superior de dimensiones $b \times b$ y la $i$-ésima columna de $V$ es el vector de Householder $v_i$ descrito anteriormente. Finalmente se actualiza el resto de la matriz, $A_1 = Q^T A_1$ (Figura 4.8). Ahora se repetiría el proceso para la submatriz $A'_{11}$. La factorización completa de $A$ se obtendrá repitiendo este proceso un total de $n/b$ veces, sobrescribiéndose la matriz $A$ por $V$ y $R$.

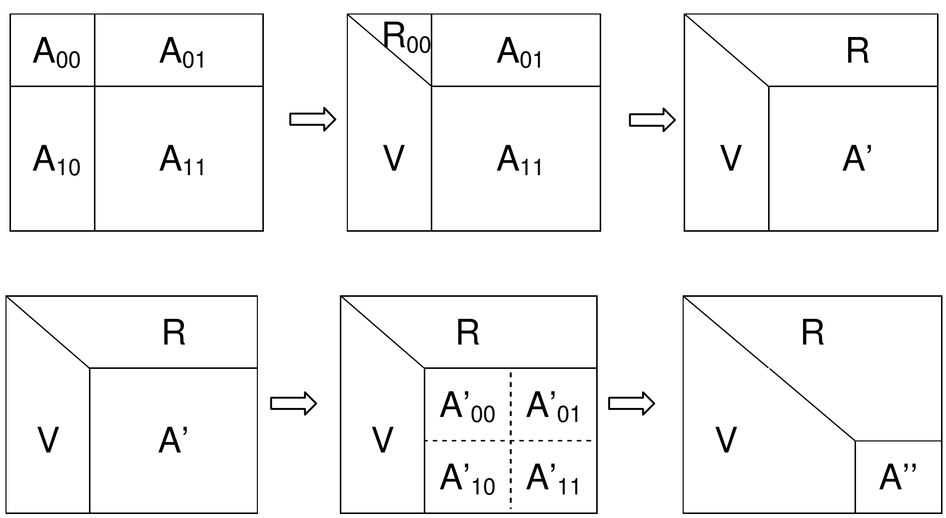


Figura 4.8. Avance en el proceso de factorización QR por bloques de la matriz particionada $A$.

En la librería LAPACK, el nombre de esta rutina es *DGEQRF*. En el código de ésta, las rutinas que se llaman son: *DGEQR2*, del nivel 2 de LAPACK, para realizar la factorización sobre $A_1$; *DLARFT*, del nivel 2 de LAPACK, para calcular el factor $T$; *DGEMM* y *DTRMM,* del nivel 3 de BLAS, para actualizar $A_2$ (Figura 4.9). Con lo que el modelo analítico que refleja el coste de esta rutina será:

$$T_{EXEC} = \frac{4}{3} n^3 k_{3\_DGEMM} + \frac{1}{2} n^2 b k_{3\_DTRMM} + n^2 b k_{2\_DGEQR2} + \frac{1}{2} n^2 b k_{2\_DLARFT} \qquad 4.19$$

donde aparecen cuatro *SP* ($k_{3\_DGEMM}$, $k_{3\_DTRMM}$, $k_{2\_DGEQR2}$, $k_{2\_DLARFT}$) que corresponden al coste computacional de una operación básica realizada por cada una las rutinas de nivel inferior que se utilizan y cuyos valores, al igual que en las rutinas anteriores, dependerán del tamaño de problema, $n$, y del bloque de cálculo que se utilice, $b$, que es el único parámetro algorítmico de la versión secuencial de esta rutina.

REPETIR $n/b$ veces:
    Se particiona la parte de la matriz $A$ aun no resuelta en:
        $A_{00}$, una submatriz $b \times b$
        $A_{01}$, una submatriz $b \times (n-b)$
        $A_{10}$, una submatriz $(n-b) \times b$
        $A_{11}$, una submatriz $(n-b) \times (n-b)$
    *DGEQR2*. Calcular la factorización QR de la submatriz $A_0$
    *DLARFT*. Calcular el factor triangular $T$ del bloque reflector $Q$
    *DLARFB*. Aplicar $Q^T$ al resto de la matriz:
        *DGEMM*: $W = V^T A_1$
        *DTRMM*: $W = T^T W$
        *DGEMM*: $A'_1 = A_1 - VW$
FIN_REPETIR

Figura 4.9. Algoritmo de factorización QR secuencial por bloques.

## Factorización QR paralela por bloques

En este apartado describiremos la versión paralela por bloques de la rutina de factorización QR. Antes de iniciar la factorización QR de la matriz $A$, al igual que ocurría con la LU, ésta debe estar distribuida entre los procesadores de la plataforma siguiendo una descomposición cíclica por bloques. En esta descomposición, bloques de datos consecutivos se distribuyen cíclicamente entre los procesadores, tomando, de cara a simplificar la explicación del algoritmo, los bloques de forma cuadrada $b \times b$.

La función que realiza la factorización QR en la librería ScaLAPACK es *PDGEQRF*. En el primer paso, las rutinas llamadas son: *PDGEQR2*, del nivel 2 de ScaLAPACK, para realizar la factorización sobre $A_0$, en los procesos de la columna 0 de la malla; *PDLARFT*, del nivel 2 de ScaLAPACK, para calcular el factor $T$, en los procesos de la columna 0 de la malla; *PDGEMM y PDTRMM,* del nivel 3 de BLAS, para actualizar $A_2$, en todos los procesos de la malla. En los sucesivos pasos, las filas y columnas de los procesos que actúan irán cambiando siguiendo el orden indicado en la Figura 4.10. Un esquema del algoritmo sería el mostrado en la Figura 4.11.

El modelo del coste de esta rutina será:

$$T_{ARI} = \frac{4n^3 k_{3\_DGEMM}}{3p} + \frac{n^2 b k_{3\_DTRMM}}{4c} + \frac{n^2 b k_{2\_DGEQR2}}{r} + \frac{n^2 b k_{2\_DLARFT}}{2r}$$

$$T_{COM} = t_s \left[ \frac{n}{b} \left( (2+3b) f_{broad}(r) + 2 f_{broad}(c) \right) \right] + t_w \left[ \frac{n^2}{2} \left( \frac{2(r-1)}{r^2} + \frac{f_{broad}(r)}{c} + \frac{f_{broad}(r)}{r} \right) + nb f_{broad}(p) \right] \qquad 4.20$$

donde aparecen cuatro *SP* computacionales ($k_{3\_DGEMM}$, $k_{3\_DTRMM}$, $k_{2\_DGEQR2}$, $k_{2\_DLARFT}$) que corresponden al coste computacional de cada operación básica realizada en las rutinas de niveles inferiores que se utilizan. Además, aparecen tres *SP* de comunicaciones ($t_s$, $t_w$, $f_{broad}$). El valor de los diferentes *SP* dependerá, como en los casos anteriores, del tamaño de problema, $n$, y de los diferentes valores que se escojan para los *AP* que aparecen (bloque de cálculo, $b$, número de procesadores, $p$, configuración lógica de esos procesadores en una malla 2D de $r$ filas y $c$ columnas).

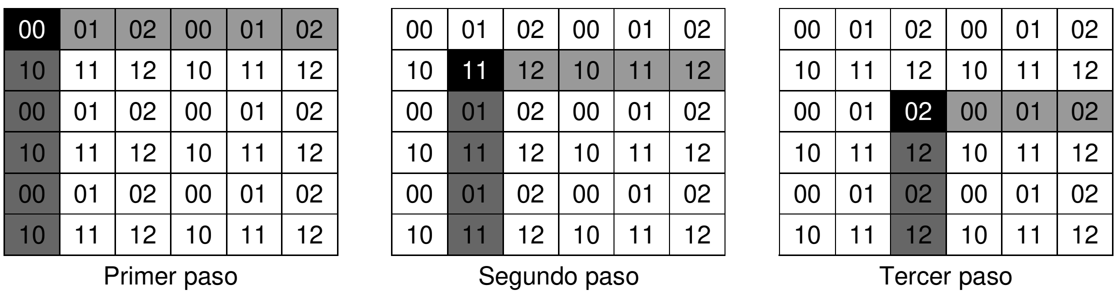


Figura 4.10. Distribución de los cálculos a realizar en los 3 primeros pasos de la rutina QR paralela por bloques de tamaño $b \times b$, en una malla de 2×3 procesos, sobre la matriz $A_{n \times n}$, con $n$=6$b$.

Distribuir la matriz $A$ entre los $r \times c$ procesadores de la plataforma, configurados lógicamente como una malla 2D
REPETIR $n/b$ veces:
  Se particiona la parte de la matriz $A$ aun no resuelta en:
    $A_{00}$, una submatriz $b \times b$
    $A_{01}$, una submatriz $b \times (n-b)$
    $A_{10}$, una submatriz $(n-b) \times b$
    $A_{11}$, una submatriz $(n-b)$
    $A_{11}$, una submatriz $(n-b) \times (n-b)$
  *PDGEQR2*. En el proceso actual, calcular la factorización QR de la submatriz $A_0$
  *PDLARFT*. En la columna actual de procesos, calcular el factor triangular $T$ del bloque reflector $Q$
  *DLARFB*. Distribuir $V$, $W$ y $T$ al resto de procesos, aplicar $Q^T$ al resto de la matriz:
    *PDGEMM*: $W = V^T A_1$
    *PDTRMM*: $W = T^T W$
    *PDGEMM*: $A'_1 = A_1 - VW$
FIN_REPETIR

Figura 4.11. Algoritmo de factorización QR paralela por bloques.

### 4.5.3 Factorización de Cholesky

La factorización de Cholesky de una matriz $A \in \mathbf{R}^{n\times n}$, simétrica definida positiva, viene dada por $A = LL^T$, con $L \in \mathbf{R}^{n\times n}$, triangular inferior y cuyos elementos de la diagonal son positivos.

#### Factorización de Cholesky secuencial por bloques

En esta sección describiremos la versión por bloques de la rutina de factorización de Cholesky siguiendo los algoritmos presentados en [GVL96] y [CDO+96].

Partiendo de que la parte triangular inferior de $A$ está almacenada en un array bidimensional, conforme se vayan calculando los elementos de $L$ éstos se irán sobrescribiendo sobre la parte triangular inferior de la propia $A$. La matriz $A$ de partida se podría ver dividida en los bloques $A_{00}$ , $A_{10}$ y $A_{11}$ (Figura 4.12), siendo $A_{00}$ de tamaño $b\times b$, $A_{10}$ de tamaño $(n\text{-}b)\times b$ y $A_{11}$ de tamaño $(n\text{-}b)\times(n\text{-}b)$. De igual manera, $L_{00}$ es de tamaño $b\times b$, $L_{10}$ de tamaño $(n\text{-}b)\times b$ y $L_{11}$ de tamaño $(n\text{-}b)\times(n\text{-}b)$.

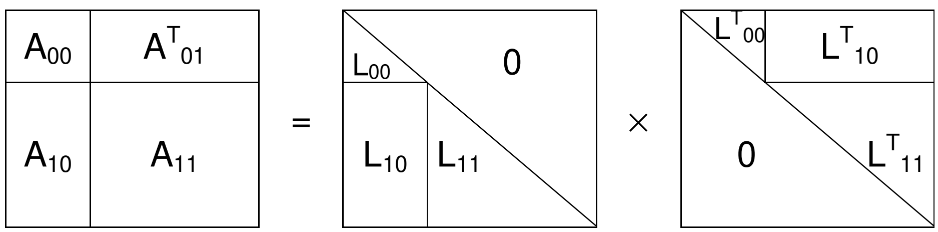


Figura 4.12. Factorización de Cholesky por bloques de la matriz particionada $A$.

En una primera pasada, el bloque $L_{00}$ se calcula realizando una factorización de Cholesky sin bloques sobre $A_{00}$. Tras ello, se calcula $L_{10}$ mediante $L_{10}= A_{10}\ (L^T{}_{00})^{-1}$. Finalmente, para completar esta primera pasada, se actualizaría la parte restante de la matriz: $A'=A_{11}-(L_{10}L^T{}_{10})$. El algoritmo continuaría aplicando en la siguiente pasada estas mismas operaciones a la submatriz $A'$ (Figura 4.13), al estilo de los algoritmos por bloques de las factorizaciones LU y QR anteriormente descritos.

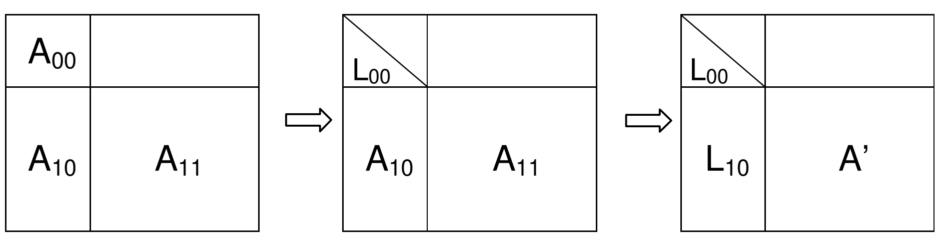


Figura 4.13. Avance en el proceso de factorización de Cholesky por bloques de la matriz particionada $A$.

En la librería LAPACK, el nombre de esta rutina es *DPOTRF*. En su código, las rutinas que se invocan son: *DPOTF2*, del nivel 2 de LAPACK, para realizar la factorización sobre $A_{11}$;

*DTRSM*, del nivel 3 de BLAS, para calcular $L_{21}$; *DGEMM y DSYRK,* del nivel 3 de BLAS, para actualizar $A_{22}$ (Figura 4.14). Con lo que el modelo del coste de esta rutina será:

$$T_{EXEC} = 2k_{3\_DGEMM}\left(\frac{n^3}{6}-\frac{n^2b}{2}+\frac{nb^2}{3}\right)+k_{3\_DSYRK}\left(\frac{n^2}{2}-\frac{nb}{2}\right)(b+1)+ \\ k_{3\_DTRSM}\left(\frac{n^2b}{2}-\frac{nb^2}{2}\right)+k_{2\_DPOTF2}\left(\frac{nb^2}{3}\right) \tag{4.21}$$

donde aparecen cuatro *SP* ($k_{3\_DGEMM}$, $k_{3\_DSYRK}$, $k_{2\_DTRSM}$, $k_{2\_DPOTF2}$) que, como en las rutinas anteriormente vistas, corresponden al coste computacional de una operación básica realizada por cada una las rutinas de nivel inferior que son utilizadas y cuyos valores dependerán del tamaño de problema, *n*, y del bloque de cálculo que se utilice, *b*.

```
REPETIR n/b veces:
        Se particiona la parte de la matriz A aun no resuelta en:
                A00, una submatriz b×b
                A01, una submatriz b×(n–b)
                A10, una submatriz (n–b)×b
                A11, una submatriz (n–b)×(n–b)
        DPOTF2. Calcular la factorización de Cholesky de la submatriz A00
        DTRSM. Calcular la submatriz L10
        DSYRK y DGEMM. Actualizar el resto de la matriz:
                A'1 = A1 – L21 L^T 21
FIN_REPETIR
```

Figura 4.14. Algoritmo de factorización de Cholesky secuencial por bloques.

### 4.5.4 Métodos de Jacobi unilaterales para el problema de valores propios

Dada una matriz *A* compleja y cuadrada, se llaman valores propios de *A* a los números complejos $\lambda$ para los que existe un vector complejo *x* que cumple la ecuación $Ax = \lambda x$. El vector *x* se llama vector propio asociado al valor propio $\lambda$.

El método de Jacobi es el método más antiguo que se utiliza para obtener los valores y vectores propios de una matriz simétrica $A \in \mathbf{R}^{n\times n}$. Actualmente este método ha recobrado gran interés debido a su paralelismo inherente, a su buena estabilidad y a su escalabilidad [DV92]. Este método va construyendo una secuencia de matrices $\{A_l\}$, donde $A_{l+1}=Q_lA_lQ^T_l$, $l=1,2,3, ...,$ con $A_1=A$; y siendo $Q_l$ una rotación de Givens en el plano $(i,j)$, con $1\le i, j\le n$, que anula los elementos $a_{ij}$ y $a_{ji}$ de $A_l$. La secuencia $\{A_l\}$ converge a una matriz diagonal $D=Q_kQ_{k-1}...Q_2Q_1AQ^T_1Q^T_2...Q^T_{k-1}Q^T_k$. Los elementos diagonales de *D* son los valores propios de *A* y las columnas de la matriz *V,* donde las rotaciones son acumuladas ($V=Q^T_1Q^T_2...Q^T_{k-1}Q^T_k$), son los vectores propios de *A*.

Las distintas maneras de elegir los pares $(i,j)$ han dado lugar a diferentes versiones del método de Jacobi. El método clásico procede eligiendo en cada iteración como elemento a anular el de mayor valor absoluto entre los no diagonales. Otros métodos, llamados cíclicos, proceden realizando sucesivos barridos, anulando en cada barrido una vez cada elemento no diagonal. Los diferentes órdenes que se pueden obtener con los $n/2(n-1)$ índices correspondientes a los elementos no diagonales se llaman órdenes de Jacobi. Algunos de estos órdenes serían: por filas [FH60], par-impar [Sam71], Eberlein [EP90], Round-Robin [BL82] y catterpillar-track [WOH84]. Por otro lado, el método semiclásico, introducido por Giménez [Gim95], utiliza características del método clásico y de los cíclicos de manera que se acelere la convergencia (se reduzca el número de barridos que se necesita con los métodos cíclicos) con un pequeño aumento del tiempo de ejecución por barrido.

En una versión por bloques, las matrices $A$ y $V$ son divididas en columnas y filas de bloques de tamaño $b\times b$ y, a su vez, estos bloques son agrupados en bloques mayores de tamaño $2b\times 2b$. Cada $Q_l$ representa un conjunto de rotaciones que anula los elementos dentro del bloque $A_l$. Estas rotaciones se calculan realizando un barrido sobre los elementos del interior de $A_l$. En un principio, los elementos subdiagonales que pertenecen a bloques diagonales no son anulados. Para corregir esta situación, los bloques del primer conjunto de Jacobi son considerados de tamaño $2b\times 2b$, añadiendo a cada bloque los dos bloques diagonales adyacentes y el bloque simétrico.

El trabajo en el interior de cada bloque se realiza usando rutinas de BLAS-1. Las rotaciones se acumulan formando una matriz $Q$ de tamaño $2b\times 2b$. Finalmente, las correspondientes columnas y filas de bloques de tamaño $2b\times 2b$ de la matriz $A$ y las filas de bloques de $V$ se actualizan usando $Q$. Estas multiplicaciones matriciales se realizan con la rutina *DGEMM* de BLAS-3. Tras completar un conjunto de rotaciones de bloques, se intercambian filas y columnas de bloques siguiendo el orden que se esté utilizando. En las rutinas que hemos desarrollado se ha utilizado un orden par-impar, ya que de esta manera se simplifica esta implementación por bloques, así como su versión paralela. El movimiento de datos se puede incluir en la actualización de las matrices realizando la permutación correspondiente en la matriz de rotaciones antes de realizar las multiplicaciones. Con este movimiento de datos, se traen a la diagonal los siguientes bloques de tamaño $b\times b$ que se deben anular y el proceso continúa de manera similar a como se llevó a cabo en el primer paso. El coste por barrido de este algoritmo por bloques cuando se calculan los valores y los vectores propios es:

$$T_{ARI} = 8n^3 k_{3\_DGEMM} - 16n^2 b k_{3\_DGEMM} + 12n^2 b k_{1\_DROT} \qquad 4.22$$

Una versión paralela para un anillo de $p=q/2$ procesadores, siendo $q=n/(2bk)$, se puede obtener asignando a cada procesador $P_i$ ($i$=0, ..., $p$-1) las filas $i$ y $q$-1-$i$ de las matrices $A$ y $V$. Por tanto, cada procesador contiene los bloques $A_{ij}$ y $A_{q-1-i,j}$ ($j$=0, ..., $i$) de la matriz $A$, y los bloques $V_{ij}$ y $V_{q-1-i,j}$ ($j$=0, ..., $q$) de $V$.

El coste aritmético por barrido es:

$$T_{ARI} = \frac{8n^3 k_{3\_DGEMM} - 8n^2 b k_{3\_DGEMM} + 12n^2 b k_{1\_DROT}}{p} \qquad 4.23$$

y el coste de las comunicaciones por barrido es:

$$T_{COM} = \frac{n}{b}(p+3)t_s + \left(10n^2 + 4nb - \frac{n^2}{2p}\right)t_w \qquad 4.24$$

A continuación veremos una versión unilateral secuencial de este método y, tras ello, se describirán las dos versiones paralelas de este algoritmo que hemos implementado. La primera versión paralela supone una configuración lógica de los procesadores en forma de anillo (versión 1D), mientras que la segunda versión está orientada a configuraciones en forma de malla bidimensional (versión 2D).

## Algoritmo unilateral

El algoritmo unilateral trabaja sobre las matrices $B_0=A$ y $W_0=I$, obteniendo $B_{r+1}=Q_rB_r$ y $W_{r+1}=Q_rW_r$, con $Q_r$ la matriz de rotación que anula un elemento no diagonal de la matriz $A=B_rW^T_r=Q_{r-1}Q_{r-2}...Q_0(B_0W^T_0)Q^T_0...Q^T_{r-2}Q^T_{r-1}$ [Cha63]. Es decir, a diferencia del método bilateral anteriormente descrito, las multiplicaciones por la matriz de rotaciones $Q_r$ se realizan siempre por un único lado, concretamente por la izquierda (premultiplicaciones).

Para anular $a_{ij}$ es necesario calcular $a_{ii}$, $a_{jj}$ y $a_{ij}$, ya que el algoritmo trabaja con las matrices $B_r$ y $W_r$, y no con la matriz $A_r$. Estos elementos se pueden obtener mediante tres productos de vectores. Tras ello, se actualizan las filas *i* y *j* de $B_r$ y de $W_r$. Si se guardan los elementos diagonales de $A_r$ en un vector auxiliar, no sería necesario calcular $a_{ii}$ ni $a_{jj}$ cada vez.

## Algoritmo unilateral secuencial por bloques

Nuestra propuesta de un algoritmo unilateral por bloques [GCR+98] se diseñó combinando las ideas introducidas en [Cha63] junto con la propuesta de algoritmo bilateral por bloques expuesta en [GHV+97].

Las matrices *B* y *W*, de tamaño $n\times n$, se dividen en bloques de tamaño $b\times b$, y los bloques de $A=BW$ son tratados usando un orden clásico par-impar. Inicialmente, los $n/(2b)$ bloques correspondientes al primer conjunto de Jacobi son tratados realizándose un barrido bilateral en el interior de los bloques $2b\times 2b$ de la matriz *A*, y acumulando las rotaciones. Estas operaciones se realizan mediante llamadas a la rutina *DROT* del nivel 1 de BLAS. Tras ello, las matrices *B* y *W* se actualizan mediante la multiplicación de las matrices de rotación, de tamaño $2b\times 2b$, por los correspondiente bloques de *B* y *W*, de tamaño $2b\times n$, usando para ello la rutina *DGEMM* de nivel 3 de BLAS. Tras completar un conjunto de rotaciones de bloques, se lleva a cabo un movimiento de filas de *B* y *W*. En el resto de pasos, será necesario calcular los bloques $A_{ii}$, $A_{ij}$ y $A_{jj}$, debido a que no se está trabajando directamente con *A*. Usando estos bloques, se forma un bloque de tamaño $2b\times 2b$ y el proceso continúa como anteriormente. Si se almacenan los bloques diagonales de *A* en una estructura auxiliar no sería necesario calcular $A_{ii}$ y $A_{jj}$ en cada paso. En la Figura 4.15 se muestra el esquema del algoritmo.

```
B=A
W=I
D=diag_bloques(A)
MIENTRAS convergencia no alcanzada HACER
        PARA cada conjunto de Jacobi S HACER
                PARA cada bloque Aij en conjunto S HACER
                        Obtener bloques Aii, Ajj  de D
                        DGEMM: Calcular bloque Aij usando B y W
                        DROT: Realizar barrido interior del bloque Aij acumulando rotaciones en Qij
                        DGEMM: Premultiplicar: W = Qij W; B = Qij B
                FIN_PARA
                Permutar bloques de filas de B, W y D
        FIN_PARA
FIN_MIENTRAS
```

Figura 4.15. Algoritmo de Jacobi unilateral secuencial por bloques.

En cada barrido, el cálculo de los bloques de $A$ a partir de $B$ y $W$ tiene un coste de $n^3 k_{3\_DGEMM}$. El barrido en el interior de los bloques de $A$ calculando las matrices de rotaciones tiene un coste de $12n^2 b k_{1\_DROT}$; y la actualización de $B$ y $W$ premultiplicando por las matrices de rotación tiene un coste de $8n^3 k_{3\_DGEMM}$. Por tanto, el coste aritmético por barrido sería:

$$T_{EXEC} = 9n^3 k_{3\_DGEMM} + 12n^2 b k_{1\_DROT} \qquad 4.25$$

## Algoritmo unilateral paralelo por bloques. Versión 1D

Esta versión paralela, introducida en [GCR+98], se denomina versión unidimensional debido a que presupone una topología de los procesadores en forma de anillo (malla unidimensional). En este algoritmo, $k$ bloques consecutivos de tamaño $2b \times n$, con $n=2bkp$, de las matrices $B$ y $W$ se asignan a cada procesador (Figura 4.16).

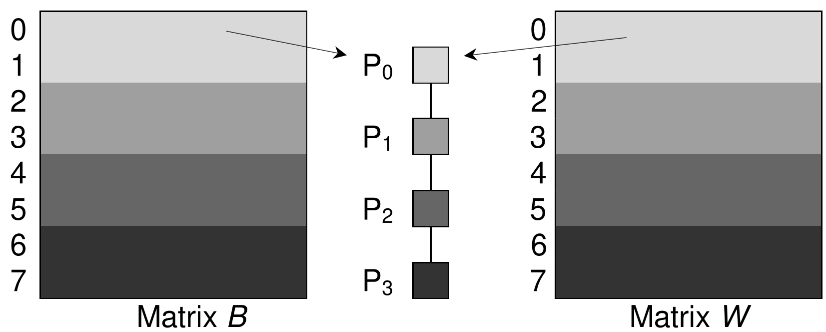


Figura 4.16. Distribución de datos en el algoritmo unilateral, versión 1D. Con $n/b$=8, $p$=4 y $k$=2.

En esta versión paralela no es necesario distribuir las matrices de rotación a los distintos procesadores, ya que cada uno actualiza únicamente las filas de bloques que contiene. Las únicas comunicaciones son las que se realizan entre cada dos pasos del algoritmo de cara a agrupar los datos de acuerdo al siguiente conjunto de Jacobi, es decir, en los pasos pares, bloques de tamaño $b \times n$ de $B$ y de $W$, y un bloque diagonal de tamaño $b \times b$ de $A$, son enviados desde $P_i$ a $P_{i-1}$, con $i=1, 2, \ldots, p-1$, y en los pasos impares estas mismas comunicaciones son llevadas a cado desde $P_{i-1}$ a $P_i$. El esquema del algoritmo es mostrado en la Figura 4.17. El coste aritmético por barrido sería:

$$T_{ARI} = \frac{9n^3 k_{3\_DGEMM} + 12n^2 b k_{1\_DROT}}{p} \qquad 4.26$$

Por otro lado, el movimiento de filas de bloques de $B$ y $W$ de tamaño $b \times n$ entre cada procesador y su vecino tendrá un coste por barrido de:

$$T_{COM} = \frac{2n}{b}\left(t_s + (2nb + b^2)t_w\right) \qquad 4.27$$

```
B=A
W=I
D=diag_bloques(A)
En CADA proceso q;  q=0, 1, ..., p-1:
B_q = porción de B en proceso q; W_q = porción de W en proceso q; D_q = porción de D en proceso q
MIENTRAS convergencia no alcanzada HACER
        PARA cada conjunto de Jacobi S HACER
                PARA cada bloque A_ij perteneciente al proceso q HACER
                        Obtener A_ii y A_jj de D_q
                        DGEMM: Calcular bloques A_ij usando  B_q y W_q
                        DROT: Realizar barrido interior del bloque A_ij acumulando rotaciones en Q_ij
                        DGEMM: Premultiplicar: W_q = Q_ij W_q; B_q = Q_ij B_q
                FIN_PARA
                Permutar bloques de filas de B_q y W_q
                Enviar primer (último) bloque de filas de B_q, W_q y D_q  a proceso q-1 (q+1)
                Recibir primer (último) bloque de filas de B_q, W_q y D_q del proceso q+1 (q-1)
        FIN_PARA
FIN_MIENTRAS
```

Figura 4.17. Algoritmo de Jacobi unilateral paralelo por bloques. Versión 1D.

## Algoritmo unilateral paralelo por bloques. Versión 2D

En [CG00] proponemos una generalización para mallas 2D de procesadores de este algoritmo unilateral por bloques. Esta versión se diseñó combinando las ideas expuestas en el apartado anterior con las del algoritmo 2D sin bloques propuesto en [RVG98]. En esta nueva versión del algoritmo unilateral consideramos una topología lógica de los procesadores en forma de malla rectangular $2^r \times 2^c$ (Figura 4.18). El objetivo de esta versión es reducir el coste de las comunicaciones que tiene la versión unidimensional descrita anteriormente. Los cálculos son organizados de tal manera que las transformaciones que en la versión 1D eran aplicadas independientemente por cada nodo son ahora llevadas a cabo por una fila de nodos de la malla 2D.

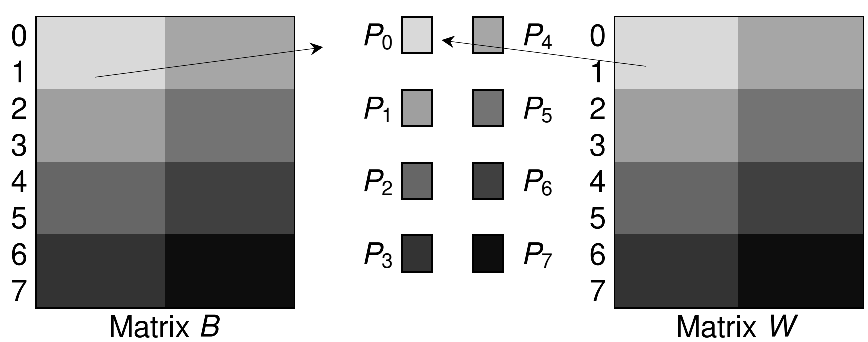


Figura 4.18. Distribución de los datos en el algoritmo unilateral, versión 2D. Con $n/b$=8, $2^r$=4, $2^c$=2 y $k$=2.

Con esta topología, se reduce el número de filas de la malla en comparación con el algoritmo 1D. Como el número de pasos por barrido depende del número de filas de la malla, se puede reducir el tiempo de ejecución del algoritmo debido a la disminución en el número de intercambios de filas de bloques que se realiza al final de cada paso. Este intercambio es lo que denominamos Comunicaciones Verticales (CV). Sin embargo, debido a que ahora los nodos de una misma fila de la malla deben cooperar para calcular y aplicar las transformaciones, aparece un nuevo tipo de comunicaciones a lo largo de estas filas (Comunicaciones Horizontales, CH). El coste de las CH dependerá del número de columnas de la malla de procesadores. El esquema del algoritmo es mostrado en la Figura 4.19.

```
B=A
W=I
D=diag_bloques(A)
En CADA proceso q;  q=0, 1, ..., p-1:
B_q = porción de B en proceso q; W_q = porción of  W en proceso q; D_q = poción de D en proceso q
MIENTRAS convergencia no alcanzada HACER
        PARA cada conjunto de Jacobi S HACER
                PARA cada bloque A_ij perteneciente al proceso q en el conjunto S HACER
                        Obtener A_ii, A_jj, de D_q
                        DGEMM: Calcular parcialmente bloques A_ij usando  B_q and W_q
                        Comunicar y acumular el bloque A_ij con procesos en la misma fila
                        DROT: Realizar barrido interior del bloque A_ij acumulando rotaciones en Q_ij
                        DGEMM: Premultiplicar: W_q = Q_ij W_q; B_q = Q_ij B_q
                FIN_PARA
                Permutar bloques de filas de B_q y W_q
                Enviar primer (último) bloque de filas de B_q, W_q y D_q  a proceso q-1 (q+1)
                Recibir primer (último) bloque de filas de B_q, W_q y D_q del proceso q+1 (q-1)
        FIN_PARA
FIN_MIENTRAS
```

Figura 4.19. Algoritmo de Jacobi unilateral paralelo por bloques. Versión 2D.

En esta versión 2D, en cada barrido, el coste total aritmético es el mismo que en la versión 1D. La diferencia la encontramos en la división del trabajo, ya que los diferentes procesadores en cada fila de la malla realizan los barridos sobre los mismos bloques. El coste por barrido será:

$$T_{ARI} = \frac{9n^3 k_{3\_DGEMM}}{p} + \frac{12n^2 bk_{1\_DROT}}{2^r} \qquad 4.28$$

En cuanto al coste de las comunicaciones, tendremos el coste de las Comunicaciones Verticales (CV), al estilo del algoritmo 1D, como:

$$T_{CV} = \frac{2n}{b}\left(t_s + (2\frac{n}{2^c}b + b^2)t_w\right) \qquad 4.29$$

y el coste de las comunicaciones horizontales entre nodos de las misma fila de la malla, con un coste por barrido, incluyendo el coste de la acumulación parcial de vectores:

$$T_{CH} = \frac{n^2}{b^2 2^{r+1}}\left((2c-1)(t_s + b^2 t_w) + (c-1)b^2 k_{1\_DCOPY}\right) \qquad 4.30$$

## Comparativa de los algoritmos

En [CG00] se comprobó teórica y experimentalmente que el tiempo de ejecución de nuestra propuesta de algoritmo unilateral por bloques en plataformas paralelas era notablemente inferior a las diferentes versiones bilaterales del algoritmo. Esta reducción se debe a dos razones, la primera es que en el unilateral el coste de las comunicaciones (4.27 ó 4.29 + 4.30) es menor que en el bilateral (4.24). La segunda razón es que en el unilateral el acceso a los datos de las matrices con las que se opera se realiza siempre en el mismo orden en el que están almacenados, bien por filas o por columnas, mientras que en el bilateral es necesario acceder en ambos sentidos continuamente, por filas y por columnas, disminuyendo la localidad espacial. De esta manera, aunque el tiempo de cómputo aritmético aparece como teóricamente superior en el unilateral (4.26 ó 4.28) frente al bilateral (4.23), el valor de los *SP* aritméticos es inferior en el unilateral (el valor de estos parámetros no sólo es el tiempo de cómputo en CPU, sino también el tiempo de acceso a los datos con que se opera).

En la Tabla 4.1 se muestran los mejores tiempos obtenidos[*] con el algoritmo bilateral y con el unilateral, con un tamaño de problema escalado al número de procesadores utilizados. Se aprecia cómo el tiempo de ejecución del algoritmo unilateral es hasta un 20% inferior al del bilateral en la plataforma TORC cuando aumenta el número de procesadores. En COCI, debido al alto coste de las comunicaciones, se consigue una disminución más importante, llegando a ser de 50%. Finalmente, en Besiberri se consiguen las mayores reducciones del tiempo de ejecución cuando crece el número de procesadores y  el tamaño de problema, debido a la mayor escalabilidad de nuestro algoritmo unilateral frente al tradicional bilateral.

| | Número de procesadores | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 8 | 10 | 12 | 14 |
| **TORC** | | | | | | | | | |
| Bilateral | 0.63 | 1.45 | 2.81 | 4.59 | 6.97 | | | | |
| Unilateral | 0.64 | 1.49 | 2.75 | 3.92 | 5.80 | | | | |
| **COCI** | | | | | | | | | |
| Bilateral | 0.67 | 1.72 | 4.21 | 12.86 | | | | | |
| Unilateral | 0.48 | 1.27 | 3.50 | 6.86 | | | | | |
| **Besiberri** | | | | | | | | | |
| Bilateral | 0.12 | 0.20 | 0.39 | 0.24 | 1.03 | 4.35 | 12.70 | 11.04 | 27.38 |
| Unilateral | 0.08 | 0.16 | 0.35 | 0.12 | 0.48 | 1.81 | 3.69 | 5.55 | 7.13 |

Tabla 4.1. Comparación del tiempo de ejecución por barrido del algoritmo bilateral y el algoritmo unilateral, con un tamaño de problema escalado $n$=64$p$, en diferentes plataformas.

En la actualidad, varios grupos de investigación continúan estudiando las versiones unilaterales del método de Jacobi:

- Daoudi *et al.* [DLO03] están mejorando las versiones paralelas de estos algoritmos con el objetivo de aumentar el solapamiento de comunicaciones y cálculo.
- Malheiro *et al.* [Mal03] centran su trabajo en la búsqueda de criterios de parada apropiados (todos los métodos de Jacobi son iterativos) que permitan aumentar la precisión de estos algoritmos.

[*] Se utiliza en cada caso la mejor selección del tamaño de bloque y de la topología lógica de los procesadores. Según veremos en el capítulo siguitente, esta elección de parámetros de los algoritmos se puede realizar de manera automatica en base a su modelo analítico.

### 4.5.5 Problema de mínimos cuadrados de matrices Toeplitz

El problema de mínimos cuadrados se presenta en multitud de aplicaciones de alto nivel, tales como análisis de series temporales, procesamiento de imágenes, teoría de control, estadísticas, etc. En algunos casos la resolución cuenta con restricciones de respuesta en tiempo real como, por ejemplo, en aplicaciones de radar y sonar.

El problema de mínimos cuadrados de matrices de Toeplitz consiste en resolver:

$$\min_x \|Tx - b\|_2 \qquad 4.31$$

donde la matriz $T \in \mathbf{R}^{m\times n}$ es Toeplitz, $T_{ij} = t_{i-j} \in \mathbf{R}$, para $i = 0, ..., m-1$ y $j = 0, ..., n-1$, siendo $b \in \mathbf{R}^m$ un vector cualquiera.

El problema de mínimos cuadrados se puede resolver de manera genérica para cualquier tipo de matriz $T$ en $O(mn^2)$ flops, usando una descomposición QR, por ejemplo, la de LAPACK, que está basada en transformaciones de Householder. Sin embargo, si se aprovecha la estructura Toeplitz de la matriz $T$, hay varios algoritmos capaces de resolver este problema en $O(mn)$ flops. Todos estos algoritmos son generalizaciones del clásico algoritmo generalizado de Schur [Sch17]. La rutina secuencial cuyo comportamiento estudiaremos corresponde a la implementación de [ABV01] basada en el algoritmo propuesto en [PE00]. La versión paralela corresponde con el algoritmo propuesto e implementado en [ABV01].

#### Base teórica del algoritmo

Asumiendo que la matriz $R_0 \in \mathbf{R}^{(n+1)\times(n+1)}$ es la submatriz triangular superior del factor $R$ de la descomposición $QR$ de la matriz $[b\ T]$,

$$R_0 = qr([bT]) = \begin{pmatrix} k & w^T \\ 0 & G \end{pmatrix} \qquad 4.32$$

donde $k \in \mathbf{R}$, $w \in \mathbf{R}^{n+1}$, $G \in \mathbf{R}^{n\times n}$ es triangular superior y el operador $qr$ denota la submatriz triangular superior del factor $R$ de la descomposición QR de una matriz dada. Entonces el problema de mínimos cuadrados (4.31) puede resolverse a través de un producto de rotaciones de Givens $J$ que reduce una matriz Hessemberg a una de forma triangular superior,

$$J\begin{pmatrix} k & w^T \\ 0 & G \end{pmatrix} = \begin{pmatrix} R & r_1 \\ 0 & r_{nm} \end{pmatrix} \qquad 4.33$$

donde $R \in \mathbf{R}^{n\times n}$ es el factor triangular superior de la descomposición $QR$ de $T$. El vector solución $x$ (4.31) se obtiene resolviendo el sistema linear triangular:

$$Rx = r_1 \qquad 4.34$$

#### Algoritmo

El algoritmo de resolución estaría formado por cuatro pasos principales:

1. Construir el *par generador* de la matriz $[T\ b]$, necesario para aplicar el Algoritmo de Schur Generalizado.

2. Obtener el factor triangular $G$ que aparece en (4.32) mediante el Algoritmo de Schur Generalizado.
3. Refinar el factor $G$ de cara a mejorar su precisión.
4. Resolver 4.33 y entonces, como en el método estándar (vía descomposición $QR$), resolver el sistema triangular de 4.34.

### Implementación

En cuanto a las diferentes rutinas básicas que se utilizarían para la implementación de este algoritmo podríamos agruparlas en cinco apartados:

**TRSM**: formado por resoluciones de sistemas triangulares de ecuaciones con múltiples vectores como lado derecho. Concretamente, se resuelven tres sistemas triangulares de tamaño $n \times n$ mediante la rutina *DTRSM* del nivel 3 de BLAS. Debido a que el lado derecho consta de tres vectores, el coste es $O(n^2)$. En la versión paralela se usa la rutina *PDTRSM* del nivel 3 de PBLAS.

**TRSV**: formado por la resolución de un sistema triangular de ecuaciones, usando la rutina *DTRSV* del nivel 2 de BLAS. El coste es $O(n^2)$, pero el tiempo de ejecución es mucho menor que el de TRSM. En la versión paralela se usa la rutina *PDTRSV* del nivel 3 de PBLAS.

**LINEAR**: formado por una serie de rutinas del nivel 1 de BLAS: *DDOT*, para el producto de dos vectores; *DAXPY*, para multiplicar un vector por una constante y sumarle otro vector; y *DSCAL*, para multiplicar un vector por una constante. También se aplican otras rutinas de LAPACK: *DLARFX*, para aplicar un vector reflector a una matriz; y *DORG2R* que genera una matriz de columnas ortonormales como el producto de una serie de reflectores. El coste es lineal en $m$ o $n$.

**ROTATION**: formado por aplicaciones de varios tipos de rotaciones a una matriz, usando las rutinas *DLARFX* (de LAPACK) y *DROT* (del nivel 1 de BLAS). El coste es $O(n^2)$.

**GTOMV**: formado por una operación matriz-vector $y = \alpha Tx + \beta y$. Se utiliza una rutina del estilo *DAXPY* del nivel 1 de BLAS, pero programada a medida. Se usa un vector para guardar los valores significativos de la matriz Toeplitz.

Con lo que el coste total de la versión secuencial sería:

$$T_{EXEC} = 26nmk_{1\_AXPY} + 8n^2k_{1\_LARFX} + 3n^2k_{1\_otros} + \frac{21}{2}n^2k_{1\_ROT} + 9n^2k_{2\_TRSM} + n^2k_{2\_TRSV} \qquad 4.35$$

## 4.5.6 Método de “Elevación y Proyección”

El problema inverso aditivo de valores propios consiste en, dado el conjunto de $(n+1)$ matrices simétricas $\{A_0, A_1, ..., A_n\}$, con $A_i \in \mathbf{R}^{n \times n}$, y un conjunto de $n$ números reales $\lambda^*_1 \leq \lambda^*_2 \leq ... \leq \lambda^*_n$, buscar $d \in \mathbf{R}^n$, tal que los valores propios $\lambda_1(d) \leq \lambda_2(d) \leq ... \leq \lambda_n(d)$ de la matriz

$$A(d) = A_0 + \sum_{k=1}^{n} d_k A_k \qquad 4.36$$

satisfagan $\lambda_i(d) = \lambda^*_i$, para $i = 1, \ldots, n$. Este problema no siempre tiene solución. Por esta razón, se suele abordar planteando una solución de mínimos cuadrados.

### Planteamiento del método de "elevación y proyección"

Sea **Γ** el conjunto de todas las matrices que tienen $\lambda^*_1$, $\lambda^*_2$, ..., $\lambda^*_m$ como parte de sus espectro, es decir, todas las matrices de la forma:

$$A = Q\begin{bmatrix} \Lambda^*_m & \\ & \Lambda_\sigma \end{bmatrix} Q^T \tag{4.37}$$

con $Q$ ortogonal, donde $\Lambda^*_{\text{m}} = diag(\lambda^*_1, \lambda^*_2, ..., \lambda^*_m)$ y $\Lambda_\sigma \in \mathbf{R}^{(n-m)\times(n-m)}$ es una matriz diagonal.

Sea **A** el conjunto de matrices dadas por (4.36). La intersección de los conjuntos **A** y **Γ** representa el conjunto de soluciones del problema inverso aditivo de valores propios planteado. Si esta intersección es vacía, se puede encontrar una solución de mínimos cuadrados buscando el elemento de **A** que verifique que su distancia a **Γ** sea mínima. Por lo tanto, se puede plantear el problema como:

Buscar $d \in \mathbf{R}^L$, con $L>m$, $Q \in \mathbf{R}^n$ ortogonal y $\Lambda_\sigma \in \mathbf{R}^{(n-m)\times(n-m)}$ diagonal, tal que el valor de la función:

$$G(d,Q,\Lambda) = \frac{1}{2}\left\| A(d) - Qdiag(\Lambda^*_m, \Lambda_\sigma)Q^T \right\|_F^2 \tag{4.38}$$

sea mínimo.

Es significativo que el número de valores propios predescritos tiene cardinal $m$ que puede ser menor que $n$. Por esta razón, asociado a este problema principal, existe un problema de optimización discreta que consiste en encontrar el subconjunto de índices $\sigma = \{\sigma_1,...,\sigma_m\}$, con $1 \leq \sigma_1 < ... < \sigma_m \leq n$, que minimicen la función:

$$\min_{1\leq\sigma_1...\leq\sigma_m\leq n} \sum_{i=1}^{m} \left(\lambda_{\sigma_i}(d) - \lambda_i^*\right) \tag{4.39}$$

Esto es, buscar el más cercano subconjunto del espectro de $A(d)$ a los valores propios dados.

### Algoritmo de "elevación y proyección"

En este algoritmo de [Alb02] se utiliza la solución propuesta en [CC96] para resolver el problema planteado con la ecuación 4.38 mediante un método iterativo. Partiendo de un vector inicial $d^0$, cada iteración $k$ estaría organizada en dos fases:

- **Elevación**: Buscar un punto $Z^k \in \Gamma$ tal que $dist(A(d^k),Z^k) = dist(A(d^k), \Gamma)$. Este punto puede calcularse a partir de valores y vectores propios de $A(d^k)$, convenientemente ordenados, es decir, $Z^k = Q^*\ diag(\Lambda^*_{\text{m}}, \Lambda_\sigma)\ Q^{*T}$, siendo $A(d^k) = Q\Lambda Q^T$, con $\Lambda$ diagonal y $Q^*$ una versión permutada de la matriz ortogonal $Q$.

- **Proyección**: Buscar el punto $d^{k+1} \in \mathbf{R}^L$ tal que $dist(A(d^{k+1}),Z^k) = dist(Z^k,A(d^k))$. El punto $d^{k+1}$ se denomina proyección de $Z^k$ sobre **A**. Este punto se puede calcular resolviendo el sistema de ecuaciones lineales $Cd^{k+1} = b^k$, donde los elementos de $C$ son de la forma $C_{ij} = traza(A^T_i\, A_j)$ y los del vector $b^k$ son de la forma $b^k_i = traza((Z^k - A_0)\, A_j)$, con $i,j = 1, 2, ..., L$.

## Implementación

En cuanto a las diferentes rutinas básicas que se utilizan para la implementación de este algoritmo podríamos agrupar las más importantes en varios apartados donde se usarán determinadas rutinas de BLAS y LAPACK, mientras que en su versión paralela se utilizarían las rutinas de PBLAS y ScaLAPACK equivalentes.

**TRACE**: Inicialmente se calcula la matriz de proyección $C$, cuyos elementos son de la forma $C_{ij} = traza(A^T_i A_j)$. Este cálculo se realiza solamente una vez, al inicio de la ejecución. Cada traza se calcula realizando $n$ productos con la rutina *DOT* y sumando los resultados. El coste es:

$$2n^2L^2k_{1\_DOT} + nL^2k_{1\_SUM} \qquad 4.40$$

El resto del algoritmo consiste en una serie de iteraciones hasta alcanzar la convergencia, con lo que el resto de grupos de rutinas se repetirán en cada iteración.

**ADK:** La construcción de la matriz $A(d^k)$ en cada iteración $k$ se realiza mediante multiplicaciones escalar-matriz y sumas matriz-matriz. Estas operaciones se realizan mediante las rutinas de nivel 1 *SCAL* y *AXPY*. El coste por iteración será:

$$n^2Lk_{1\_SCAL} + n^2Lk_{1\_AXPY} \qquad 4.41$$

**EIGEN:** Los valores y vectores propios de $A(d)$ se obtienen en cada iteración usando la rutina *SYEV* de LAPACK. El coste por iteración será:

$$\frac{22}{3}n^3k_{SYEV} \qquad 4.42$$

**MATEIG:** Para la obtención de $Z^k = Q^* \, diag(\Lambda^*_m , \Lambda_\sigma) \, Q^{*T}$ se realiza una premultiplicación y una postmultiplicación. MATEIG corresponde con la premultiplicación y su coste será:

$$n^3k_{3\_DIAG\_GEMM} \qquad 4.43$$

donde $k_{3\_DIAG\_GEMM}$ representa el coste de una operación aritmética cuando se está utilizando una multiplicación matricial, pero siendo una de las matrices de forma diagonal. Este coste es claramente inferior al correspondiente a una multiplicación de matrices densas.

**MATMAT:** Es la multiplicación de matrices densas necesaria para resolver el problema combinatorio asociado que se indicó en 4.39. El coste por iteración es:

$$2n^3k_{3\_GEMM} \qquad 4.44$$

**ZKAOA:** La matriz $traza((Z^k–A_0)A_j)$ se obtiene para resolver el sistema $b^k_i=traza((Z^k–A_0)A_j)$. El coste de formar esta matriz en cada iteración es:

$$2n^2Lk_{1\_DOT} \qquad 4.45$$

mientras que el coste de la resolución del sistema es despreciable.

## 4.6 Resultados experimentales

De cara a validar nuestra propuesta de modelado, en esta sección se muestran diversas comparativas de los tiempos de ejecución de las rutinas modeladas, en las plataformas descritas en el capítulo anterior y con varias librerías básicas. Para cada rutina se compararán el tiempo teórico según el modelo propuesto DLAM+, el tiempo teórico según el modelo DLAM y el tiempo de ejecución que realmente se obtiene.

En la Tabla 4.2, en secuencial, y en la Tabla 4.3, para 16 procesadores, se muestra una comparativa para la rutina de factorización LU en la plataforma Karnak, usando como librería básica un BLAS específico para esta plataforma (BLAS-maq).

| | Tamaño de bloque | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 16 | | 32 | | 64 | | 128 | |
| Tamaño del problema | DL | DL+ | DL | DL+ | DL | DL+ | DL | DL+ |
| 256 | 70 % | 16 % | 19 % | 22 % | 6 % | 16 % | 15 % | 37 % |
| 512 | 66 % | 7 % | 15 % | 12 % | 26 % | 1 % | 6 % | 22 % |
| 768 | 65 % | 4 % | 28 % | 24 % | 51 % | 19 % | 37 % | 1 % |
| 1024 | 64 % | 2 % | 10 % | 12 % | 6 % | 16 % | 3 % | 28 % |
| 1280 | 65 % | 3 % | 5 % | 2 % | 23 % | 3 % | 55 % | 14 % |
| 1536 | 64 % | 2 % | 2 % | 4 % | 30 % | 3 % | 36 % | 0 % |
| 1792 | 65 % | 5 % | 19 % | 15 % | 45 % | 15 % | 57 % | 16 % |
| 2048 | 65 % | 4 % | 55 % | 56 % | 47 % | 58 % | 49 % | 62 % |
| 2304 | 65 % | 5 % | 18 % | 14 % | 46 % | 15 % | 58 % | 16 % |
| 2560 | 66 % | 5 % | 1 % | 3 % | 31 % | 4 % | 42 % | 5 % |
| 2816 | 65 % | 2 % | 15 % | 11 % | 43 % | 13 % | 57 % | 15 % |
| 3072 | 65 % | 3 % | 7 % | 10 % | 16 % | 9 % | 23 % | 10 % |

Tabla 4.2. Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico siguiendo el modelo DLAM (izquierda) y del tiempo teórico siguiendo el modelo DLAM+ (derecha). Para la rutina de factorización LU secuencial en la plataforma Karnak con BLAS-maq.

| | Tamaño de bloque | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 16 | | 32 | | 64 | | 128 | |
| Tamaño del problema | DL | DL+ | DL | DL+ | DL | DL+ | DL | DL+ |
| 1024 | 67 % | 20 % | 15 % | 11 % | 4 % | 15 % | 7 % | 36 % |
| 1536 | 21 % | 11 % | 1 % | 3 % | 13 % | 3 % | 13 % | 11 % |
| 2048 | 64 % | 15 % | 25 % | 21 % | 8 % | 21 % | 11 % | 13 % |
| 2560 | 39 % | 4 % | 14 % | 11 % | 9 % | 1 % | 23 % | 2 % |
| 3072 | 14 % | 22 % | 5 % | 8 % | 21 % | 11 % | 27 % | 7 % |
| 3584 | 56 % | 8 % | 4 % | 1 % | 21 % | 12 % | 26 % | 5 % |
| 4096 | 45 % | 0 % | 6 % | 3 % | 10 % | 0 % | 13 % | 11 % |

Tabla 4.3. Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico siguiendo el modelo DLAM (izquierda) y del tiempo teórico siguiendo el modelo DLAM+ (derecha). Para la rutina de factorización LU paralela, con 4×4 procesadores, en Karnak con BLAS-maq.

Los datos mostrados son las desviaciones respecto al tiempo experimental del tiempo teórico de la rutina modelado según la propuesta DLAM y del tiempo teórico según nuestra propuesta DLAM+. En este caso, la diferencia entre estos dos planteamiento consiste en tomar el valor del principal parámetro del sistema que aparece en el modelo analítico de la rutina ($k_{3\_DGEMM}$), bien como un valor constante (tomando la media de los valores obtenida experimentalmente, en DLAM), o bien como valores diferentes que dependen del tamaño de bloque con el que se opera (DLAM+, Tabla 4.4). Como se puede apreciar, la desviación del tiempo teórico se reduce notablemente, desde un 30% con DLAM a un 14% con DLAM+.

| | Tamaño de bloque | | | |
|---|---|---|---|---|
| Tamaño del problema | 16 | 32 | 64 | 128 |
| 256, …, 3072 | 0.0070 | 0.0030 | 0.0025 | 0.0025 |

Tabla 4.4. Valores medidos experimentalmente de $k_{3\_DGEMM}$ en Karnak con BLAS-maq (en microsegundos).

En la Tabla 4.5 podemos ver para la rutina de Jacobi unilateral para el problema de valores propios una comparativa en la misma plataforma y con diferentes topologías lógicas de los procesadores,. Con nuestra propuesta de modelo se consigue reducir la desviación media respecto al tiempo de ejecución real desde un 26% a un 9%.

| | Tamaño de bloque | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 16 | | 32 | | 64 | | 128 | |
| Topología procesadores | DL | DL+ | DL | DL+ | DL | DL+ | DL | DL+ |
| 1×8 | 33 % | 11 % | 13 % | 12 % | 7 % | 4 % | 56 % | 15 % |
| 2×4 | 28 % | 5 % | 17 % | 3 % | 9 % | 6 % | 44 % | 30 % |
| 4×2 | 33 % | 8 % | 12 % | 4 % | 15 % | 9 % | 46 % | 14 % |
| 8×1 | 21 % | 7 % | 15 % | 3 % | 9 % | 2 % | 65 % | 8 % |

Tabla 4.5. Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico siguiendo el modelo DLAM (izquierda) y del tiempo teórico siguiendo el modelo DLAM+ (derecha). Para la rutina de Jacobi para el problema de valores propios, con 8 procesadores y $n$=3072, en la plataforma Karnak con BLAS-maq.

Probando con otras rutinas y plataformas, tenemos en la Tabla 4.6 el tiempo medido para el parámetro $k_{3\_DGEMM}$ en la plataforma TORC con la librería ATLAS. El valor medido experimentalmente de $f_{broad}(p)$ es $\log_2(p)$, es decir, la red de interconexión de esta plataforma se ha mostrado con capacidad dinámica para implementar una operación de *broadcast* de manera eficiente. En esta ocasión, para la rutina secuencial de factorización QR, con nuestro planteamiento de modelo se reduce en promedio a la mitad la desviación con el tiempo de ejecución experimental (Tabla 4.7), pasando de un 12% a un 6%.

| | Tamaño de bloque | | | |
|---|---|---|---|---|
| Tamaño del problema | 16 | 32 | 64 | 128 |
| 256, …, 3072 | 0.0038 | 0.0033 | 0.0030 | 0.0030 |

Tabla 4.6. Valores medidos experimentalmente de $k_{3\_DGEMM}$ en TORC con ATLAS (en microsegundos).

| | Tamaño de bloque | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 16 | | 32 | | 64 | | 128 | |
| Tamaño del problema | DL | DL+ | DL | DL+ | DL | DL+ | DL | DL+ |
| 512 | 33 % | 23 % | 15 % | 15 % | 12 % | 16 % | 13 % | 9 % |
| 1024 | 31 % | 21 % | 9 % | 8 % | 6 % | 1 % | 9 % | 4 % |
| 1536 | 25 % | 13 % | 3 % | 3 % | 8 % | 0 % | 5 % | 2 % |
| 2048 | 21 % | 9 % | 2 % | 1 % | 7 % | 1 % | 8 % | 1 % |
| 2560 | 20 % | 7 % | 1 % | 1 % | 7 % | 1 % | 8 % | 0 % |
| 3072 | 19 % | 6 % | 3 % | 2 % | 7 % | 1 % | 8 % | 1 % |

Tabla 4.7. Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico siguiendo el modelo DLAM (izquierda) y del tiempo teórico siguiendo el modelo DLAM+ (derecha). Para la rutina de factorización QR secuencial en la plataforma TORC con ATLAS.

Finalmente, en la Tabla 4.8 tenemos los valores $k_{3\_DGEMM}$ en una SUN1 de la plataforma COCI con BLAS-maq como librería básica. En esta plataforma el valor medido experimentalmente de $f_{broad}(p)$ es $p$, es decir, la red de interconexión no tiene capacidad dinámica para implementar una operación de *broadcast* de manera eficiente, aumentando considerablemente el tiempo de ejecución de la rutina. Esto último, junto a la variabilidad de $k_{3\_DGEMM}$, trae como consecuencia, para la rutina paralela de factorización QR, que nuestro planteamiento reduzca la desviación respecto al tiempo de ejecución real desde un 31% a un 9% (Tabla 4.9).

| | Tamaño de bloque | | | |
|---|---|---|---|---|
| Tamaño del problema | 16 | 32 | 64 | 128 |
| 256, …, 3072 | 0.0120 | 0.0110 | 0.0110 | 0.0110 |

Tabla 4.8. Valores medidos experimentalmente de $k_{3\_DGEMM}$ en una SUN1 de COCI, usando BLAS-maq (en microsegundos).

| | Tamaño de bloque | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 16 | | 32 | | 64 | | 128 | |
| Tamaño del problema | DL | DL+ | DL | DL+ | DL | DL+ | DL | DL+ |
| 512 | 48 % | 20 % | 45 % | 17 % | 37 % | 6 % | 23 % | 14 % |
| 1024 | 36 % | 8 % | 37 % | 12 % | 30 % | 3 % | 23 % | 4 % |
| 1536 | 29 % | 4 % | 28 % | 5 % | 38 % | 20 % | 28 % | 8 % |
| 2048 | 26 % | 3 % | 26 % | 7 % | 25 % | 6 % | 22 % | 4 % |

Tabla 4.9. Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico siguiendo el modelo DLAM (izquierda) y del tiempo teórico siguiendo el modelo DLAM+ (derecha). Para la rutina de factorización QR paralela, en 2×2 nodos SUN1 de la plataforma COCI con BLAS-maq.

Se ha mostrado en esta sección cómo el modelo analítico de una rutina puede predecir su comportamiento gracias al acercamiento a la realidad que se introduce en la fórmula teórica cuando se usan los valores medidos experimentalmente de los *SP*. Estos parámetros toman valores diferentes dependiendo de la plataforma hardware, de las librerías básicas que se utilicen, del tamaño de problema a resolver y de algunas características del algoritmo, como, por ejemplo, el tamaño de bloque utilizado. Además, como veremos detenidamente en el capítulo siguiente, gracias a la mejor capacidad de predicción que nos ofrece nuestra propuesta DLAM+ frente al DLAM, se podrán tomar mejores decisiones acerca de qué valores asignar a los parámetros ajustables de las rutinas, previamente a su ejecución. Así, por ejemplo, para la rutina de Jacobi descrita, se puede observar en la Tabla 4.11 y en la Tabla 4.10 cómo se reducen notablemente las desviaciones con respecto al tiempo de ejecución óptimo (conocido únicamente a posteriori) del tiempo de ejecución obtenido utilizando DLAM+ como herramienta de decisión, frente a utilizar DLAM.

| Tamaño de problema | Tiempo Óptimo (conocido a posteriori) | Tiempo con parámetros DLAM | Tiempo con parámetros DLAM+ | Desviación con parámetros DLAM | Desviación con parámetros DLAM+ |
|---|---|---|---|---|---|
| 512 | 1.19 | 1.19 | 1.19 | 0 % | 0 % |
| 1024 | 9.48 | 9.48 | 9.48 | 0 % | 0 % |
| 1536 | 21.50 | 31.60 | 21.50 | 32 % | 0 % |
| 2048 | 83.17 | 94.60 | 83.17 | 12 % | 0 % |
| 2560 | 101.83 | 152.48 | 101.83 | 33 % | 0 % |
| 3072 | 229.73 | 262.66 | 237.01 | 13 % | 3 % |

Tabla 4.10. Comparación de los tiempos de ejecución obtenidos con los parámetros algorítmicos elegidos según el modelo DLAM y según el modelo DLAM+, con respecto al tiempo óptimo. Para la rutina de Jacobi para el problema de valores propios, con 4 procesadores, en la plataforma Karnak con BLAS-maq.

| Tamaño de problema | Tiempo Óptimo (conocido a posteriori) | Tiempo con parámetros DLAM | Tiempo con parámetros DLAM+ | Desviación con parámetros DLAM | Desviación con parámetros DLAM+ |
|---|---|---|---|---|---|
| 512 | 0.90 | 0.90 | 0.90 | 0 % | 0 % |
| 1024 | 6.37 | 7.91 | 6.74 | 19 % | 5 % |
| 1536 | 13.71 | 24.01 | 13.71 | 43 % | 0 % |
| 2048 | 44.25 | 53.34 | 47.96 | 17 % | 8 % |
| 2560 | 69.88 | 91.49 | 69.88 | 24 % | 0 % |
| 3072 | 116.85 | 163.49 | 116.85 | 29 % | 0 % |

Tabla 4.11. Comparación de los tiempos de ejecución obtenidos con los parámetros algorítmicos elegidos según el modelo DLAM y según el modelo DLAM+, con respecto al tiempo óptimo. Para la rutina de Jacobi para el problema de valores propios, con 8 procesadores, en la plataforma Karnak con BLAS-maq.

## 4.7 Resumen y conclusiones

En este capítulo se ha descrito una propuesta de modelo analítico teórico-experimental donde quede reflejado el tiempo de ejecución de software de álgebra lineal sobre una plataforma paralela genérica.

En primer lugar, se ha repasado una serie de antecedentes en el campo del modelado de computación paralela (PRAM, BSP, LogP, ...). A continuación, tomando como punto de partida los anteriores planteamientos históricos en este campo de investigación, se ha mostrado nuestra propuesta de modelado para un sistema paralelo de computación numérica, el modelo DLAM+. Este modelo está formado por un conjunto de parámetros que determinan cómo influyen las características del sistema (hardware y software básico instalado) en el comportamiento del software de álgebra lineal.

A continuación se ha mostrado nuestra propuesta de modelo analítico para el tiempo de ejecución de rutinas de álgebra lineal. Dada una rutina, su modelo analítico teórico se desarrolla mediante el estudio de complejidad de su algoritmo y, tras ello, se realizan un conjunto de experimentos con los que se miden los valores de los parámetros del sistema que aparecen en este modelo (estos parámetros del sistema corresponden al modelo DLAM+ de la arquitectura destino).

Finalmente, se han descrito las distintas rutinas para las que se ha realizado un modelo analítico de su tiempo de ejecución, mostrándose una serie de comparaciones entre el tiempo predicho por nuestro modelo frente al tiempo real de ejecución de cada rutina sobre plataformas de diferente naturaleza.

Como conclusión podemos decir que se ha demostrado como un estudio teórico de la complejidad del algoritmo de una rutina nos puede proporcionar un detallado modelo analítico del tiempo de ejecución. Ahora bien, si queremos dar el salto desde lo teórico a lo real y que este modelo sea una herramienta para predecir el comportamiento real de cada rutina debemos introducirle información empírica sobre las características del sistema donde se ejecuta. Esta información se introduce en el modelo gracias al conjunto de parámetros del sistema que aparecen en él, teniendo en cuenta que estos parámetros pueden tomar valores diferentes dependiendo de la plataforma hardware, de las librerías básicas que se utilicen, del tamaño del problema a resolver y de algunas características del propio algoritmo.

Como veremos en los siguientes capítulos, el modelo analítico propuesto será una pieza clave en la arquitectura del sistema software de ajuste de rutinas de álgebra lineal que se plantea en esta tesis, pues se utilizará su capacidad de predicción del comportamiento de las rutinas para decidir los valores de los parámetros ajustables de éstas, previamente a su ejecución.

# Capítulo 5. Propuesta de Optimización Automática de Rutinas de Álgebra Lineal

## 5.1 Introducción

En este capítulo se va a describir en detalle la arquitectura propuesta para una rutina de álgebra lineal (*LAR*, *Linear Algebra Routine*) con capacidad de ajustarse automáticamente a las condiciones de la plataforma de ejecución. En este planteamiento no es necesario modificar la *LAR* original, sino que ésta pasa a formar parte de una estructura software llamada *SOLAR* (*Self-Optimised Linear Algebra Routine*) que le confiere la capacidad de adaptarse a las condiciones del sistema.

Como se mostró en el capítulo de trabajos relacionados, el proceso de ajustar la rutina a las condiciones del sistema se puede realizar en tres momentos diferentes: durante su instalación, cuando ésta se va a ejecutar, o bien a lo largo de su ejecución. En la primera opción la labor de ajuste de la rutina se basa en una visión estática de la plataforma de ejecución, no recogiéndose información útil del momento en que la rutina se ejecuta. Por el contrario, un ajuste en el momento de la ejecución de la rutina permitirá adaptarla a cualquier cambio que se produjera en la carga de trabajo del sistema, aunque, por contra, tiene el inconveniente de que puede introducir una sobrecarga no deseada, debida al tiempo de testear el sistema y al tiempo de ajuste de la rutina, en el tiempo de ejecución total.

En este capítulo presentaremos, en primer lugar, una propuesta inicial de *SOLAR* que realiza la totalidad del proceso de ajuste durante la instalación. Esta primera versión proporciona buenos resultados de ajuste de las rutinas en plataformas donde las condiciones de carga de trabajo no sufran grandes variaciones.

A continuación, se describirá una segunda propuesta de *SOLAR* que realiza una labor mixta. En este caso, la parte más costosa del proceso, donde se capturan las condiciones estáticas de la plataforma (capacidad de cómputo de los procesadores y capacidad de comunicación de la red de interconexión) se realiza durante la instalación. Cuando se va a ejecutar la rutina se lleva a cabo un proceso de refinamiento de la información obtenida durante la instalación, en base a la carga de trabajo que en ese momento esté soportando cada parte de la plataforma. Esta segunda versión será válida en cualquier tipo de plataformas, incluidas aquellas donde la carga de trabajo sufre grandes variaciones e, incluso, de manera heterogénea.

Para ambas propuestas, se mostrará la arquitectura software propuesta, se describirá cómo sería el ciclo de vida (diseño, instalación y ejecución) de una rutina y, finalmente, se mostrarán resultados experimentales obtenidos con diversas rutinas sobre plataformas de diferentes características.

## 5.2 Plataformas con carga de trabajo constante

En esta sección se describirá nuestra primera propuesta de una rutina de álgebra lineal con capacidad de optimización automática, *SOLAR* (*Self-Optimised Linear Algebra Routine*). En esta primera versión de *SOLAR* todo el proceso de ajuste de la rutina se realiza durante su instalación. Por esta razón, la rutina tendrá capacidad de adaptarse únicamente a las condiciones estáticas de la plataforma donde se instala (capacidad de cómputo de los procesadores y capacidad de comunicación de la red de interconexión). Se consiguen buenas prestaciones si las condiciones de carga de trabajo no sufren grandes variaciones, bien porque la plataforma cuenta con algún sistema propio de control de la carga de trabajo (colas de procesos, *scheduler*, ...), o bien porque la plataforma tenga un uso restringido y pueda utilizarse fácilmente en modo exclusivo.

En primer lugar describiremos la arquitectura software que configura esta propuesta. A continuación, se describirá su ciclo de vida (diseño, instalación y ejecución). Seguidamente, a modo de ejemplo, se mostrará el ciclo de vida de una rutina concreta, la rutina de factorización QR, tanto en su versión secuencial como paralela. Para finalizar esta sección se mostrará la funcionalidad de esta propuesta mediante un estudio experimental, mostrando los ciclos de vida de varias rutinas.

### 5.2.1 Arquitectura de una SOLAR

En la Figura 5.1 podemos ver un esquema de la arquitectura de una *SOLAR*. En esta estructura encontramos dos componentes de código (*LAR* y *SOLAR_manager*) y una serie de componentes de datos, agrupados según la etapa en la vida de una *SOLAR* en que se instancian con información útil. A continuación se describirán las principales características de cada uno de ellos

***LAR***

Es la rutina de álgebra lineal original, sin ninguna modificación. Entre sus parámetros de entrada podemos distinguir, de manera general:

- ***OPR:*** Es el conjunto de operandos de entrada/salida que describen el problema a resolver por la rutina. Normalmente, estará constituido por un conjunto de escalares y apuntadores a matrices y vectores.
- ***n***: Uno o varios parámetros que definen el tamaño del problema a resolver, es decir, las dimensiones de las matrices y vectores que se operarán.
- ***AP:*** Un conjunto de parámetros del algoritmo, $\{ap_1,...,ap_x\}$, que se pueden ajustar para mejorar las prestaciones de la rutina, sin que se vea afectada la corrección del resultado que se produce. Algunos de estos parámetros pueden ser: el tamaño de bloque de cómputo, $b$, en algoritmos por bloques; el número de procesadores a utilizar de entre todos los disponibles, $p$; algunos parámetros que definan la topología lógica de los $p$ procesadores, etc.

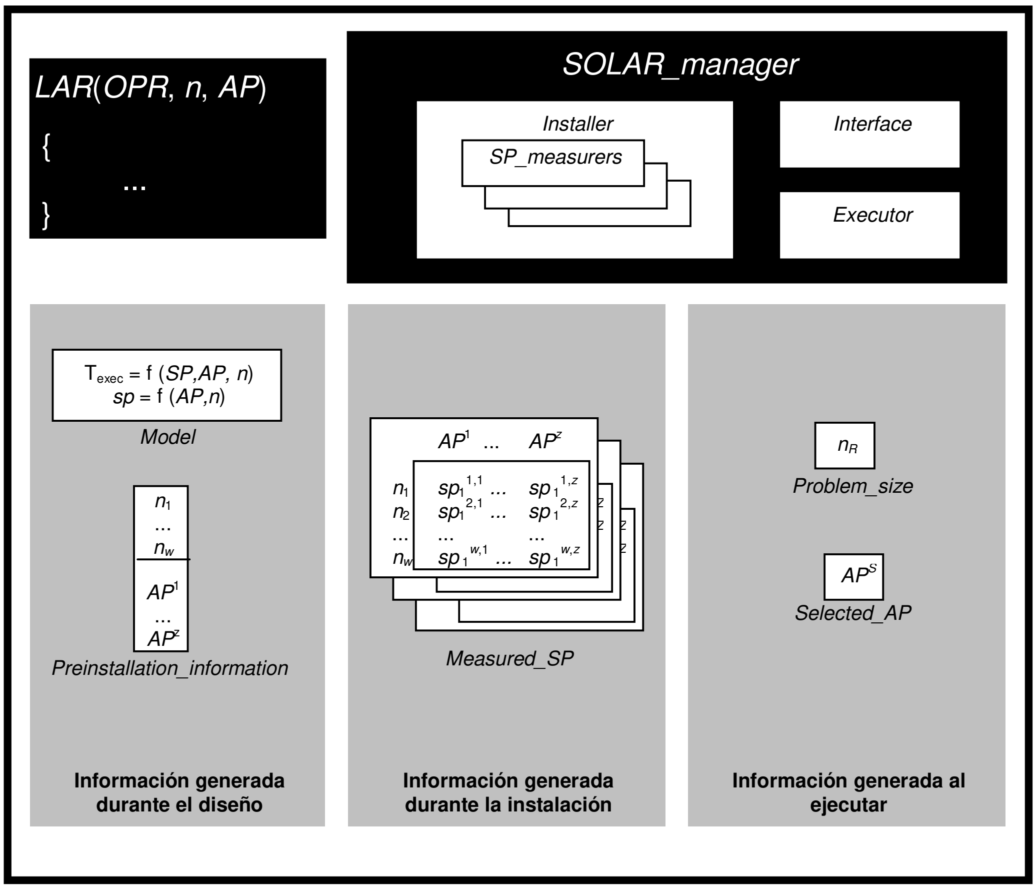


Figura 5.1. Arquitectura de una *SOLAR* para plataformas con carga de trabajo constante.

## *Model*

Es el modelo analítico de la *LAR* tal cual como se describió en el capítulo anterior. En este modelo queda reflejado su tiempo de ejecución de esta manera:

$$T_{EXEC} = f(n, SP, AP) \qquad 5.1$$

donde tendríamos:

- ***n***: tamaño de problema a resolver.
- ***AP***: el conjunto de parámetros algorítmicos, $\{ap_1,...,ap_x\}$, anteriormente descritos.
- ***SP***: el conjunto de parámetros $\{sp_1,...,sp_y\}$ que caracterizan el sistema donde la rutina se instala. Podemos distinguir dos subconjuntos de *SP*:
  - Los *SP* computacionales (*Comp_SP*). Son los $k_{i\_operacion}$, para $i$=1,2,3 y para cada posible operación de nivel *i* de BLAS. Estos parámetros $k_{i\_operacion}$ caracterizan la influencia del subsistema de cómputo (la unión de la plataforma hardware y de las librerías de álgebra linear instaladas) en el tiempo de ejecución de la rutina.
  - Los *SP* de comunicaciones (*Comm_SP*). Son la latencia o tiempo de inicio de una comunicación, $t_s$ (*startup time*), el tiempo de envío de una palabra, $t_w$ (*word-sending time*) y el parámetro *f*, que, dada una operación de comunicación colectiva, va a caracterizar las prestaciones que ofrece la red para llevar a cabo esa operación. Estos parámetros caracterizan la influencia del subsistema de comunicaciones (la unión de la plataforma hardware y de las librerías de comunicaciones instaladas) en el tiempo de ejecución de la rutina.

### *SOLAR_manager*

Es el motor software que gestiona toda la estructura de información de la *SOLAR*, siendo además su interfaz con el exterior. Los módulos principales que forman este componente son:

- ***Interface***: Módulo que reconoce las órdenes de instalación y de ejecución de la rutina, poniendo en marcha el proceso correspondiente.
- ***Installer*:** Módulo formado por un conjunto de subrutinas de medición de los *SP*. Por cada *sp* que aparezca en *Model*, su correspondiente rutina de medición (*sp_measurer*) consiste en el núcleo computacional o de comunicaciones que ese *sp* representa.
- ***Executor***: Módulo encargado de la elección de los valores más apropiados para los *AP* y la puesta en ejecución de la *LAR* con estos valores.

Como se verá en el apartado siguiente, tras su creación, el *SOLAR_Manager* intervendrá de manera directa en los distintos procesos que se llevan a cabo sobre la *SOLAR* a lo largo de todo su ciclo de vida.

### *Preinstallation_information*

Como ya se comentó en el capítulo anterior, los valores de los *SP* no se consideran constantes, pues pueden depender del tamaño del problema y de los valores de los *AP* que se tomen a la hora de ejecutar la rutina:

$$SP = f(n, AP) \qquad 5.2$$

En el componente *Preinstallation_information* quedan recogidos los valores seleccionados de *n* y de los *AP* que se utilizarán durante el proceso de instalación de la rutina para la medición de cada *sp*.

Formalmente, si en *Model* aparecen $x$ parámetros algorítmicos $\{ap_1,...,ap_x\}$ y $d_i$ es la cardinalidad del conjunto de valores seleccionados para cada $ap_i$, entonces $z = (d_1 d_2 .... d_x)$ será el número de posibles combinaciones de valores para el conjunto de *AP*, siendo $AP^i$ cada una de estas combinaciones. Por otro lado, sea $w$ la cardinalidad del conjunto de valores escogidos para el tamaño del problema. De esta manera, el componente *Preinstallation_information* estará formado por los conjuntos $\{n_1,n_2,...,n_w\}$ y $\{AP^1,AP^2,...,AP^z\}$.

### *Measured_SP*

Durante la instalación de la rutina, en cada *Measured_sp*$_i$ se almacenan los valores de $sp_i$ que se miden experimentalmente para cada combinación de $\{n_1,n_2,...,n_w\}\times\{AP^1,AP^2,...,AP^z\}$ recogidas en *Preinstallation_information*. Este proceso se describirá detalladamente en el siguiente apartado.

### *Problem_size*

A la hora de ejecutar la rutina, en este componente se almacenará el tamaño del problema a resolver, $n_R$.

### *Selected_AP*

Cuando se invoca a la rutina para resolver un problema de tamaño $n_R$ el *SOLAR_manager* obtiene y almacena en este componente la combinación de valores de los *AP* que, según la información que maneja sobre las características de la rutina y del sistema, minimizarán el tiempo de ejecución. Este proceso también se describirá detalladamente en el siguiente apartado.

## 5.2.2 Ciclo de vida de una *SOLAR*

El ciclo de vida de una *SOLAR* (Figura 5.2) comienza con un fase de diseño cuando se implementa la rutina de álgebra lineal, *LAR* (de igual manera, también se puede partir de una ya implementada con anterioridad), se realiza el modelo analítico de ésta y se implementa su gestor. A partir de este momento la rutina se podrá instalar en diferentes plataformas paralelas, realizándose en cada una de éstas un proceso de medición de los parámetros del sistema. Finalmente, cuando se invoca una *SOLAR* para su ejecución, se seleccionarán los valores óptimos de sus parámetros algorítmicos, según la información que se maneja, y, finalmente, se ejecutará la *LAR*.

### Diseño

Este proceso se realizará una única vez, siendo normalmente el diseñador de la rutina de álgebra lineal el encargado de llevarlo a cabo. Abarca las siguientes tareas:

**Creación de la rutina**: Se implementa la rutina de álgebra lineal sin necesidad de tener en cuenta todo el proceso de ajuste automático en el que se va a ver inmersa. En el caso de que la rutina ya esté implementada, como ya se comentó anteriormente, no es necesario modificación alguna de esta rutina, sino que se incluiría, tal cual, dentro de la arquitectura de autoadaptación que planteamos.

**Modelado de la rutina**: Se estudia el algoritmo de la *LAR* y su complejidad, obteniéndose el modelo analítico de su tiempo de ejecución, *Model*. En el capítulo anterior, donde se describió nuestra propuesta de modelado, se mostraron ejemplos de modelos para diferentes rutinas (factorización LU, factorización QR, ...).

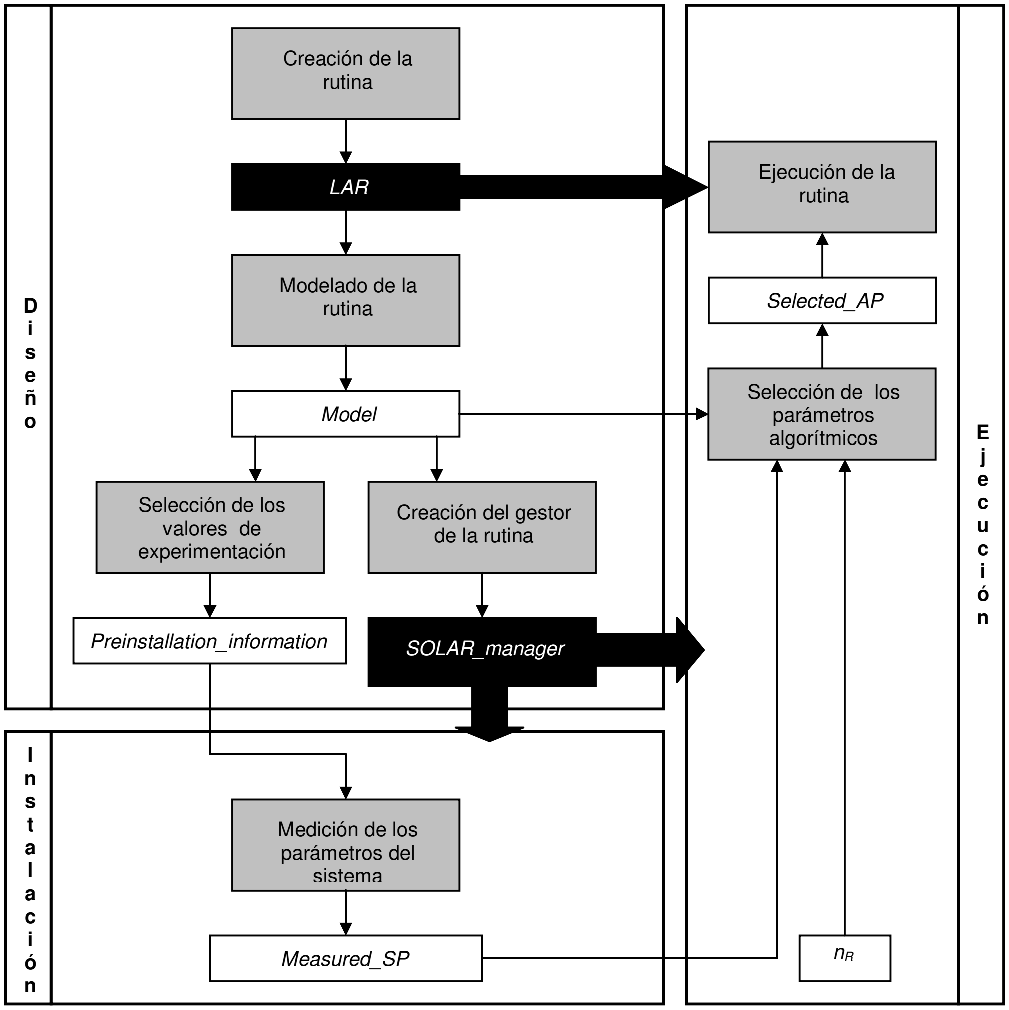


Figura 5.2. Ciclo de vida de una *SOLAR*

**Preselección de valores de experimentación**: Se debe seleccionar un conjunto significativo de valores para $n$ y para los diferentes *AP* que aparezcan en *Model*. Estos valores se almacenarán en el componente *Preinstallation_information*. El espacio multidimensional formado por las distintas combinaciones de valores de $n$, $\{n_1,...,n_w\}$, y de los *AP*, $\{AP^1,...,AP^z\}$, se utilizará durante la instalación de la rutina como marco de experimentación donde realizar mediciones de los diferentes *SP*. En otras aproximaciones, como en el proyecto de Brewer [Bre95], se proporciona un rango de valores sobre los que experimentar en lugar de un conjunto discreto, como proponemos nosotros. Nuestra decisión está motivada porque si se maneja un rango de valores gran parte de las combinaciones de éstos no serían válidas, lo que conduciría a realizar experimentos sin sentido.

Este proceso es tan solo una primera selección del conjunto de valores de experimentación. Justo al iniciar la instalación de la rutina el administrador del sistema puede modificar este conjunto en base a sus conocimientos de la plataforma. Además se podría añadir la posibilidad de que se utilice la información histórica que se va generando con las sucesivas ejecuciones de la rutina para modificar automáticamente este conjunto, con vistas a futuras reinstalaciones. Por ejemplo, si se invoca la rutina en repetidas ocasiones para resolver un problema de tamaño *n'*, no estando este valor entre los que pertenecen a *Preinstallation_information,* se podría incluir dicho valor en este componente, de cara a tenerlo en cuenta en la próxima reinstalación de la rutina.

**Creación del gestor de la rutina**: En base a las características de la *LAR* y de los *SP* que aparecen en su modelo analítico, se implementa el *SOLAR_manager* con la estructura descrita en el apartado anterior.

### Instalación

Cuando el administrador de un sistema quiere instalar una *SOLAR* le envía la orden de instalación. El módulo *Interface* del *SOLAR*_manager recoge esta orden y, tras ello, se ponen en marcha las siguientes tareas:

**Postselección de los valores de experimentación**: El *Interface* muestra al administrador del sistema los valores almacenados en *Preinstallation_information*, es decir, el marco de experimentación por defecto donde se realizarán las mediciones de los diferentes *SP*. En este momento el administrador del sistema puede decidir cualquier modificación de estos valores de acuerdo a sus conocimientos del sistema donde se está realizando la instalación.

**Medición de los parámetros del sistema:** El *Installer* pone a funcionar a los diferentes *SP_measurers*. Cada $sp_i$*_measurer* obtiene experimentalmente el valor de $sp_i$ para cada combinación de valores de tamaños de problema $\{n_1,...,n_w\}$ y parámetros algorítmicos $\{AP^1,...,AP^z\}$ almacenados en *Preinstallation_information*. Con estas mediciones obtenidas se va rellenando cada componente *Measured_*$sp_i$.

### Ejecución

Cuando un usuario quiere ejecutar una *SOLAR* únicamente tiene que llamar a la rutina pasándole los operandos, *OPR*, y el tamaño del problema a resolver, $n_R$. Esta orden de ejecución se recoge por el módulo *Interface* del *SOLAR_manager*, poniéndose en marcha las siguientes tareas:

**Selección de los parámetros algorítmicos:** El *Executor* realiza una búsqueda exhaustiva de los valores de los *AP* que, junto a los correspondientes valores de los *SP* y con $n=n_R$, minimizarán el tiempo de ejecución de la rutina según *Model*. Este proceso consiste en calcular el tiempo teórico de ejecución de la rutina para cada combinación de los parámetros algorítmicos $AP^j$, con el tamaño de problema fijado por el usuario, $n_R$, junto a los valores de los *SP* almacenados en *Measured_SP* para la combinación $(n_R, AP^j)$*. Aquella combinación de valores de los *AP* que proporcione el menor tiempo teórico de ejecución se almacenará en *Selected_AP*.

**Ejecución de la rutina:** El *Executor* pone a funcionar la rutina con los parámetros *OPR* y $n_R$ que el usuario le proporcionó, junto con los valores *Selected_AP* como valores elegidos para los parámetros ajustables del algoritmo.

## 5.2.3 Ciclo de vida de la rutina secuencial de factorización QR

En este apartado, se hará el seguimiento, a modo de ejemplo, del ciclo de vida de la rutina secuencial de factorización *QR* por bloques de la librería secuencial LAPACK, *DGEQRF*. La

* Si en *Measured_SP* no hubiese información sobre el valor que toman los diferentes *SP* para el valor concreto $n_R$, se tomarán los valores interpolados de estos *SP* para los dos tamaños de problema más próximos a $n_R$ de los que sí se tenga información en *Measured_SP*

descripción detallada de esta rutina, así como del modelo analítico de su tiempo de ejecución se ha mostrado en el capítulo anterior.

## Diseño

**Creación de la rutina**: En este caso partimos de una rutina ya implementada dentro de la librería LAPACK.

**Modelado de la rutina:** Al partir de la rutina ya diseñada e implementada, el primer proceso a realizar en la fase de diseño sería el estudio de su algoritmo para obtener la fórmula teórica del tiempo de ejecución. Como ya se mostró en el capítulo anterior, el modelo analítico de esta rutina, que quedaría almacenado en *Model*, sería:

$$T_{EXEC} = \frac{4}{3}n^3 k_{3\_DGEMM} + \frac{1}{2}n^2 b k_{3\_DTRMM} + n^2 b k_{2\_DGEQR2} + \frac{1}{2}n^2 b k_{2\_DLARFT} \qquad 5.3$$

**Selección de los valores de experimentación:** Como valores significativos para el tamaño de problema, *n*, seleccionaremos los valores {512, 1024, 1536, 2048, 2560, 3072, 3584, 4096}. Para el tamaño de bloque, *b*, que es el único *ap* que aparece en *Model*, tomaremos los valores {16, 32, 64, 128}.

**Creación del gestor de la rutina:** Dentro del *SOLAR_manager*, el módulo *Instalator* del *SOLAR_manager* tendría que estar formado por cuatro *SP_measurers*, uno para cada uno de los *SP* que encontramos en *Model*. Con lo que tendríamos las subrutinas: $k_{3\_DGEMM}$*_measurer*, $k_{3\_DTRMM}$*_measurer*, $k_{2\_DGEQR2}$*_measurer* y $k_{2\_DLARFT}$*_measurer*.

Cada *sp_measurer* estaría preparado para realizar medidas de tiempo con la rutina básica correspondiente (*DGEMM*, *DGEQR2*, *DTRMM* o *DLARFT*) teniendo en cuenta las posibles combinaciones de valores de *n* y *b* recogidos en *Preinstallation_information*. Así, por ejemplo, para un tamaño de problema *n* y un tamaño de bloque *b*, $k_{3\_DTRMM}$*_measurer* estaría preparada para medir el valor del parámetro de sistema $k_{3\_DTRMM}$ mediante llamadas a la rutina *DTRMM* que multiplica dos matrices (una de ellas triangular) conforme aparecen en la rutina *DGETRF*, es decir, con dimensiones $b \times b$ por $b \times c_i$, siendo $c_i = n-b, n-2b, ..., b$. De manera similar se construirían los demás *SP_measures.*

## Instalación

Vamos a realizar el proceso de instalación en procesadores individuales de las plataformas descritas en el capítulo referente al entorno de trabajo. Estos procesadores son: una SUN Ultra 1 (SUN1) de la plataforma COCI, una SUN Ultra 5 (SUN5) de la plataforma COCI, un IBM Power-PC (PPC) de la plataforma Besiberri, un SGI R10000 (R10K) de la plataforma Karnak y un Pentium III (PIII) de la plataforma TORC. En las dos estaciones de trabajo SUN las librerías básicas consideradas son: un BLAS de referencia (BLAS-ref), un BLAS específico de la máquina (BLAS-maq) y ATLAS. En PPC y en R10K, la librería utilizada es una BLAS-maq, mientras que en PIII la librería utilizada es ATLAS.

**Postselección de valores de experimentación**: A la hora de instalar la rutina podemos decidir mantener los valores preseleccionados en *Preinstallation_information* y continuar con el siguiente proceso.

**Medición de los parámetros del sistema**: Se ponen a funcionar los cuatro *SP_measurers* de esta *SOLAR* obteniéndose para cada plataforma los valores mostrados en la Tabla 5.1 ($k_{3\_DTRMM}$ y $k_{3\_DGEMM}$) y en la Tabla 5.2 ($k_{2\_DGEQR2}$ y $k_{2\_DLARFT}$). Esto valores se almacenan en *Measured_SP*.

| Sistema | | *n* | Tamaño de bloque, *b* | | | |
|---|---|---|---|---|---|---|
| | | | 16 | 32 | 64 | 128 |
| SUN1 | BLAS-ref | 512,...,4096 | **0.0200** | **0.0200** | 0.0220 | 0.0280 |
| | BLAS-maq | 512,...,4096 | 0.0120 | **0.0110** | **0.0110** | **0.0110** |
| | ATLAS | 512,...,4096 | 0.0070 | **0.0060** | **0.0060** | **0.0060** |
| SUN5 | BLAS-ref | 512,...,4096 | **0.0120** | 0.0130 | 0.0140 | 0.0150 |
| | BLAS-maq | 512,...,4096 | 0.0060 | **0.0050** | **0.0050** | **0.0050** |
| | ATLAS | 512,...,4096 | 0.0040 | 0.0032 | **0.0025** | **0.0025** |
| PIII | ATLAS | 512,...,4096 | 0.0038 | 0.0033 | **0.0030** | **0.0030** |
| PPC | BLAS-maq | 512,...,4096 | 0.0023 | 0.0019 | **0.0018** | **0.0018** |
| R10K | BLAS-maq | 512,...,4096 | 0.0070 | 0.0030 | **0.0025** | **0.0025** |

Tabla 5.1. Valores of $k_3$ ($k_{3\_DTRMM}$ y $k_{3\_DGEMM}$) obtenidos en el proceso de instalación de la rutina *DGEQRF*, en diferentes plataformas (en microsegundos).

| Sistema | | *n* | Tamaño de bloque, *b* |
|---|---|---|---|
| | | | 16, 32, 64, 128 |
| SUN1 | BLAS-ref | 512,...,4096 | 0.0200 |
| | BLAS-maq | 512,...,4096 | 0.0500 |
| | ATLAS | 512,...,4096 | 0.0700 |
| SUN5 | BLAS-ref | 512,...,4096 | 0.0050 |
| | BLAS-maq | 512,...,4096 | 0.0300 |
| | ATLAS | 512,...,4096 | 0.0500 |
| PIII | ATLAS | 512,...,4096 | 0.0150 |
| PPC | BLAS-maq | 512,...,4096 | 0.0100 |
| R10K | BLAS-maq | 512,...,4096 | 0.0250 |

Tabla 5.2. Valores of $k_2$ ($k_{2\_DGEQR2}$ y $k_{2\_DLARFT}$) obtenidos en el proceso de instalación de la rutina *DGEQRF*, en diferentes plataformas (en microsegundos).

Podemos observar que:

- El valor de $k_3$ ($k_{3\_DTRMM}$ y $k_{3\_DGEMM}$ ) depende del *b*, pero no de *n*, en cualquiera de las plataformas.
- El valor de *b* que proporciona un valor óptimo (mínimo) para $k_3$ es diferente en cada plataforma. De ahí la importancia de un estudio experimental como el planteado de cara a una posterior elección de valores para los *AP*, como en este caso es *b*.

## Ejecución

Cuando el usuario decida ejecutar la rutina para resolver un problema de tamaño $n_R$ se pondría en marcha este proceso. El módulo *Interface* del *SOLAR_manager* recoge la orden de ejecución, poniendo en marcha los procesos descritos a continuación.

**Selección de los parámetros algorítmicos**: Tomando los valores de los *Measured_SP* calculados durante la instalación (Tabla 5.1 y Tabla 5.2) y el valor $n_R$, se busca el valor de $b$ que minimiza la fórmula de *Model* (5.3). Ese valor se almacenaría en *Selected_AP*. En la Tabla 5.3 podemos ver los valores del tamaño de bloque seleccionados para diferentes tamaños del problema a resolver. Como podemos observar, el tamaño de bloque que proporciona un tiempo de ejecución teóricamente mínimo no es un valor constante que se pueda poner por defecto en el código de la rutina. Este valor depende del sistema en el que nos encontremos y del tamaño de problema.

| Sistema | | Tamaño del problema, *n* | | | | |
|---|---|---|---|---|---|---|
| | | 512 | 1024 | 1536 | 2048 | 2560 |
| SUN1 | BLAS-ref | 16 | 16 | 16 | 16 | 16 |
| | BLAS-maq | 16 | 32 | 32 | 32 | 32 |
| | ATLAS | 16 | 16 | 32 | 32 | 32 |
| SUN5 | BLAS-ref | 16 | 16 | 16 | 16 | 16 |
| | BLAS-maq | 16 | 32 | 32 | 32 | 32 |
| | ATLAS | 16 | 16 | 32 | 32 | 64 |
| PIII | ATLAS | 32 | 32 | 32 | 64 | 64 |
| PPC | BLAS-maq | 32 | 32 | 32 | 32 | 32 |
| R10K | BLAS-maq | 32 | 32 | 32 | 64 | 64 |

Tabla 5.3. Valores de *Selected_b* obtenidos justo antes de ejecutar la rutina *DGEQRF*, para diferentes tamaños de problema y en diferentes plataformas.

**Ejecución de la rutina:** El *SOLAR_manager* pone a funcionar la rutina utilizando los valores *Selected_AP* como parámetros ajustables del algoritmo.

De cara a comprobar la validez de nuestra propuesta, se han realizado ejecuciones para diferentes tamaños de problema, utilizando para cada uno todos los tamaños de bloque entre los que, dependiendo de la plataforma y del tamaño de problema, se puede encontrar el óptimo {16, 32, 64, 128}. En la Tabla 5.4 se muestra una comparación para cada tamaño de problema, entre:

- El tiempo óptimo, tomando el mínimo obtenido a posteriori, tras llevar cabo todas estas ejecuciones de prueba, nos indica la cota inferior del tiempo que podríamos haber alcanzado con la óptima decisión de valores para el tamaño de bloque.
- El tiempo promedio, tomando la media de tiempos para los valores diferentes de $b$ con los que se ha ejecutado la rutina. Este tiempo promedio representa el que necesitaría la rutina ejecutada por un usuario que escoge un tamaño de bloque al azar, que es lo único que puede hacer si no cuenta con las informaciones teórico-experimentales de la *SOLAR*.
- El tiempo *SOLAR*: el que corresponde con el que se obtendría utilizando el tamaño de bloque que la *SOLAR* elige automáticamente en base a la información que maneja.

Como podemos observar, en la mayoría de los casos el tamaño de bloque que la *SOLAR* escoge habría proporcionado el tiempo óptimo (esto se observa en aquellos casos donde la desviación es del 0%, es decir, donde la *SOLAR* ha escogido los valores para los *AP* que a posteriori sabemos que son los óptimos). En el resto de los casos, aunque no se realiza la óptima elección para $b$, se consigue normalmente un resultado mucho mejor que si se deja esta elección al azar, reduciendo la desviación media que obtendría el usuario desde un 42% a tan solo un 4%.

| Sistema | | 512 Pro | 512 SOL | 1024 Pro | 1024 SOL | 1536 Pro | 1536 SOL | 2048 Pro | 2048 SOL | 2560 Pro | 2560 SOL |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SUN1 | BLAS-ref | 37 % | 2 % | 45 % | 3 % | 47 % | 0 % | 45 % | 0 % | 5 % | 0 % |
| | BLAS-maq | 246 % | 0 % | 19 % | 0 % | 13 % | 0 % | 11 % | 0 % | 9 % | 0 % |
| | ATLAS | 320 % | 0 % | 14 % | 0 % | 15 % | 0 % | 15 % | 0 % | 12 % | 0 % |
| SUN5 | BLAS-ref | 32 % | 0 % | 32 % | 0 % | 13 % | 4 % | 10 % | 0 % | 22 % | 0 % |
| | BLAS-maq | 240 % | 0 % | 17 % | 0 % | 10 % | 0 % | 24 % | 6 % | 10 % | 0 % |
| | ATLAS | 305 % | 0 % | 18 % | 16 % | 10 % | 1 % | 10 % | 0 % | 10 % | 0 % |
| PIII | ATLAS | 14 % | 3 % | 12 % | 2 % | 12 % | 0 % | 9 % | 0 % | 9 % | 0 % |
| PPC | BLAS-maq | 29 % | 19 % | 10 % | 0 % | 8 % | 0 % | 26 % | 9 % | 3 % | 1 % |
| R10K | BLAS-maq | 63 % | 80 % | 13 % | 0 % | 18 % | 34 % | 23 % | 0 % | 18 % | 0 % |

Tabla 5.4. Comparación de la desviación del tiempo promedio (izquierda) y del tiempo *SOLAR* (derecha) con respecto al tiempo óptimo conocido a posteriori, para la rutina *DGEQRF* con diferentes tamaños de problema y en diferentes plataformas (en microsegundos).

Viendo la información mostrada en la Tabla 5.4, se podría pensar que otra posibilidad para un usuario experto, que pudiera contar con toda esa información, sería fijar $b$=32 para todos los casos (por ser el tamaño de bloque que más aparece como valor óptimo). En tal caso, para este usuario experto, la desviación con respecto al óptimo se reduciría hasta un promedio de un 13 % (Tabla 5.5), lo que demuestra la importancia de manejar esta información calculada experimentalmente. Sin embargo, incluso en esta situación, la *SOLAR* proporciona un mejor rendimiento (desviación de un 4%) gracias a su capacidad de adaptarse a cada caso concreto de ejecución. De todas maneras, la utilidad de la *SOLAR*, como veremos más adelante, se manifestará mejor en plataformas paralelas donde aparecen un mayor número de *SP* que hay que medir, así como un aumento en el número de *AP* cuyo valor hay que decidir.

| Sistema | | 512 $b$=32 | 512 SOL | 1024 $b$=32 | 1024 SOL | 1536 $b$=32 | 1536 SOL | 2048 $b$=32 | 2048 SOL | 2560 $b$=32 | 2560 SOL |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SUN1 | BLAS-ref | 0 % | 2 % | 0 % | 3 % | 14 % | 0 % | 32 % | 0 % | 11 % | 0 % |
| | BLAS-maq | 44 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % |
| | ATLAS | 56 % | 0 % | 2 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % |
| SUN5 | BLAS-ref | 25 % | 0 % | 26 % | 0 % | 0 % | 4 % | 1 % | 0 % | 12 % | 0 % |
| | BLAS-maq | 65 % | 0 % | 0 % | 0 % | 0 % | 0 % | 6 % | 6 % | 0 % | 0 % |
| | ATLAS | 56 % | 0 % | 11 % | 16 % | 1 % | 1 % | 0 % | 0 % | 3 % | 0 % |
| PIII | ATLAS | 3 % | 3 % | 2 % | 2 % | 0 % | 0 % | 11 % | 0 % | 3 % | 0 % |
| PPC | BLAS-maq | 19 % | 19 % | 0 % | 0 % | 0 % | 0 % | 9 % | 9 % | 1 % | 1 % |
| R10K | BLAS-maq | 80 % | 80 % | 0 % | 0 % | 34 % | 34 % | 22 % | 0 % | 43 % | 0 % |

Tabla 5.5. Comparación de la desviación del tiempo escogiendo en todos los casos $b$=32 y del tiempo *SOLAR* con respecto al tiempo óptimo conocido a posteriori, para la rutina *DGEQRF* con diferentes tamaños de problema y en diferentes plataformas (en microsegundos).

### 5.2.4 Ciclo de vida de la rutina paralela de factorización QR

En este apartado, se mostrará, a modo de ejemplo, el ciclo de vida de la rutina de factorización *QR* por bloques de la librería paralela ScaLAPACK, *PDGEQRF*. La descripción detallada de esta rutina, así como de su modelo analítico se ha mostrado en el capítulo anterior.

#### Fase de diseño

**Creación de la rutina**: Partimos de una rutina ya implementada dentro de la librería ScaLAPACK.

**Modelado de la rutina:** Como ya se mostró en el capítulo anterior, el modelo analítico de esta rutina, el cual quedaría almacenado en el componente *Model*, sería:

$$T_{ARI} = \frac{4n^3 k_{3\_DGEMM}}{3p} + \frac{n^2 b k_{3\_DTRMM}}{4c} + \frac{n^2 b k_{2\_DGEQR2}}{r} + \frac{n^2 b k_{2\_DLARFT}}{2r}$$

$$T_{COM} = t_s \left[ \frac{n}{b}\left( (2+3b) f_{broad}(r) + 2 f_{broad}(c) \right) \right] + t_w \left[ \frac{n^2}{2}\left( \frac{2(r-1)}{r^2} + \frac{f_{broad}(r)}{c} + \frac{f_{broad}(r)}{r} \right) + nbf_{broad}(p) \right] \quad 5.4$$

**Selección de los valores de experimentación:**. Ahora, como *AP* tenemos, además del tamaño de bloque, *b*, el número de procesadores a utilizar, *p*, entre los *P* procesadores que estén disponibles en la plataforma. Para la configuración lógica en malla bidimensional[*] de los *p* procesadores escogidos tendríamos el número de filas, *r*, y de columnas, *c*. Como valores significativos para *n* y los *AP* se almacenarían en *Preinstallation_information* los valores:

| | | | |
|---|---|---|---|
| **valores para los *AP*:** | | | |
| | $b$ : | 16, 32, 64, 128 | |
| | $p$ : | 1,2,3, ..., $P$ | (teniendo en cuenta la restricción de que $p = r \times c$) |
| | $r$ : | 1,2,3, ..., $p$ | |
| | $c$ : | 1,2,3, ..., $p$ | |
| **valores para el tamaño del problema:** | | | |
| | $n$: | 1024, 2048, 3072, 4096 | |

**Creación del gestor de la rutina:** Dentro del *SOLAR_manager*, el módulo *Instalator* del *SOLAR_manager* tendría que estar formado por siete *SP_measurers*, uno para cada *sp* que encontramos en *Model* (5.4), donde aparecen cuatro *SP* computacionales: $k_{3\_DGEMM}$, $k_{3\_DTRMM}$, $k_{2\_DGEQR2}$ y $k_{2\_DLARFT}$; y tres *SP* de comunicaciones: $t_s$, $t_w$ y $f$.

Los *SP* de computo y, por tanto, sus respectivos *SP_measurers*, son los mismos que los que aparecen en la versión secuencial de esta rutina, anteriormente descrita. En cuanto a los tres *SP_measures* de comunicaciones estarían preparados para realizar medidas de sus correspondientes *SP* mediante experimentaciones con las rutinas básicas de comunicaciones de MPI, teniendo en cuenta los distintos valores de *n* y *AP* recogidos en *Preinstallation_information*.

[*] Como ya se comentó en el capítulo 3, las rutinas de las librerías paralelas de álgebra lineal habitualmente utilizan una topología lógica de procesadores en forma de malla 2D. Esta topología, junto a una distribución de los datos cíclica por bloques, suele ser la más adecuada para el esquema de acceso a los datos que siguen este tipo de software.

## Fase de instalación

Vamos a realizar el proceso de instalación en las plataformas descritas en el capítulo segundo:

- Plataforma Karnak: un multiprocesador SGI Origin 2000, de memoria compartida físicamente distribuida.
- Plataforma Besiberri: un multiprocesador IBM-SP2, de memoria distribuida.
- Plataforma TORC: un *cluster* de PCs conectados por una red FastEthernet.
- Plataforma COCI: un *cluster* de estaciones Sun conectadas por una red Ethernet.

En TORC la librería básica considerada es ATLAS, mientras que en las otras tres será un BLAS específico de la máquina (BLAS-maq).

**Postselección de valores de experimentación**: A la hora de instalar la rutina podemos mantener los valores preseleccionados en *Preinstallation_information* y continuar el proceso.

**Medición de los parámetros del sistema**: Se ponen a funcionar los cuatro *SP_measurers* de cómputo de esta *SOLAR* obteniéndose, evidentemente, los mismos valores que para la rutina *DGEQRF* (Tabla 5.1 y Tabla 5.2).

En cuanto a los *SP* de comunicaciones, los valores medidos por $t_s$*_measurer* y $t_w$*_measurer* se muestran en la Tabla 5.6. En los experimentos realizados mediante una rutina *ping-pong* implementada con MPI (de manera equivalente a los mostrados en la Tabla 3.1) el valor medido de $t_s$ se ha manifestado variable en función del tamaño de mensaje enviado, sin embargo, debido a la poca repercusión que el valor de este parámetro tiene en el tiempo total de comunicación para mensajes que no sean demasiado pequeños, se ha tomado un valor promedio. El valor de $t_w$ ha mostrado un comportamiento constante para los tamaños de mensajes que se manejan en esta rutina.

| | | *b* | | | |
|---|---|---|---|---|---|
| Sistema | *n* | 16 | 32 | 64 | 128 |
| COCI | 1024,...,4096 | 170 / 7.0 | | | |
| TORC | 1024,...,4096 | 60 / 0.7 | | | |
| Besiberri | 1024,...,4096 | 75 / 0.3 | | | |
| Karnak | 1024,...,4096 | 20 / 0.1 | | | |

Tabla 5.6. Valores de $t_s$ y $t_w$ obtenidos en el proceso de instalación de la rutina *PDGEQRF*, en diferentes plataformas (en microsegundos).

Por su parte, en los valores del parámetro $f_{broad}$, medidos por $f_{broad}$*_measurer* mediante una rutina de distribución (*broadcast*), encontramos una clara dependencia del número de procesadores involucrados en la comunicación, siendo esta dependencia distinta en cada plataforma. En la Tabla 5.7 se muestra el coste que representaría los valores obtenidos para este parámetro en cada sistema. Podemos observar que para Besiberri y Karnak, el algoritmo de distribución de datos tiene un comportamiento ideal, en TORC, con una red más lenta, el comportamiento empieza a empeorar, finalmente, en COCI, encontramos que un *broadcast* resulta equivalente a realizar un conjunto de envíos individuales. Todos estos valores medidos experimentalmente se almacenan en *Measured_SP*.

| Sistema | $n$ | $p$ = 1, 2, ...,8 |
|---|---|---|
| COCI | 1024,...,4096 | $p$ |
| TORC | 1024,...,4096 | $1.15 \log_2(p)$ |
| Besiberri | 1024,...,4096 | $\log_2(p)$ |
| Karnak | 1024,...,4096 | $\log_2(p)$ |

Tabla 5.7. Coste que representan los valores de $f_{broad}$ obtenidos en el proceso de instalación de la rutina *PDGEQRF*, en diferentes plataformas.

## Fase de ejecución

Cuando el usuario decida ejecutar la rutina para resolver un problema de tamaño $n_R$ se pondría en marcha este proceso.

**Selección de los parámetros algorítmicos**: Tomando los valores de los *Measured_SP* calculados durante la instalación (Tabla 5.1, Tabla 5.2, Tabla 5.6 y Tabla 5.7) y el valor $n_R$, se busca la combinación de valores de *AP* que minimiza la fórmula de *Model* (5.4). Esta combinación de valores se almacenaría en *Selected_AP*. En la Tabla 5.8 podemos ver los valores que se obtienen para diferentes tamaños de problema y número de procesadores disponibles.

Al igual que ocurría en la versión secuencial de esta rutina, el tamaño de bloque escogido varía de una plataforma a otra, e, incluso para una misma plataforma, es diferente dependiendo del tamaño del problema a resolver.

| | Sistema | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | COCI | | | TORC | | | | | | Besiberri | | | | | | Karnak | | | | | | | | |
| | $P$=4 | | | $P$=4 | | | $P$=8 | | | $P$=4 | | | $P$=8 | | | $P$=4 | | | $P$=8 | | | $P$=16 | | |
| $n$ | $B$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ |
| 1024 | 16 | 1 | 4 | 16 | 1 | 4 | 16 | 1 | 8 | 16 | 1 | 4 | 16 | 1 | 8 | 32 | 4 | 1 | 32 | 4 | 2 | 32 | 4 | 4 |
| 2048 | 16 | 1 | 4 | 32 | 1 | 4 | 16 | 1 | 8 | 32 | 1 | 4 | 16 | 1 | 8 | 64 | 4 | 1 | 32 | 4 | 2 | 32 | 4 | 4 |
| 3072 | | | | 32 | 1 | 4 | 16 | 1 | 8 | 32 | 1 | 4 | 32 | 2 | 4 | 64 | 4 | 1 | 32 | 4 | 2 | 64 | 8 | 2 |
| 4096 | | | | 32 | 1 | 4 | 16 | 1 | 8 | 32 | 1 | 4 | 32 | 2 | 4 | | | | 64 | 4 | 2 | 64 | 8 | 2 |

Tabla 5.8. Valores de *Selected_AP* ($b$, $r$, $c$) justo antes de ejecutar la rutina *PDGEQRF*, para diferentes tamaños de problema y en diferentes plataformas.

En lo referente a la selección de la configuración lógica de los procesadores, observamos que ésta cambia de una plataforma a otra para un mismo número de procesadores. Esto se debe a que en esta rutina, para un número de procesadores dado en cualquiera plataforma, el tiempo de ejecución dedicado a cálculo numérico desciende conforme la topología lógica en malla 2D de los $p$ procesadores adquiere una forma más vertical, es decir, al disminuir el número de columnas, $c$, y aumentar el número de filas, $r$. En cambio, el tiempo dedicado a las comunicaciones tiene el comportamiento contrario, ya que disminuye cuando la topología lógica de los procesadores se hace más horizontal. Por esta razón, dependiendo del peso relativo del tiempo de cálculo y del de comunicaciones en el tiempo total, la topología elegida tenderá hacia una forma más vertical u horizontal. Se da la circunstancia de que esta relación entre el tiempo dedicado a cómputo y el tiempo dedicado a comunicaciones no es la misma en las distintas plataformas. Por un lado tenemos la plataforma Karnak, que cuenta con el sistema de comunicaciones más rápido de las cuatro, donde el principal peso en el tiempo de ejecución de esta rutina lo tiene el tiempo dedicado a cálculo computacional. Consecuentemente, en esta plataforma la topología escogida tiende a ser más vertical. Por contra, en las otras tres plataformas, el mayor peso en el tiempo de ejecución de esta rutina lo tiene el tiempo dedicado a las comunicaciones, por lo que se escogen topologías con formas más horizontales.

En los resultados mostrados en la Tabla 5.8 siempre se escogen tantos procesadores como haya disponible ($p$=$P$). Sin embargo, esto no ocurre cuando el número de procesadores disponibles no permite una topología lógica óptima que los utilice a todos. De esta manera, como se muestra en la Tabla 5.9 para la plataforma Karnak, cuando tenemos $P$=11, la *SOLAR* decide utilizar únicamente 10 de ellos y lo mismo ocurre para $P$=13. Para $P$=14, si el tamaño del problema es pequeño, solo utilizará 12 procesadores, pasando a utilizar 14 cuando el $n$ es mayor o igual que 1024. Más adelante, al ejecutar la rutina se comprobará si son acertadas estas decisiones de dejar algún procesador ocioso.

| | $P$=10 | | | $P$=11 | | | $P$=12 | | | $P$=13 | | | $P$=14 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ | $b$ | $r$ | $c$ |
| 1024 | 32 | 5 | 2 | 32 | 5 | 2 | 32 | 4 | 3 | 32 | 4 | 3 | 32 | 4 | 3 |
| 2048 | 32 | 5 | 2 | 32 | 5 | 2 | 32 | 4 | 3 | 32 | 4 | 3 | 32 | 7 | 2 |
| 3072 | 32 | 5 | 2 | 32 | 5 | 2 | 32 | 4 | 3 | 32 | 4 | 3 | 32 | 7 | 2 |
| 4096 | 64 | 5 | 2 | 64 | 5 | 2 | 32 | 4 | 3 | 32 | 4 | 3 | 32 | 7 | 2 |

Tabla 5.9. Valores de *Selected_AP* ($b$, $r$, $c$) obtenidos justo antes de ejecutar la rutina *PDGEQRF*, para diferentes tamaños de problema y de procesadores disponibles en la plataforma Karnak.

**Ejecución de la rutina:** El *SOLAR_manager* pone a funcionar la rutina utilizando los valores *Selected_AP* (Tabla 5.8 y Tabla 5.9) como parámetros ajustables del algoritmo.

De cara a comprobar la validez de nuestra propuesta, al igual que con la versión secuencial de esta rutina, se han realizado ejecuciones para diferentes tamaños de problema y con las diferentes combinaciones de valores de los *AP* entre las que se encuentra la óptima para diferentes situaciones. En la Tabla 5.10 se muestra una comparación, para cada tamaño de problema, entre:

- El tiempo óptimo: tomando el mínimo obtenido entre todas estas ejecuciones de prueba. Será la cota inferior del tiempo que se puede obtener si se toma la decisión óptima de valores para los *AP*.
- El tiempo promedio: tomando el valor medio para los diferentes valores de los $AP$=$\{b,r,c\}$. Este tiempo promedio representa el que podría tardar la rutina ejecutada por un usuario que escoge los valores de los *AP* al azar, utilizando tantos procesadores como tenga disponibles ($p$=$P$). Esta es la única postura que puede adoptar al no contar con las informaciones teórico-experimentales de la *SOLAR*.
- El tiempo *SOLAR*: el que corresponde con el que se habría obtenido utilizando los *Selected_AP* que la *SOLAR* elige automáticamente en base a la información que maneja (Tabla 5.8).

De igual manera, en la Tabla 5.11 podemos ver el resultado de las decisiones que se tomaron en cuanto al número de procesadores a utilizar ante diferentes situaciones de procesadores disponibles en la plataforma Karnak (Tabla 5.9). Se puede observar que no siempre es conveniente utilizar todos los procesadores disponibles, produciendo de esta manera una notable reducción del tiempo de ejecución, que pasa de un 30%, en promedio, a un 7% de desviación respecto al óptimo.

| | | Tamaño del problema | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1024 | | 2048 | | 3072 | | 4096 | |
| Sistema | | Pro | SOL | Pro | SOL | Pro | SOL | Pro | SOL |
| COCI | *P*=4 | 33 % | 17 % | 33 % | 0 % | 49 % | 0 % | | |
| TORC | *P*=4 | 50 % | 61 % | 23 % | 11 % | 20 % | 7 % | | |
| TORC | *P*=8 | 113 % | 0 % | 43 % | 0 % | 40 % | 24 % | 82 % | 11 % |
| Besiberri | *P*=4 | 133 % | 10 % | 93 % | 0 % | 76 % | 25 % | 65 % | 0 % |
| Besiberri | *P*=8 | 333 % | 0 % | 150 % | 58 % | 97 % | 0 % | 102 % | 1 % |
| Karnak | *P*=4 | 25 % | 12 % | 16 % | 3 % | 50 % | 18 % | | |
| Karnak | *P*=8 | 92 % | 12 % | 102 % | 0 % | 53 % | 0 % | 50 % | 31 % |
| Karnak | *P*=16 | 53 % | 19 % | 42 % | 14 % | 36 % | 12 % | 31 % | 0 % |

Tabla 5.10. Comparación de la desviación del tiempo promedio (izquierda) y del tiempo *SOLAR* (derecha) con respecto al tiempo óptimo conocido a posteriori, para la rutina *PDGEQRF*, con diferentes tamaños de problema y en diferentes plataformas paralelas.

| | Tamaño del problema | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1024 | | 2048 | | 3072 | | 4096 | |
| nº procesadores disponibles | Pro | SOL | Pro | SOL | Pro | SOL | Pro | SOL |
| 10 | 20 % | 7 % | 19 % | 13 % | 20 % | 10 % | 20 % | 7 % |
| 11 | 32 % | 7 % | 43 % | 13 % | 37 % | 10 % | 32 % | 7 % |
| 12 | 23 % | 0 % | 16 % | 4 % | 14 % | 0 % | 23 % | 0 % |
| 13 | 33 % | 0 % | 22 % | 4 % | 23 % | 0 % | 33 % | 0 % |
| 14 | 32 % | 0 % | 15 % | 2 % | | | 32 % | 0 % |

Tabla 5.11. Comparación de la desviación del tiempo promedio (izquierda) y del tiempo *SOLAR* (derecha) con respecto al tiempo óptimo conocido a posteriori, para la rutina *PDGEQRF* con diferentes tamaños de problema, en la plataforma Karnak.

Finalmente, en la Tabla 5.12 se muestran el mismo estudio pero ahora para tamaños de problemas diferentes de los que no se tiene información en *Measured_SP*, con lo que para elegir *Selected_AP* los valores que se toman para los *SP* son el resultado de interpolar entre los correspondientes a los dos tamaños de problemas más cercanos (Tabla 5.1, Tabla 5.2, Tabla 5.6 y Tabla 5.7).

En resumen, tras observar estos resultados con la rutina paralela de factorización QR, podemos afirmar que con nuestra metodología se consigue, en general, una importante disminución del tiempo de ejecución con respecto al tiempo promedio, obteniéndose en muchos casos un tiempo óptimo (desviación del 0% respecto al óptimo conocido a posteriori), para cualquier plataforma, con diferente número de procesadores disponibles y distintos tamaños de problema a resolver.

Para finalizar este apartado, se puede concluir que cuando se cambia la situación en que la rutina se va a ejecutar, es decir, el tamaño de problema y/o el número de procesadores disponibles y/o la plataforma de ejecución; los valores óptimos para los *AP* que presenta esta rutina pueden ser bastante diferentes. Por esta razón, un ajuste manual por parte del usuario se convierte un una labor muy compleja de realizar, mientras que con el sistema de ajuste automático que proponemos se obtienen tiempos de ejecución cercanos al óptimo en cualquier situación en que la rutina sea invocada.

| | | Tamaño del problema | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 500 | | 1500 | | 2500 | | 3500 | |
| Sistema | | Pro | SOL | Pro | SOL | Pro | SOL | Pro | SOL |
| COCI | *P=4* | 31 % | 21 % | 59 % | 44 % | 49 % | 0 % | | |
| TORC | *P=4* | 89 % | 29 % | 31 % | 6 % | 24 % | 0 % | | |
| TORC | *P=8* | 226 % | 5 % | 63 % | 2 % | 46 % | 10 % | 32 % | 12 % |
| Besiberri | *P=4* | 371 % | 0 % | 116 % | 15 % | 81 % | 0 % | 71 % | 0 % |
| Besiberri | *P=8* | 376 % | 0 % | 280 % | 0 % | 145 % | 21 % | 102 % | 1 % |
| Karnak | *P=4* | 61 % | 10 % | 16 % | 3 % | 23 % | 8 % | | |
| Karnak | *P=8* | 122 % | 46 % | 38 % | 5 % | 45 % | 0 % | 50 % | 31 % |

Tabla 5.12. Comparación de la desviación del tiempo promedio (izquierda) y del tiempo *SOLAR* (derecha) con respecto al tiempo óptimo conocido a posteriori, para la rutina *PDGEQRF* con diferentes tamaños de problema y en diferentes plataformas paralelas.

### 5.2.5 Rutina secuencial de factorización LU

Como ya se comentó en el capítulo anterior, hemos implementado una rutina de factorización LU secuencial siguiendo un algoritmo semejante al de la rutina correspondiente de la librería LAPACK (*DGETFR)*. El modelo analítico de esta rutina, como ya se mostró, es:

$$T_{EXEC} = \frac{2}{3} n^3 k_{3\_DGEMM} + bn^2 k_{3\_DTRSM} + \frac{1}{3} b^2 n k_{2\_DGETF2} \qquad 5.5$$

Como valores significativos para el tamaño de problema, *n*, se pueden seleccionar, como se hacía en el caso anterior, los valores {512, 1024, 1536, 2048, 2560, 3072, 3584, 4096}. Para el tamaño de bloque, *b* (el único *ap* que aparece en *Model*), los valores también podrían ser {16, 32, 64, 128}. Estos valores serían almacenados en *Preinstallation_information*. El módulo *Instalator* tendría que estar formado por tres *SP_measurers*, para los tres *SP* que encontramos en *Model*.

El proceso de instalación sería similar al que se ha visto para la rutina QR. Los valores de sus *SP* son semejantes a los de la rutina QR anteriormente expuesta debido a la similitud de los algoritmos que siguen estas rutinas, con lo que los valores de $k_{3\_DTRSM}$ y $k_{3\_DGEMM}$ serían los mostrados en la Tabla 5.1 y para $k_{2\_DGETF2}$ en la Tabla 5.2*.

Cuando el usuario decida ejecutar la rutina para resolver un problema de tamaño $n_R$ se pondría en marcha el proceso de selección de los parámetros algorítmicos. Tomando los valores de los *Measured_SP* calculados durante la instalación (Tabla 5.1 y Tabla 5.2) y el valor $n_R$, se busca el valor de *b* que minimiza la fórmula de *Model* (5.5), el cual se almacena en *Selected_AP*. En la Tabla 5.13 podemos ver los valores del tamaño de bloque que se selecciona para diferentes tamaños del problema a resolver.

Como podemos observar, el tamaño de bloque escogido depende del sistema en el que nos encontremos, tanto de la plataforma hardware como de la librería básica que utilicemos. Además, estos valores son diferentes a los obtenidos para una rutina bastante semejante como la de factorización QR (Tabla 5.3), lo que nos lleva a pensar que tampoco podemos fijar unos valores constantes de los *AP* para cada sistema, siendo necesario un estudio teórico-práctico mediante la metodología que planteamos.

* Esta similitud de la información de auto-optimización que necesitan distintas rutinas es una de las razones, como veremos en el capítulo siguiente, que motivará un nuevo enfoque global a nivel de librería, que permita compartir dicha información entre diferentes rutinas de una misma librería o, incluso, de diferentes librerías que estén relacionadas.

| Sistema | | Tamaño del problema 512 | 1024 | 1536 | 2048 | 2560 | 3072 | 3584 |
|---|---|---|---|---|---|---|---|---|
| SUN1 | BLAS-ref | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| | BLAS-maq | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| | ATLAS | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| SUN5 | BLAS-ref | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | BLAS-maq | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | ATLAS | 128 | 128 | 128 | 128 | 128 | 128 | 128 |
| PPC | BLAS-maq | 128 | 128 | 128 | 128 | 128 | 128 | 128 |
| R10K | BLAS-maq | 128 | 128 | 128 | 128 | 128 | 128 | 128 |

Tabla 5.13. Valores de *Selected_b* obtenidos justo antes de ejecutar la rutina de factorización LU secuencial, para diferentes tamaños de problema y en diferentes plataformas.

Al igual que con la rutina anterior, de cara a comprobar la validez de nuestra propuesta, se han realizado ejecuciones para diferentes tamaños de problema. En la Tabla 5.14 se muestra una comparación entre:

- El tiempo óptimo a posteriori.
- El tiempo promedio, que representa el que podría tardar la rutina ejecutada por un usuario que escoge un tamaño de bloque al azar.
- El tiempo que se habría obtenido utilizando el tamaño de bloque que la *SOLAR* elige.

Como podemos observar, de nuevo en la mayoría de los casos el tamaño de bloque que la *SOLAR* escoge habría proporcionado el tiempo óptimo (desviación del 0%). En el resto de los casos, aunque no se realiza la óptima elección para *b*, se consigue un mejor resultado que si se deja esta elección al azar, reduciendo la desviación media que obtendría el usuario desde un 20% a tan solo un 5%.

| Sistema | | Tamaño del problema 512 | | 1024 | | 1536 | | 2048 | | 2560 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SUN1 | BLAS-ref | 5 % | 2 % | 26 % | 0 % | 18 % | 0 % | | | | |
| | BLAS-maq | 12 % | 0 % | 9 % | 0 % | 8 % | 10 % | | | | |
| | ATLAS | 28 % | 25 % | 20 % | 0 % | 6 % | 0 % | | | | |
| SUN5 | BLAS-ref | 2 % | 0 % | 105 % | 0 % | 10 % | 11 % | 7 % | 4 % | | |
| | BLAS-maq | 11 % | 0 % | 6 % | 0 % | 5 % | 2 % | 3 % | 0 % | | |
| | ATLAS | 10 % | 6 % | 4 % | 0 % | 10 % | 1 % | 6 % | 4 % | | |
| PPC | BLAS-maq | 17 % | 17 % | 6 % | 1 % | 5 % | 0 % | 3 % | 0 % | 6 % | 0 % |
| R10K | BLAS-maq | 60 % | 7 % | 44 % | 1 % | 69 % | 0 % | 14 % | 2 % | 72 % | 1 % |

Tabla 5.14. Comparación de la desviación del tiempo promedio (izquierda) y del tiempo *SOLAR* (derecha) con respecto al tiempo óptimo conocido a posteriori, para la rutina de factorización LU secuencial con diferentes tamaños de problema y en diferentes plataformas (en microsegundos).

### 5.2.6 Rutina paralela de factorización LU

Como ya se mostró en el capítulo anterior, el modelo analítico de esta rutina, el cuál quedaría almacenado en el componente *Model*, sería:

$$T_{ARI} = \frac{2}{3} k_{3\_DGEMM} \frac{n^3}{p} + \frac{r+c}{p} b k_{3\_DTRSM} n^2 + \frac{1}{3} b^2 k_{2\_DGETF2} n$$

$$T_{COM} = t_s \frac{2nd}{b} + t_w \frac{2n^2 d}{p} \qquad 5.6$$

Al igual que ocurría en la rutina de factorización QR, como *AP* tenemos el tamaño de bloque, *b*, y el número de procesadores a utilizar, *p*, entre los *P* disponibles en la plataforma. Para la configuración lógica en malla bidimensional de los *p* procesadores escogidos tendríamos el número de filas, *r*, y de columnas, *c*. De nuevo, como valores significativos para *n* y los *AP* se podrían almacenar en *Preinstallation_information* los valores:

| | | | |
|---|---|---|---|
| **valores para los *AP*:** | | | |
| | *b* : | 16, 32, 64, 128 | |
| | *p* : | 1,2,3, ..., *P* | (teniendo en cuenta la |
| | *r* : | 1,2,3, ..., *p* | restricción de que $p = r \times c$) |
| | *c* : | 1,2,3, ..., *p* | |
| **valores para el tamaño del problema:** | | | |
| | *n*: | 1024, 2048, 3072, 4096 | |

El módulo *Instalator* del *SOLAR_manager* tendría que estar formado por cinco *SP_measurers*, uno para cada *sp* que encontramos en *Model* (5.6), donde aparecen tres *SP* computacionales: $k_{3\_DGEMM}$ , $k_{3\_DTRSM}$ y $k_{2\_DGETF2}$; y dos *SP* de comunicaciones: $t_s$ y $t_w$. En este caso, la rutina, tal como la hemos implementado, no utiliza ninguna operación explícita de comunicación colectiva, con lo que no aparece el parámetro *f*.

Los *SP_measures* de cómputo serían los mismos que los de la versión secuencial anteriormente descrita. En cuanto a los dos *SP_measures* de comunicaciones son semejantes a los de la rutina QR paralela.

El proceso de instalación se ha realizado en las cuatro plataformas paralelas vistas en el capítulo del entorno de trabajo. En TORC la librería básica considerada es ATLAS, en COCI se ha usado ATLAS y un BLAS específico de la plataforma (BLAS-maq), mientras que en Karnak y Besiberri se ha usado únicamente un BLAS específico para cada plataforma. Se ponen a funcionar los *SP_measurers* de cómputo de esta *SOLAR* obteniéndose los mismos valores que para la rutina LU secuencial, pues ambas rutinas realizan las mismas operaciones de cálculo. Por otro lado, los valores medidos por $t_s$*_measurer* y $t_w$*_measurer* son los ya obtenidos anteriormente para la rutina QR paralela (Tabla 3.1). Como siempre, todos los valores medidos de los *SP* se almacenan en *Measured_SP*.

Cuando el usuario decida ejecutar la rutina para resolver un problema de tamaño $n_R$ se pondría en marcha el proceso de selección de los parámetros algorítmicos. Tomando los valores de los *Measured_SP* calculados durante la instalación y el valor $n_R$, se busca la combinación de valores de *AP* que minimiza la fórmula de *Model* (5.6). Esta combinación de valores se almacenaría en *Selected_AP*. En la Tabla 5.15 podemos ver el valor que se obtendría para diferentes tamaños de problema y número de procesadores disponibles. Al igual que ocurría en la versión secuencial de esta rutina, el tamaño de bloque escogido varía de una plataforma a otra, e, incluso para una misma plataforma, es diferente dependiendo del tamaño del problema a resolver.

| | Sistema | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | COCI | | | | | | Besiberri | | | | | | Karnak | | | | | | | | |
| | BLAS-maq | | | ATLAS | | | BLAS-maq | | | | | | BLAS-maq | | | | | | | | |
| | *P*=4 | | | *P*=4 | | | *P*=4 | | | *P*=8 | | | *P*=4 | | | *P*=8 | | | *P*=16 | | |
| *n* | *b* | *r* | *c* | *b* | *r* | *c* | *b* | *r* | *c* | *b* | *r* | *c* | *b* | *r* | *C* | *b* | *r* | *c* | *b* | *r* | *c* |
| 512 | 32 | 1 | 1 | 64 | 1 | 1 | 32 | 2 | 2 | 32 | 4 | 2 | 32 | 2 | 2 | 32 | 4 | 2 | 32 | 4 | 4 |
| 1024 | 32 | 1 | 1 | 64 | 1 | 1 | 32 | 2 | 2 | 32 | 4 | 2 | 64 | 2 | 2 | 32 | 4 | 2 | 32 | 4 | 4 |
| 1536 | 32 | 2 | 2 | 64 | 1 | 1 | 64 | 2 | 2 | 64 | 4 | 2 | 64 | 2 | 2 | 64 | 4 | 2 | 32 | 4 | 4 |
| 2048 | 32 | 2 | 2 | 16 | 2 | 2 | 64 | 2 | 2 | 64 | 4 | 2 | 64 | 2 | 2 | 64 | 4 | 2 | 64 | 4 | 4 |
| 2560 | | | | | | | 64 | 2 | 2 | 64 | 4 | 2 | 128 | 2 | 2 | 64 | 4 | 2 | 64 | 4 | 4 |
| 3072 | | | | | | | 64 | 2 | 2 | 64 | 4 | 2 | 128 | 2 | 2 | 64 | 4 | 2 | 64 | 4 | 4 |
| 3584 | | | | | | | 64 | 2 | 2 | 64 | 4 | 2 | 128 | 2 | 2 | 64 | 4 | 2 | 64 | 4 | 4 |

Tabla 5.15. Valores de *Selected_AP* (*b*, *r*, *c*) obtenidos justo antes de ejecutar la rutina de factorización LU paralela, para diferentes tamaños de problema y en diferentes plataformas.

Es importante resaltar cómo para la plataforma COCI, debido al alto coste de las comunicaciones en este sistema, para tamaños de problema pequeños (hasta 1024 con la librería BLAS-maq y hasta 1536 con ATLAS) la *SOLAR* decide usar un único procesador y ejecutar la versión secuencial de la rutina. Para tamaños de problema mayores, el tiempo de cómputo aritmético, que es de $O(n^3)$, empieza a tener un mayor peso en el tiempo total de ejecución frente al tiempo de comunicaciones, que es de $O(n^2)$, con lo que la versión paralela pasa a ser la seleccionada. Además, también se observa que para ningún tamaño de problema se decide usar 2 ó 3 procesadores. Esto es debido a que la topologías lógicas que mejor tiempo de ejecución proporcionan para esta rutina tienden a ser de forma lo más cuadrada posible.

Como podemos observar en la Tabla 5.16, se consigue una importante disminución del tiempo de ejecución con respecto al tiempo promedio para cualquier tamaño de problema y en los diferentes sistemas, obteniéndose, para esta rutina, el tiempo óptimo en la mayoría de los casos. La desviación media respecto al óptimo se reduce de un 40 % a un 3 %.

| | Sistema | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | COCI | | | | Besiberri | | | | Karnak | | | | | |
| | BLAS-maq | | ATLAS | | BLAS-maq | | | | BLAS-maq | | | | | |
| *n* | *p*=4 | | *p*=4 | | *p*=4 | | *p*=8 | | *p*=4 | | *p*=8 | | *p*=16 | |
| 512 | 84 % | 0 % | 97 % | 0 % | 47 % | 0 % | 55 % | 7 % | 39 % | 0 % | 99 % | 0 % | 50 % | 25 % |
| 1024 | 63 % | 0 % | 45 % | 4 % | 42 % | 0 % | 9 % | 9 % | 33 % | 0 % | 73 % | 12 % | 25 % | 0 % |
| 1536 | 51 % | 6 % | 74 % | 0 % | 47 % | 0 % | 6 % | 6 % | 33 % | 0 % | 50 % | 20 % | 11 % | 0 % |
| 2048 | 40 % | 5 % | 47 % | 0 % | 50 % | 0 % | 7 % | 3 % | 39 % | 0 % | 48 % | 0 % | 7 % | 0 % |
| 2560 | | | | | 50 % | 0 % | 8 % | 0 % | 38 % | 5 % | 32 % | 0 % | 11 % | 0 % |
| 3072 | | | | | 48 % | 0 % | 6 % | 0 % | 39 % | 7 % | 35 % | 2 % | 10 % | 0 % |
| 3584 | | | | | 51 % | 0 % | 5 % | 0 % | 32 % | 0 % | 45 % | 0 % | 26 % | 0 % |

Tabla 5.16. Comparación de la desviación del tiempo promedio (izquierda) y del tiempo *SOLAR* (derecha) con respecto al tiempo óptimo conocido a posteriori, para la rutina de factorización LU paralela con diferentes tamaños de problema y en diferentes plataformas.

### 5.2.7 Rutina secuencial de factorización de Cholesky

En [CGG+03a] se ha realizado un estudio similar para una implementación de la rutina de factorización de Cholesky siguiendo el mismo algoritmo que la rutina correspondiente de las librería LAPACK (*DPOTFR*). Como ya se mostró en el capítulo anterior, el modelo analítico de esta rutina es:

$$T_{EXEC} = 2k_{3\_DGEMM}\left(\frac{n^3}{6}-\frac{n^2b}{2}+\frac{nb^2}{3}\right)+k_{3\_DSYRK}\left(\frac{n^2}{2}-\frac{nb}{2}\right)(b+1)+ \\ k_{3\_DTRSM}\left(\frac{n^2b}{2}-\frac{nb^2}{2}\right)+k_{2\_DPOTF2}\left(\frac{nb^2}{3}\right) \quad 5.7$$

El proceso de instalación se ha realizado para dos procesadores diferentes de la plataforma TORC, un Pentium III y un Pentium 4, utilizando dos posibles librerías básicas, un BLAS de referencia y un BLAS precompilado para Pentium III. Cuando el usuario decida ejecutar la rutina para resolver un problema de tamaño $n_R$ se pondría en marcha el proceso de selección de los parámetros algorítmicos. Tomando los valores de los *Measured_SP* calculados durante la instalación se busca el valor de $b$ que minimiza la fórmula de *Model* (5.7), el cual se almacena en *Selected_AP*. En la Tabla 5.17 podemos ver los valores del tamaño de bloque que se selecciona para diferentes tamaños del problema a resolver.

Como podemos observar, el tamaño de bloque que proporciona un tiempo de ejecución teóricamente mínimo depende del sistema en el que nos encontremos. Además, estos valores son diferentes a los obtenidos para la rutina de factorización QR (Tabla 5.3) y para la factorización LU (Tabla 5.13), lo cuál refuerza nuestro planteamiento sobre la necesidad de incluir análisis teórico-prácticos para cada rutina.

| | | Tamaño del problema | | | |
|---|---|---|---|---|---|
| Sistema | | 256 | 512 | 1024 | 2048 |
| Pentium III | BLAS-ref | 32 | 32 | 32 | 32 |
| | BLAS PIII | 64 | 64 | 128 | 128 |
| Pentium 4 | BLAS-ref | 32 | 32 | 32 | 32 |
| | BLAS PIII | 64 | 64 | 128 | 128 |

Tabla 5.17. Valores de *Selected_b* obtenidos justo antes de ejecutar la rutina de factorización de Cholesky secuencial, para diferentes tamaños de problema, en la plataforma TORC.

Al igual que con las rutinas anteriores, de cara a comprobar la validez de nuestra propuesta, se han realizado ejecuciones para diferentes tamaños de problema. Como podemos observar en la Tabla 5.18, en la amplía mayoría de los casos el tamaño de bloque que escoge la *SOLAR* habría proporcionado el tiempo óptimo (desviación del 0%).

| | | Tamaño del problema | | | |
|---|---|---|---|---|---|
| Sistema | | 256 | 512 | 1024 | 2048 |
| Pentium III | BLAS-ref | 1 % | 0 % | 0 % | 0 % |
| | BLAS PIII | 0 % | 10 % | 0 % | 0 % |
| Pentium 4 | BLAS-ref | 1 % | 0 % | 0 % | 0 % |
| | BLAS PIII | 0 % | 0 % | 0 % | 0 % |

Tabla 5.18. Desviación del tiempo *SOLAR* con respecto al tiempo óptimo conocido a posteriori, para la rutina de factorización de Cholesky secuencial con diferentes tamaños, en la plataforma TORC.

### 5.2.8 Rutina paralela de Jacobi unilateral para el problema de valores propios

La descripción detallada de esta rutina, así como de el modelo analítico de su tiempo de ejecución se ha mostrado en el capítulo anterior. El modelo analítico de esta rutina, donde consideramos una topología lógica de los procesadores en forma de malla rectangular $2^r \times 2^c$, que quedaría almacenado en *Model*, sería:

$$T_{ARI} = \frac{9n^3 k_{3\_DGEMM}}{p} + \frac{12n^2 b k_{1\_DROT}}{2^r}$$

$$T_{COM} = \frac{2n}{b}\left(t_s + (2\frac{n}{2^c}b + b^2)t_w\right) + \frac{n^2}{b^2 2^{r+1}}\left((2c-1)(t_s + b^2 t_w) + (c-1)b^2 k_{1\_DCOPY}\right) \qquad 5.8$$

El módulo *Instalator* del SOLAR*_manager* tendría que estar formado por cuatro *SP_measurers*, uno para cada *sp* que encontramos en *Model* (5.8). El proceso de instalación se ha realizado en Karnak y se ha usado únicamente un BLAS específico para la plataforma. Cuando el usuario decida ejecutar la rutina para resolver un problema de tamaño $n_R$ se pondría en marcha el proceso de selección de los parámetros algorítmicos (Tabla 5.19). Esta combinación de valores se almacenaría en *Selected_AP*. De nuevo, como se puede observar, no se puede saber a priori una combinación óptima de valores de los *AP*, haciendo necesario este proceso de instalación teórico-práctico apoyado en el *Model* y acompañado de la experimentación necesaria para medir los *SP* de la plataforma.

Finalmente, se han realizado ejecuciones para diferentes tamaños de problema. Como podemos observar en la Tabla 5.20, se consigue una importante disminución del tiempo de ejecución con respecto al tiempo promedio para cualquier tamaño de problema y en los diferentes sistemas, obteniéndose en muchos casos el tiempo óptimo.

| | *P*=4 | | | *P*=8 | | |
|---|---|---|---|---|---|---|
| *n* | *b* | $2^r$ | $2^c$ | *b* | $2^r$ | $2^c$ |
| 512 | 64 | 4 | 1 | 32 | 8 | 1 |
| 1024 | 64 | 4 | 1 | 64 | 8 | 1 |
| 1536 | 128 | 4 | 1 | 64 | 8 | 1 |
| 2048 | 128 | 4 | 1 | 64 | 8 | 1 |
| 2560 | 128 | 4 | 1 | 128 | 8 | 1 |
| 3072 | 128 | 4 | 1 | 128 | 8 | 1 |

Tabla 5.19. Valores de *Selected_AP* (*b*, $2^r$,$2^c$) obtenidos justo antes de ejecutar la rutina de Jacobi paralela. Para diferentes tamaños de problema en la plataforma Karnak.

| *n* | *P*=4 | | *P*=8 | |
|---|---|---|---|---|
| 512 | 62% | 0% | 158% | 0% |
| 1024 | 34% | 0% | 63% | 5% |
| 1536 | 68% | 0% | 112% | 0% |
| 2048 | 24% | 0% | 66% | 8% |
| 2560 | 57% | 0% | 47% | 0% |
| 3072 | 11% | 3% | 58% | 0% |

Tabla 5.20. Comparación de la desviación del tiempo promedio (izquierda) y del tiempo *SOLAR* (derecha) con respecto al tiempo óptimo conocido a posteriori. Tiempos para la rutina de Jacobi paralela con diferentes tamaños de problema en la plataforma Karnak.

## 5.3 Plataformas con carga de trabajo variable

Como se ha mostrado en la sección anterior, en sistemas donde la carga de trabajo permanece más o menos constante, la primera versión de *SOLAR* ofrece una más que aceptable adaptación a las condiciones del sistema (capacidad de cómputo de los procesadores y capacidad de comunicación de la red de interconexión), mediante un proceso que se lleva a cabo durante la instalación de la rutina. Sin embargo, si nos encontramos con una plataforma donde la carga de trabajo sufra importantes variaciones frecuentemente, tanto en los distintos procesadores como en la red de interconexión, cualquier proceso de ajuste de una rutina que se realice totalmente durante su instalación puede resultar improductivo si, a la hora de ejecutar dicha rutina, la carga del sistema ha cambiado notablemente respecto al momento en que se instaló. Así, por ejemplo, si la red de interconexión está muy sobrecargada puede ser conveniente utilizar menos procesadores de los previstos inicialmente y de esta manera poder reducir el coste de las comunicaciones. De igual manera se pueden producir variaciones en el tamaño óptimo de bloque de cómputo cuando la carga de los procesadores crece. Por otro lado, si la carga de la plataforma es heterogénea, puede ser conveniente utilizar solamente los procesadores que se encuentren más libres de trabajo.

Como ya se mostró en el capítulo 2, existen en la actualidad diferentes proyectos de ajuste de software de álgebra lineal sobre plataformas paralelas, donde podemos distinguir dos grupos:

- Un primer grupo estaría formado por aquellos que no realizan ninguna medida de la carga de trabajo de la plataforma (Multicomputer Toolbox [SLS+94], ILIB [KKK01] y Zhang *et al.* [ZC03]).
- Un segundo grupo lo formarían todos aquellos que realizan el ajuste de la rutina teniendo en cuenta la carga que soporta la plataforma en el momento de ejecutar la rutina. A su vez, podemos distinguir dos subgrupos, teniendo en cuenta el momento de realizar el ajuste de la rutina :
    - Un primer subgrupo de proyectos estaría formado por los que realizan la totalidad del proceso de ajuste cuando se va a ejecutar la rutina (GTP [GAM+03][GLR+02], Yves Robert *et al.* [BLR+01], Mills *et al.* [CMS03], LFC [CDL+03]). El inconveniente de esta primera aproximación es que el proceso de ajuste puede introducir una importante sobrecarga en el tiempo de ejecución de la rutina.
    - Un segundo subgrupo, en el que estaría incluida nuestra propuesta, estaría formado por aquellos proyectos que realizan parte del proceso durante la instalación y otra parte a la hora de ejecutar la rutina (mpC [Las03], PINCO [CDG01]). El inconveniente que encontramos en mpC es que para testear el estado de la plataforma utiliza una rutina a modo de *benchmark*, lo que puede introducir una importante sobrecarga en el tiempo de ejecución. Por otro lado, en el proyecto PINCO no consideran la posibilidad de que la red de interconexión pueda ser también heterogénea.

En nuestra propuesta, la parte más costosa del proceso, donde se capturan las condiciones estáticas de la plataforma (capacidad de cómputo de los procesadores y capacidad de comunicación de la red de interconexión) se realiza durante la instalación. Cuando se va a ejecutar la rutina se lleva a cabo un proceso de ajuste de la información que se obtuvo durante la instalación, para adaptarla a la carga de trabajo que en ese momento esté soportando cada parte de la plataforma. Como se describirá a continuación, este proceso de ajuste en tiempo de ejecución introduce una mínima sobrecarga en el tiempo de ejecución de la rutina, ya que no se realiza ningún tipo de experimentación para conocer la situación de la plataforma, sino que se consulta la información que aporta la herramienta NWS, realizándose a continuación un ajuste lineal de los valores de los parámetros del sistema en función de dicha información.

En el resto de esta sección, se van a describir, en primer lugar, las modificaciones y ampliaciones que se necesita introducir en la arquitectura de la *SOLAR* para conferirle la capacidad de adaptarse a posibles variaciones en la carga de trabajo del sistema donde esté instalada. Seguidamente, se mostrará el esquema del ciclo de vida de esta nueva versión de *SOLAR*, comenzando desde su diseño, siguiendo por las tareas que se realizan durante su instalación y finalizando por el proceso que se lleva a cabo cuando se invoca la rutina para su ejecución. Finalmente se mostrará experimentalmente la funcionalidad de esta propuesta, detallando cómo se verían modificados los ciclos de vida de diferentes rutinas y los resultados experimentales obtenidos.

### 5.3.1 Arquitectura de una *SOLAR* dinámica (*D-SOLAR*)

En la Figura 5.3 podemos ver un esquema de la arquitectura de una *D-SOLAR*. Al igual que en la primera versión, encontramos dos componentes de código (*LAR* y *SOLAR_manager*) y una serie de componentes de datos, agrupados según la etapa en la vida de una *D-SOLAR* en que se instancian con información útil.

Al igual que en la versión inicial, *LAR* es la rutina de álgebra lineal original, sin ninguna modificación, distinguiendo entre sus parámetros los operandos de entrada/salida de la rutina, *OPR*, el tamaño del problema a resolver, *n,* y el conjunto de parámetros del algoritmo que pueden ser ajustados, *AP*.

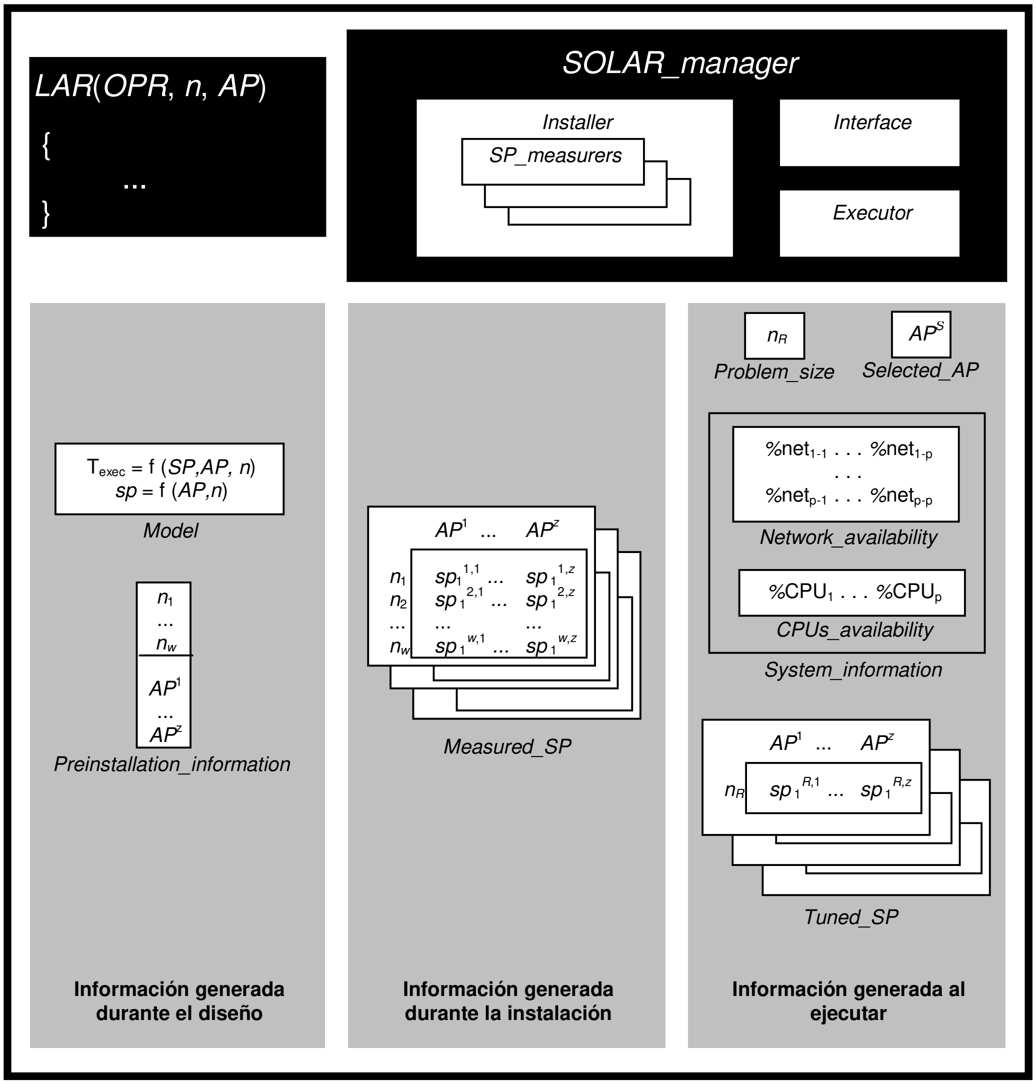


Figura 5.3. Arquitectura de una *D-SOLAR*.

El *SOLAR_manager* gestiona toda la estructura de información de la *D-SOLAR* durante su instalación y ejecución, haciendo de interfaz con el exterior. Su función durante la instalación y su interfaz no cambian sustancialmente respecto a la versión anterior. Las modificaciones aparecen en su modulo *Ejecutor*, que es el encargado de la elección de los valores más apropiados para los *AP* y la puesta en ejecución de la *LAR* con estos valores. En la primera versión su función era la de buscar los valores de los *AP* más apropiados al tamaño de problema a resolver, *n*, en el sistema actual descrito con los parámetros *SP.* En esta segunda versión, previamente a esa selección de los *AP,* el ejecutor tendrá que consultar la situación de la carga de la plataforma y ajustar los valores de los *SP* acorde con esa información. Su funcionamiento será descrito detalladamente en el siguiente apartado.

Al igual que en la versión estática, en *Model* queda reflejado su tiempo de ejecución en función del tamaño a resolver, *n*, de los parámetros algorítmicos, *AP*, y de los parámetros que caracterizan el sistema donde la rutina se instala, *SP*. Por otro lado, en el componente *Preinstallation_information* quedan recogidos los valores de *n* y de los *AP* que serán utilizados durante la instalación de la rutina para medir cada *sp*. Durante la instalación de la rutina, en cada *Measured_*$sp_i$ se almacenan los valores de $sp_i$ que se miden experimentalmente para las diferentes combinaciones de $\{AP,n\}$ recogidas en *Preinstallation_information*.

Los principales variaciones en la arquitectura de esta segunda versión de *SOLAR* se dan en los componentes cuya información se genera en la fase de ejecución de la rutina.

Por un lado, existen dos componentes cuya información que contienen y función que realizan no cambian. Dichos componentes serían *Problem_size*, que almacenará el tamaño del problema a resolver, $n_R$, y *Optimum_AP*, que almacenará la combinación de valores de los *AP* que, según la información que se maneja sobre las características de la rutina y del sistema, minimizarán el tiempo de ejecución.

Por otro lado, aparecen dos nuevos componentes: *System_information* y *Tuned_SP*. En *System_information* se va almacenar la información sobre el estado de la plataforma cuando se va a ejecutar la rutina. Este componente contendrá, la disponibilidad para ejecutar nuevos procesos en cada uno de los *P* procesadores de la plataforma, *CPUs_availability*, y la disponibilidad de la red de interconexión entre cada par de estos procesadores, *Network_availability*. *Tuned_SP* va a contener los valores de los *SP* para el tamaño de problema a resolver, $n_R$, y las diferentes combinaciones de los *AP*. Estos valores se obtendrán tomando de *Measured_SP* los valores de los *SP* para el tamaño de problema a resolver, $n_R$, y ajustándolos a las condiciones del sistema recogidas en *System_information*. Este ajuste consistirá básicamente en aumentar los valores de cada *sp* de cómputo proporcionalmente a la disminución de la disponibilidad de cada procesador y, de igual manera, cada *sp* de comunicaciones aumentará en proporción a la reducción de la capacidad de la red para realizar comunicaciones entre cada par de nodos.

### 5.3.2 Ciclo de vida de una *D-SOLAR*

El ciclo de vida de una *SOLAR* dinámica (Figura 5.4) coincide con el de la versión anterior en sus fases de diseño e instalación. De esta manera, en la fase de diseño se implementa la rutina de álgebra lineal, *LAR* (también se puede partir de una ya implementada con anterioridad), se realiza el modelo analítico de ésta, *Model*, y se implementa su gestor, *SOLAR_manager*. A la hora de instalar la rutina en un sistema se recogen las características estáticas de este sistema mediante la medición de los parámetros del sistema, *SP*. En la fase de ejecución es donde aparecen las diferencias en esta segunda versión de *SOLAR*. Primero, se consultará el estado de la plataforma, tras ello, teniendo en cuenta esta última información, se ajustarán los valores de los parámetros del sistema, finalmente, se seleccionarán los valores óptimos de los *AP*, según la información que se maneja, y se ejecutará la *LAR*. A continuación se detallará esta última fase y se verá, a modo de ejemplo, cómo se llevaría a cabo la ejecución de la rutina de factorización QR en su versión secuencial.

#### Fase de Ejecución

Al igual que antes, cuando un usuario quiere ejecutar una *D-SOLAR* lo único que tiene que realizar es llamar a la rutina pasándole los operandos, *OPR*, y el tamaño del problema a resolver,

$n_R$. Esta orden de ejecución es recogida por el módulo *Interface* del *SOLAR_manager* que pone en marcha al módulo *Executor*. Este último realiza las siguientes tareas:

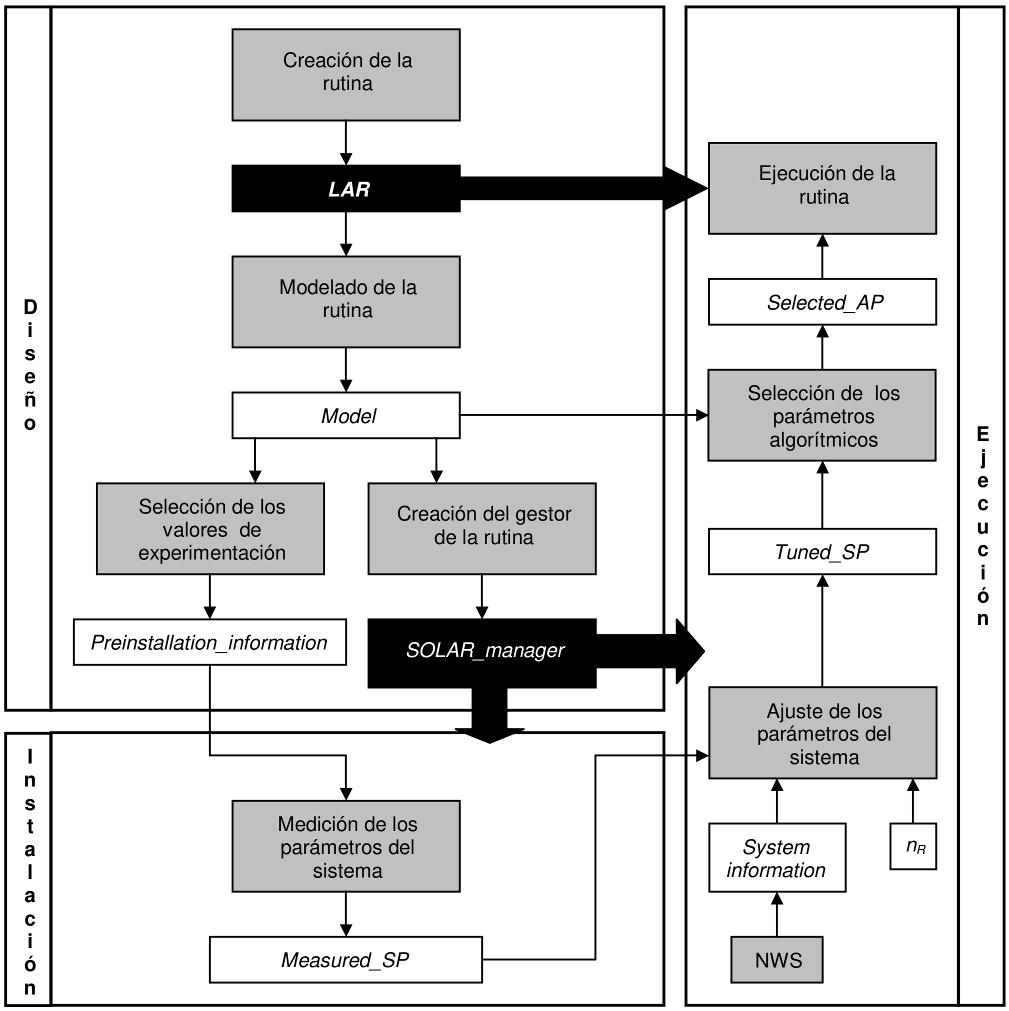


Figura 5.4. Ciclo de vida de una D-SOLAR.

**Consulta el estado de la plataforma:** La herramienta NWS*, u otra herramienta similar que se encuentre instalada, se encarga de testear el sistema recogiendo diversa información sobre el estado de la plataforma. Lo único que tiene que hacer el módulo *Executor* es consultar la información generada por esta herramienta, lo que le supone un tiempo despreciable (inferior a un 1% del tiempo total de ejecución [PBD+01]) e ir escribiendo en el componente *System_information* el grado de disponibilidad de cada procesador *i*, $CPU_i$*_availability*, y el grado de disponibilidad de la red de interconexión entre cada par de nodos (*i*,*j*), $Network_{i\text{-}j}$*_availability*.

* NWS es una herramienta software que proporciona mediciones y predicciones de diversas características de un sistema en un momento dado. La versión actual de NWS ofrece información sobre de la disponibilidad de cada procesador para ejecutar nuevos procesos, así como los que están en ejecución en ese momento. De igual manera informa sobre el tiempo de conexión, el ancho de banda y la latencia entre cada par de nodos de la plataforma. Se puede utilizar NWS tanto en multicomputadores y redes locales como en un entorno distribuido (*grid environment*).

**Ajuste de los parámetros del sistema:** El *Executor* realiza un ajuste de los valores de los *SP* para $n_R$, que durante la instalación fueron calculados y almacenados en *Measured_SP*, teniendo en cuenta *System_information*. Este ajuste consiste en dividir el valor de cada *sp* de cómputo, $Comp\_sp_k$, para cada combinación de *AP*, $AP^z$, para cada procesador *i* de la plataforma, $processor_i$, por la disponibilidad de este procesador, $CPU_i\_availability$:

$$Tuned\_Comp\_sp_k(processor_i, AP^z) = \frac{Measured\_Comp\_sp_k(processor_i, n_R, AP^z)}{CPU_i\_availability} \qquad 5.9$$

De igual manera, se divide el valor de cada *sp* de comunicaciones, $Comm\_sp_k$, para cada combinación de *AP*, $AP^z$, para cada par de procesadores (*i*,*j*), $pair_{i\text{-}j}$, por la disponibilidad de la red de interconexión entre ese par de nodos, $Network_{i\text{-}j}\_availability$:

$$Tuned\_Comm\_sp_k(pair_{i-j}, AP^z) = \frac{Measured\_Comm\_sp_k(pair_{i-j}, n_R, AP^z)}{Network_{i-j}\_availability} \qquad 5.10$$

**Selección de los parámetros algorítmicos:** El *Executor* realiza una búsqueda exhaustiva de los valores de los *AP* que, junto a los correspondientes valores de los *Tuned_SP* y con $n=n_R$, minimizarán el tiempo de ejecución de la rutina según *Model*. Este proceso consiste en calcular el tiempo teórico de ejecución de la rutina para cada combinación de los parámetros algorítmicos $AP^z$, con el tamaño de problema fijado por el usuario, $n_R$, junto a los valores de los *SP* almacenados en *Tuned_SP* para la combinación $AP^z$. Aquella combinación de valores de los *AP* que proporcione el menor tiempo teórico de ejecución se almacenará en *Selected_AP*.

**Ejecución de la rutina:** El *Executor* pone a funcionar la rutina con los parámetros *OPR* y $n_R$ que el usuario le proporcionó, junto con los valores *Selected_AP* como valores elegidos para los parámetros ajustables del algoritmo.

## Ciclo de vida de la rutina secuencial de factorización QR

El proceso de diseño e instalación sería similar al visto anteriormente para plataformas con carga de trabajo constante (apartado 5.2.3), de manera que *Model* sería el descrito en la ecuación 5.3 y los valores de *Measured_SP* serían los mostrados en Tabla 5.1 para $k_{3\_DTRMM}$ y $k_{3\_DGEMM}$, y en Tabla 5.2 para $k_{2\_DGEQR2}$ y $k_{2\_DLARFT}$. En cuanto a la fase de ejecución, veamos un caso concreto a modo ejemplo para ocho Pentium III unidos por FastEthernet de la plataforma TORC y el tamaño de problema a resolver $n_R$=1024.

**Consulta del estado de la plataforma:** NWS informa que la disponibilidad del procesador para ejecutar nuevos procesos es de, por ejemplo, un 40% debido a que, en estos momentos, hay otros procesos ejecutándose en esta plataforma, lo que disminuye su capacidad de cómputo notablemente. El *Executor* almacena este valor en *CPU_availability*.

**Ajuste de los parámetros del sistema:** El *Executor* toma los valores de *Measured_SP* para $n=n_R$ y les aplica el ajuste lineal, dividiéndolos por el valor almacenado en *CPU_availability*. Los resultados se almacenan en *Tuned_SP* (Tabla 5.21).

**Selección de los parámetros algoritmos:** El *Executor* calcula el tiempo teórico de ejecución de la rutina, según *Model* (ecuación 5.3), para cada valor del tamaño de bloque *b* (único *ap* de esta rutina secuencial) con el tamaño de problema fijado por el usuario, $n_R$=1024, junto a los valores de los *SP* almacenados en *Tuned_SP* (Tabla 5.21). En la Tabla 5.22 se muestran cuáles serían los tiempos teóricos que se obtendrían en estas condiciones para diferentes tamaños de bloque. Como se puede apreciar, teóricamente el menor tiempo se obtiene para *b*=64, con lo que éste sería el valor almacenado en *Selected_AP*.

| $k_{3\_DTRMM} \approx k_{3\_DGEMM}$ | Tamaño de bloque, *b* | | | |
|---|---|---|---|---|
| | 16 | 32 | 64 | 128 |
| $n_R$ =1024 | 0.0095 µs | 0.0083 µs | 0.0075 µs | 0.0075 µs |

| $k_{2\_DGEQR2} \approx k_{2\_DLARFT}$ | Tamaño de bloque, *b* | | | |
|---|---|---|---|---|
| | 16 | 32 | 64 | 128 |
| $n_R$ =1024 | 0.0375 µs | 0.0375 µs | 0.0375 µs | 0.0375 µs |

Tabla 5.21. Valores de los *SP* de la rutina *DGEQRF* ajustados en tiempo de ejecución en el sistema TORC+ATLAS, con un 40% de disponibilidad del procesador, para $n_R$=1024.

| | *b* | | | |
|---|---|---|---|---|
| | 16 | 32 | 64 | 128 |
| $n_R$=1024 | 13.95 | 12.51 | **12.15** | 13.56 |

Tabla 5.22. Tiempos teóricos de la rutina *DGEQRF*, para diferentes tamaños de bloque, en TORC+ATLAS, con un 40% de disponibilidad del procesador, para $n_R$=1024.

Finalmente, el *Executor* invocaría la rutina, pasándole los parámetros de entrada del usuario, junto al tamaño de bloque *b*=64. En la Tabla 5.23 se muestra los tiempos de ejecución obtenidos no sólo para el tamaño de bloque escogido, sino también para los otros posibles.

| | *b* | | | |
|---|---|---|---|---|
| | 16 | 32 | 64 | 128 |
| $n_R$=1024 | 14.55 | 12.05 | 11.22 | 13.82 |

Tabla 5.23. Tiempos de ejecución reales de la rutina *DGEQRF*, para diferentes tamaños de bloque, en TORC+ATLAS, con un 40% de disponibilidad del procesador, para $n_R$=1024.

Comparando estos valores con los de la Tabla 5.22 podemos comprobar que la fidelidad del modelo teórico, apoyado con el ajuste dinámico que se ha realizado, nos ha permitido escoger adecuadamente el valor óptimo del único *ap* que teníamos. Frente a esta aproximación, si no se hubiese tenido en cuenta la disponibilidad de la CPU se habría tomado una peor decisión, al seleccionar *b*=32 (Tabla 5.3). Un proceso similar se realizaría para cualquier otro tamaño de problema, con las demás rutinas y en cualquiera de las plataformas.

### 5.3.3 Resultados experimentales

En este apartado se va a mostrar experimentalmente la funcionalidad de esta propuesta de ajuste a las características dinámicas de carga de trabajo de la plataforma. Se detallará cómo se vería modificado el comportamiento de algunas de las rutinas estudiadas cuando se enfrentan a varias situaciones de carga de trabajo en los procesadores y redes de interconexión de la plataforma.

#### Rutina paralela de factorización LU en TORC

En la Tabla 5.24 se describen varios escenarios de carga de trabajo de ocho Pentium III conectados por FastEthernet en la plataforma TORC. Estos escenarios se han conseguido de manera artificial poniendo a funcionar en los distintos nodos un número variado de procesos improductivos (*dummy jobs*), que consumen potencia de cálculo y de comunicación.

En el escenario A, no existe carga en la plataforma, teniendo totalmente disponibles los 8 nodos y su red de interconexión, en el escenario B encontramos que en los nodos 1-4 se están ejecutando varios procesos que reducen la disponibilidad de sus procesadores al 80% y la capacidad de comunicación al 90%, posteriormente, en el escenario C, aumenta la carga de estos mismos nodos. En el escenario D, los cuatro primeros nodos mantienen la misma carga, pero aparece algo de carga en los nodos 7 y 8. Finalmente el escenario E los nodos 7 y 8 se encuentran tremendamente cargados, teniendo pues muy poca disponibilidad para ejecutar nuevos procesos.

En cada uno de estos escenarios hemos puesto a funcionar dos *SOLARs* de factorización LU, una de primera generación, con ajuste estático en tiempo de instalación (*S-SOLAR*), y una de segunda generación que realiza el ajuste dinámico de los valores de los *SP* cuando es invocada para su ejecución (*D-SOLAR*). Dado un tamaño de problema, la *S-SOLAR* siempre elige los mismos valores para los *AP* independiente de la carga de trabajo de la plataforma (primera columna de la Tabla 5.25), mientras que la *D-SOLAR*, irá cambiando su decisión para ajustarse a la situación de la plataforma en cada momento (todas las columnas de la Tabla 5.25).

En los escenarios A y B, la *D-SOLAR* sigue escogiendo los valores de los *AP* iniciales. En el escenario C, como estamos manejando una rutina que realiza un reparto homogéneo de trabajo entre los procesadores, si se utilizan los 8 nodos la ejecución se ralentiza debido a que los nodos 1-4 marcarán un ritmo de ejecución muy lento al conjunto. Por esta razón, la *D-SOLAR* decide utilizar solamente los nodos 5-8, configurados en una malla 2×2. En el escenario D, la *D-SOLAR* toma la misma decisión, pese a tener algo de carga los nodos 7-8. Finalmente, en el escenario E, la carga de los nodos 7-8 es tan alta que la *D-SOLAR* decide prescindir de ellos y ejecutar únicamente en los nodos 5-6 configurados en forma 2×1.

En la Tabla 5.26 se muestra una comparativa para estos 5 escenarios estudiados entre el tiempo de ejecución óptimo (*optimum*), obtenido a posteriori, tras ejecutar la rutina para todos los posibles valores de los *AP* y tomando el menor, el tiempo obtenido con la *S-SOLAR* y el tiempo obtenido con la *D-SOLAR*. Como se puede apreciar, en los escenarios A y B, ambas *SOLARs* tienen el mismo comportamiento, con unas buenas aproximaciones al tiempo óptimo. Sin embargo, cuando la carga de la plataforma empieza a aumentar y se vuelve más heterogénea (escenarios C, D y E), se observa cómo las decisiones de la *S-SOLAR* ya no son tan buenas, mientras que la versión de ajuste dinámico mantiene una buena aproximación al tiempo óptimo en cualquiera de los casos.

| | nodo 1 | nodo 2 | nodo 3 | nodo 4 | nodo 5 | nodo 6 | nodo 7 | nodo 8 |
|---|---|---|---|---|---|---|---|---|
| **Escenario A** | | | | | | | | |
| Disponibilidad de CPU | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 100% | | | | | | | |
| **Escenario B** | | | | | | | | |
| Disponibilidad de CPU | 80% | 80 % | 80 % | 80 % | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 90% | | | | 100% | | | |
| **Escenario C** | | | | | | | | |
| Disponibilidad de CPU | 60 % | 60 % | 60 % | 60 % | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 40% | | | | 100% | | | |
| **Escenario D** | | | | | | | | |
| Disponibilidad de CPU | 60 % | 60 % | 60 % | 60 % | 100 % | 100 % | 80 % | 80 % |
| Disponibilidad de la red | 40% | | | | 100% | | 90% | |
| **Escenario E** | | | | | | | | |
| Disponibilidad de CPU | 60 % | 60 % | 60 % | 60 % | 100 % | 100 % | 50 % | 50 % |
| Disponibilidad de la red | 40% | | | | 100% | | 20% | |

Tabla 5.24. Diferentes escenarios de carga de las distintas CPUs y de tráfico de la red donde se ha ejecutado la rutina LU paralela, en la plataforma TORC.

| | Escenarios de la Plataforma | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *n* | Escenario A | | Escenario B | | Escenario C | | Escenario D | | Escenario E | |
| 1024 | 32 | 4×2 | 32 | 4×2 | 32 | 2×2 | 64 | 2×2 | 64 | 2×1 |
| 2048 | 64 | 4×2 | 64 | 4×2 | 64 | 2×2 | 128 | 2×2 | 128 | 2×1 |
| 3072 | 64 | 4×2 | 64 | 4×2 | 128 | 2×2 | 128 | 2×2 | 128 | 2×1 |

Tabla 5.25. Valores de los *Selected_AP* en los diferentes escenarios de carga de las CPUs de tráfico de la red, para la rutina LU paralela, en la plataforma TORC.

| *n* | *óptimo* | *S-SOLAR* | *D-SOLAR* | dev *S-SOLAR* | dev *D-SOLAR* |
|---|---|---|---|---|---|
| | | | Escenario A | | |
| 1024 | 1.06 | 1.06 | 1.06 | 0 % | 0 % |
| 2048 | 4.86 | 5.60 | 5.60 | 15 % | 15 % |
| 3072 | 13.32 | 13.48 | 13.48 | 1 % | 1 % |
| | | | Escenario B | | |
| 1024 | 1.16 | 1.38 | 1.38 | 19 % | 19 % |
| 2048 | 7.01 | 7.51 | 7.51 | 7 % | 7 % |
| 3072 | 18.46 | 18.46 | 18.46 | 0 % | 0 % |
| | | | Escenario C | | |
| 1024 | 1.30 | 2.54 | 1.36 | 95 % | 5 % |
| 2048 | 7.76 | 12.11 | 7.76 | 56 % | 0 % |
| 3072 | 19.28 | 29.11 | 19.41 | 51 % | 1 % |
| | | | Escenario D | | |
| 1024 | 1.75 | 2.54 | 1.98 | 46 % | 13 % |
| 2048 | 9.67 | 12.11 | 9.87 | 25 % | 2 % |
| 3072 | 26.54 | 29.11 | 27.01 | 10 % | 2 % |
| | | | Escenario E | | |
| 1024 | 1.75 | 2.75 | 2,11 | 57 % | 21 % |
| 2048 | 11.26 | 26.43 | 11.69 | 135 % | 4 % |
| 3072 | 30.22 | 58.76 | 30.22 | 94 % | 0 % |

Tabla 5.26. Comparación del mínimo tiempo experimental (*optimum*), el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo estático (*S-SOLAR*) y el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo dinámico (*D-SOLAR*). Para la rutina LU paralela, en la plataforma TORC.

## Rutina paralela de factorización QR en TORC

De igual manera que se mostraba con la rutina de factorización LU en el anterior apartado, se ha experimentado con diversos escenarios de carga de trabajo heterogénea para la rutina QR, en ocho Pentium III con FastEthernet de la plataforma TORC (Tabla 5.27).

Como ocurría con la LU, cuando aumenta demasiado la carga de trabajo de algunos de los nodos, la *D-SOLAR* puede decidir no utilizarlos (gracias a las predicciones del modelo teórico de la rutina con los valores de los *SP* ajustados a cada situación), pasando a ejecutarse únicamente en aquellos nodos con mayor disponibilidad de cómputo y de comunicaciones. De esta forma, en los escenarios C y D, la *D-SOLAR* decide ejecutarse únicamente en los nodos 5-8, no utilizando los cuatro primero nodos, y en el escenario E se realiza la ejecución únicamente en los nodos 5-6 (Tabla 5.28). Como se muestra en la Tabla 5.29, la *D-SOLAR* ofrece tiempos de ejecución muy cercanos al óptimo para diferente tamaños de problema y en cualquier situación de carga de la plataforma.

| | nodo 1 | nodo 2 | nodo 3 | nodo 4 | nodo 5 | nodo 6 | nodo 7 | nodo 8 |
|---|---|---|---|---|---|---|---|---|
| | Escenario A | | | | | | | |
| Disponibilidad de CPU | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 100% | | | | | | | |
| | Escenario B | | | | | | | |
| Disponibilidad de CPU | 80% | 80 % | 80 % | 80 % | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 80% | | | | 100% | | | |
| | Escenario C | | | | | | | |
| Disponibilidad de CPU | 60 % | 60 % | 60 % | 60 % | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 45% | | | | 100% | | | |
| | Escenario D | | | | | | | |
| Disponibilidad de CPU | 60 % | 60 % | 60 % | 60 % | 100 % | 100 % | 80 % | 80 % |
| Disponibilidad de la red | 45% | | | | 100% | | 80% | |
| | Escenario E | | | | | | | |
| Disponibilidad de CPU | 60 % | 60 % | 60 % | 60 % | 100 % | 100 % | 50 % | 50 % |
| Disponibilidad de la red | 45% | | | | 100% | | 15% | |

Tabla 5.27. Diferentes escenarios de carga de las distintas CPUs y de tráfico de la red donde se ha ejecutado la rutina QR paralela, en la plataforma TORC.

| | Escenarios de la Plataforma | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *n* | Escenario A | | Escenario B | | Escenario C | | Escenario D | | Escenario E | |
| 1024 | 16 | 1×8 | 16 | 1×8 | 16 | 1×4 | 16 | 1×4 | 16 | 1×2 |
| 2048 | 16 | 1×8 | 16 | 1×8 | 16 | 1×4 | 16 | 1×4 | 32 | 1×2 |
| 3072 | 16 | 1×8 | 16 | 1×8 | 32 | 1×4 | 32 | 1×4 | 32 | 1×2 |

Tabla 5.28. Valores de los *Selected_AP* en los diferentes escenarios de carga de las CPUs de tráfico de la red, para la rutina QR paralela, en la plataforma TORC.

| *n* | *óptimo* | *S-SOLAR* | *D-SOLAR* | dev *S-SOLAR* | dev *D-SOLAR* |
|---|---|---|---|---|---|
| | | | Escenario A | | |
| 1024 | 1.94 | 1.94 | 1.94 | 0 % | 0 % |
| 2048 | 11.66 | 11.66 | 11.66 | 0 % | 0 % |
| 3072 | 29.70 | 36.80 | 36.80 | 24 % | 24 % |
| | | | Escenario B | | |
| 1024 | 2.25 | 2.28 | 2.28 | 1 % | 1 % |
| 2048 | 11.90 | 15.80 | 15.80 | 33 % | 33 % |
| 3072 | 31.90 | 37.48 | 37.48 | 17 % | 17 % |
| | | | Escenario C | | |
| 1024 | 4.55 | 6.52 | 5.08 | 43 % | 12 % |
| 2048 | 19.85 | 43.04 | 19.90 | 117 % | 1 % |
| 3072 | 55.12 | 121.63 | 55.12 | 121 % | 0 % |
| | | | Escenario D | | |
| 1024 | 3.56 | 6.52 | 4.50 | 83 % | 26 % |
| 2048 | 26.39 | 43.04 | 28.17 | 63 % | 7 % |
| 3072 | 73.53 | 121.63 | 73.53 | 65 % | 0 % |
| | | | Escenario E | | |
| 1024 | 4.32 | 6.03 | 4.32 | 40 % | 0 % |
| 2048 | 29.07 | 44.33 | 29.60 | 52 % | 2 % |
| 3072 | 81.66 | 115.30 | 102.75 | 41 % | 26 % |

Tabla 5.29. Comparación del mínimo tiempo experimental (optimum), el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo estático (*S-SOLAR*) y el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo dinámico (*D-SOLAR*). Para la rutina QR paralela, en la plataforma TORC.

## Rutina paralela de factorización LU en COCI

Volviendo de nuevo a la rutina de factorización LU, pero ahora para cuatro Sun Ultra 1 de la plataforma COCI, podemos observar cómo la *D-SOLAR* consigue también una buena adaptación. Vemos en la Tabla 5.30 tres escenarios de carga de trabajo utilizados en la fase de experimentación. Como se muestra en la Tabla 5.31, en el escenario A, a pesar de contar con una total disponibilidad de la plataforma se utiliza sólo un procesador para resolver problemas de tamaño pequeño. Esto es debido, como ya se comentó anteriormente, al alto coste de las comunicaciones que tiene esta rutina sobre esta plataforma. Esta situación se ve agravada si aparece algo de carga en algunos nodos de la plataforma (escenario B), donde entonces sólo interesaría (siempre según la *D-SOLAR*) utilizar la versión paralela en 4 nodos para problemas de tamaño a partir de 2048, manteniéndose esta tendencia cuando la carga continúa aumentando (escenario C). También se puede apreciar cómo en ningún momento la *D-SOLAR* decide utilizar 2 ó 3 procesadores, esto es debido a que según el modelo de esta rutina la configuración lógica que tiende a ser la óptima es la de forma más cuadrada posible, con lo que la selección de procesadores pasa directamente de ser 4 procesadores (2×2) a 1.

En la Tabla 5.32 se puede observar lo acertado de las diferentes decisiones que toma la *D-SOLAR* de esta rutina para cualquier carga de trabajo de esta plataforma. Se mantienen siempre unas desviaciones mínimas con respecto a la que sería la decisión óptima (conocida a posteriori), mientras que observamos cómo la versión estática que no tiene en cuenta la situación de la plataforma (*S-SOLAR*) produce unos tiempos de ejecución que se alejan bastante de los deseados cuando la carga del sistema varía heterogéneamente.

| | nodo 1 | nodo 2 | nodo 3 | nodo 4 |
|---|---|---|---|---|
| | | Escenario A | | |
| Disponibilidad de CPU | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 100 % | | | |
| | | Escenario B | | |
| Disponibilidad de CPU | 90 % | 90 % | 100 % | 100 % |
| Disponibilidad de la red | 90 % | | 100 % | |
| | | Escenario C | | |
| Disponibilidad de CPU | 75 % | 75 % | 100 % | 100 % |
| Disponibilidad de la red | 75 % | | 100 % | |

Tabla 5.30. Diferentes escenarios de carga de las distintas CPUs y de tráfico de la red donde se ha ejecutado la rutina LU paralela, en la plataforma COCI.

| $n$ | Escenario A | | Escenario B | | Escenario C | |
|---|---|---|---|---|---|---|
| 512 | 32 | 1×1 | 32 | 1×1 | 32 | 1×1 |
| 1024 | 32 | 1×1 | 32 | 1×1 | 32 | 1×1 |
| 1536 | 16 | 2×2 | 32 | 1×1 | 32 | 1×1 |
| 2048 | 16 | 2×2 | 16 | 2×2 | 32 | 1×1 |

Tabla 5.31. Valores de los *Selected_AP* en los diferentes escenarios de carga de las CPUs de tráfico de la red, para la rutina LU paralela, en la plataforma COCI.

## Rutina paralela de factorización QR en COCI

Por último, en este apartado se va a mostrar el compartimiento de la rutina de factorización QR en cuatro Sun Ultra 1 de la plataforma COCI. En la Tabla 5.33 se muestran diferentes escenarios estudiados, mientras que en la Tabla 5.34 se detallan las decisiones que toma la *D-SOLAR* en cada escenario para diferentes tamaños de problema. Como se puede observar, en el escenario A, para problemas pequeños se utiliza un único procesador. Para el resto de tamaños se utilizarían los cuatro procesadores, ya que el coste de las comunicaciones es inferior a la reducción del tiempo de cómputo que se consigue con la paralelización. Cuando comenzamos a tener carga en dos nodos de la plataforma (escenario B), la *D-SOLAR* sólo cambia la decisión para $n$=1024, donde decide utilizar únicamente los dos procesadores que están libres de carga, pero todavía mantiene el uso de los cuatro para problemas mayores. En el escenario C, donde la carga de los nodos 1-2 es tal que se reduce su disponibilidad de cómputo y comunicación a la mitad, la decisión tomada es dejar de utilizar totalmente estos dos nodos, ejecutándose la rutina en los nodos 3-4 para cualquier tamaño de problema. Finalmente, en el escenario D, los nodos 1-3 tiene tan poca disponibilidad, que la *D-SOLAR* decide prescindir de ellos y ejecutarse en secuencial en el nodo 4.

En la Tabla 5.35 se observa cómo las sucesivas decisiones que toma la *D-SOLAR* le permiten, para esta rutina y esta plataforma, conseguir el tiempo de ejecución óptimo en cualquier situación de disponibilidad de los distintos nodos de la plataforma. La *S-SOLAR* sólo consigue unos buenos tiempos de ejecución cuando la plataforma está libre de carga (como era de esperar), mientras que se aleja de los tiempo de ejecución óptimos conforme la disponibilidad de la plataforma disminuye.

| *n* | *óptimo* | *S-SOLAR* | *D-SOLAR* | dev *S-SOLAR* | dev *D-SOLAR* |
|---|---|---|---|---|---|
| | | | Escenario A | | |
| 512 | 0.89 | 0.89 | 0.89 | 0 % | 0 % |
| 1024 | 7.16 | 7.16 | 7.16 | 0 % | 0 % |
| 1536 | 23.43 | 24.66 | 24.66 | 5 % | 5 % |
| 2048 | 46.19 | 48.39 | 48.39 | 5 % | 5 % |
| | | | Escenario B | | |
| 512 | 0.89 | 0.89 | 0.89 | 0 % | 0 % |
| 1024 | 7.16 | 7.16 | 7.16 | 0 % | 0 % |
| 1536 | 24.16 | 25.93 | 24.16 | 7 % | 0 % |
| 2048 | 49.36 | 50.06 | 50.06 | 1 % | 1 % |
| | | | Escenario C | | |
| 512 | 0.89 | 0.89 | 0.89 | 0 % | 0 % |
| 1024 | 7.16 | 7.16 | 7.16 | 0 % | 0 % |
| 1536 | 24.16 | 45.93 | 24.16 | 90 % | 0 % |
| 2048 | 57.27 | 88.78 | 57.27 | 55 % | 0 % |

Tabla 5.32. Comparación del mínimo tiempo experimental (optimum), el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo estático (*S-SOLAR*) y el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo dinámico que ajusta los valores de sus *SP* de acuerdo a *Runtime_system_information* (*D-SOLAR*). Para la rutina LU paralela, en la plataforma COCI.

| | nodo 1 | nodo 2 | nodo 3 | nodo 4 |
|---|---|---|---|---|
| **Escenario A** | | | | |
| Disponibilidad de CPU | 100 % | 100 % | 100 % | 100 % |
| Disponibilidad de la red | 100 % | | | |
| **Escenario B** | | | | |
| Disponibilidad de CPU | 90 % | 90 % | 100 % | 100 % |
| Disponibilidad de la red | 90 % | | 100 % | |
| **Escenario C** | | | | |
| Disponibilidad de CPU | 50 % | 50 % | 100 % | 100 % |
| Disponibilidad de la red | 50 % | | 100 % | |
| **Escenario D** | | | | |
| Disponibilidad de CPU | 33 % | 33 % | 33 % | 100 % |
| Disponibilidad de la red | 33 % | | | 100 % |

Tabla 5.33. Diferentes escenarios de carga de las distintas CPUs y de tráfico de la red donde se ha ejecutado la rutina QR paralela, en la plataforma COCI.

| | Escenarios de la Plataforma | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *n* | Escenario A | | Escenario B | | Escenario C | | Escenario D | |
| 512 | 16 | 1×1 | 16 | 1×1 | 16 | 1×1 | 16 | 1×1 |
| 1024 | 16 | 1×4 | 16 | 1×2 | 32 | 1×2 | 32 | 1×1 |
| 1536 | 16 | 1×4 | 16 | 1×4 | 32 | 1×2 | 32 | 1×1 |
| 2048 | 16 | 1×4 | 16 | 1×4 | 32 | 1×2 | 32 | 1×1 |
| 2560 | 16 | 1×4 | 16 | 1×4 | 32 | 1×2 | 32 | 1×1 |

Tabla 5.34. Valores de los *Selected_AP* en los diferentes escenarios de carga de las CPUs de tráfico de la red, para la rutina QR paralela, en la plataforma COCI.

| *n* | *óptimo* | *S-SOLAR* | *D-SOLAR* | dev *S-SOLAR* | dev *D-SOLAR* |
|---|---|---|---|---|---|
| | | Escenario A | | | |
| 512 | 2.13 | 2.13 | 2.13 | 0 % | 0 % |
| 1024 | 13.80 | 13.80 | 13.80 | 0 % | 0 % |
| 1536 | 36.90 | 36.90 | 36.90 | 0 % | 0 % |
| 2048 | 74.36 | 74.36 | 74.36 | 0 % | 0 % |
| 2560 | 129.56 | 129.56 | 129.56 | 0 % | 0 % |
| | | Escenario B | | | |
| 512 | 2.13 | 2.13 | 2.13 | 0 % | 0 % |
| 1024 | 15.22 | 18.54 | 15.22 | 22 % | 0 % |
| 1536 | 44.72 | 44.72 | 44.72 | 0 % | 0 % |
| 2048 | 85.27 | 85.27 | 85.27 | 0 % | 0 % |
| 2560 | 151.89 | 151.89 | 151.89 | 0 % | 0 % |
| | | Escenario C | | | |
| 512 | 2.13 | 2.13 | 2.13 | 0 % | 0 % |
| 1024 | 15.22 | 23.13 | 15.22 | 52 % | 0 % |
| 1536 | 46.85 | 60.72 | 46.85 | 30 % | 0 % |
| 2048 | 127.03 | 132.05 | 127.03 | 4 % | 0 % |
| 2560 | 198.85 | 220.29 | 198.85 | 11 % | 0 % |
| | | Escenario D | | | |
| 512 | 2.13 | 2.13 | 2.13 | 0 % | 0 % |
| 1024 | 15.22 | 33.43 | 15.22 | 120 % | 0 % |
| 1536 | 49.14 | 91.74 | 49.14 | 87 % | 0 % |
| 2048 | 145.70 | 178.17 | 145.70 | 22 % | 0 % |
| 2560 | 231.60 | 320.64 | 231.60 | 38 % | 0 % |

Tabla 5.35. Comparación del mínimo tiempo experimental (optimum), el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo estático (*S-SOLAR*) y el tiempo experimental obtenido por los parámetros seleccionados por una *SOLAR* con un modelo dinámico que ajusta los valores de sus *SP* de acuerdo a *Runtime_system_information* (*D-SOLAR*). Para la rutina QR paralela, en la plataforma COCI.

## 5.4 Resumen y Conclusiones

En este capítulo se han descrito dos versiones de nuestra propuesta para resolver el problema de ajustar rutinas de álgebra lineal a las condiciones de la plataforma de ejecución de la manera más óptima posible.

La primera propuesta se centra en plataformas donde la carga de trabajo mantiene unos valores más o menos constantes. En este caso, dada una rutina y una plataforma concreta, la primera y principal labor a realizar consiste en recoger las características estáticas de esta plataforma (capacidad de cómputo de cada procesador y capacidad de comunicación de la red de interconexión) e incorporarlas, gracias a los parámetros del sistema, al modelo analítico del tiempo de ejecución de la rutina. La recogida de los valores de cada parámetro del sistema, que se realiza mediante experimentación con el núcleo de cómputo o de comunicación que corresponda a la operación que define dicho parámetro, se puede llevar a cabo completamente durante la instalación de la rutina. A la hora de ejecutar la rutina, se parte del modelo analítico, del tamaño de problema a resolver y de los valores de los parámetros del sistema ya calculados, y se escogen los valores de los parámetros algoritmos que minimizan el tiempo de ejecución teórico. De esta manera, al no tener que realizar ninguna experimentación en tiempo de ejecución, no se introduce ninguna sobrecarga significativa en el tiempo de ejecución de la rutina.

Para esta primera propuesta, se ha descrito detalladamente la arquitectura software y el ciclo de vida de una rutina que se ajustase a este esquema. Finalmente, se han mostrado resultados experimentales con diferentes rutinas y en plataformas de diversa naturaleza, obteniéndose en todos los casos unos tiempos de ejecución muy cercanos al óptimo.

En una segunda propuesta se ha abordado el caso de plataformas donde la carga de trabajo sufre grandes variaciones en el tiempo y además de manera heterogénea, lo que complica la labor de ajuste del software. El objetivo principal sigue siendo, obviamente, realizar una buena toma de decisiones para los valores de los parámetros algorítmicos, como se hacía en la primera propuesta, pero, ahora, con la necesidad de tener en cuenta la carga de trabajo de la plataforma en el momento de la ejecución de la rutina. Junto a este objetivo principal surge un objetivo colateral, que consiste en minimizar la sobrecarga introducida por este proceso. Para cumplir este segundo objetivo ha sido necesario que la parte más costosa del proceso (experimentación para recoger las características estáticas de la plataforma) se siga realizando durante la instalación, dejando para el momento de la ejecución una labor de mucho menor coste computacional que consiste en obtener estadísticas acerca de la carga de la plataforma (que aporta la herramienta NWS u otra similar) y realizar un ajuste lineal de los valores de los parámetros del sistema en función de esta carga. Tras ello, ya se pueden escoger los valores de los parámetros algorítmicos que minimizan el tiempo de ejecución teórico.

Para esta segunda propuesta, se han detallado las modificaciones y ampliaciones que se necesitarían en la arquitectura software de nuestras rutinas con respecto a la primera versión, así como los cambios en el ciclo de vida de estas rutinas. Finalmente, se han mostrado resultados experimentales con diferentes rutinas en varias plataformas para diversos escenarios de carga de trabajo heterogénea. Las prestaciones obtenidas han sido muy cercanas a las óptimas en cualquiera de las situaciones.

Por último, cabría destacar que los valores de los parámetros algorítmicos que se escogen varían completamente dependiendo de la rutina, y para una misma rutina pueden cambiar según la plataforma donde se encuentre y de su carga de trabajo e, incluso, para distintos tamaños del problema a resolver. Por esta razón, podemos concluir que resulta totalmente necesario un sistema de selección automática de estos valores como el que proponemos, que tenga en consideración todas estas diferentes circunstancias, si nuestro objetivo es obtener buenas prestaciones en cualquier situación.

# Capítulo 6. Propuesta de Optimización Automática de Librerías de Álgebra Lineal

## 6.1 Introducción

En este capítulo se va a ampliar el campo de visión del software de álgebra lineal cuya optimización automática proponemos. Si en el capítulo anterior todos los procesos propuestos partían de una perspectiva a nivel de rutina, en este capítulo el punto de vista se hace más general, al considerar las relaciones existentes entre diferentes librerías. El objetivo general sería justificar la conveniencia de incluir un sistema de información de auto-optimización en la jerarquía clásica de librerías de álgebra lineal (descrita en el capítulo 3). Ahora bien, debido a la amplitud del campo de desarrollo que se propone, debería ser la comunidad de desarrolladores de software de álgebra lineal la que estableciera el estándar general para el formato de esta información de auto-optimización que se incluiría tanto en el software ya existente como en el de nueva creación.

En una primera sección se presenta una propuesta para reducir el grado de experimentación que se realiza durante la instalación de una *SOLAR*. Para ello, partiendo de la jerarquía clásica de librerías de álgebra lineal, se aumenta la relación existente en esta jerarquía, de manera que la información generada empíricamente al instalar las rutinas pertenecientes a librerías de niveles inferiores sea utilizada para la instalación de las rutinas de librerías de niveles superiores, siguiendo el mismo esquema jerárquico que cuando éstas últimas invocan a las primeras en su código.

En la segunda sección introduciremos en nuestra propuesta la posibilidad de seleccionar qué librería básica utilizar para llevar a cabo cada operación dentro de una rutina de nivel superior, escogiendo entre aquellas que tengan funcionalidad semejante y que estén instaladas previamente en la plataforma. Para ello la librería básica a utilizar para cada operación de la rutina se tratará como un parámetro algorítmico más y, por tanto, su valor se seleccionará utilizando el modelo analítico de la rutina, el tamaño de problema a resolver y el valor de los parámetros del sistema.

## 6.2 Jerarquía de librerías auto-optimizadas

Como ya se introdujo en el capítulo tercero donde se describió el entorno de trabajo, el software de álgebra lineal más comúnmente usado se encuentra organizado en forma de jerarquía de librerías (Figura 3.9), de manera que las rutinas pertenecientes a una librería de esta jerarquía cuentan en su código con llamadas a rutinas que pertenecen a librerías de los niveles inferiores. Esta situación se ve reflejada en nuestra propuesta mediante la aparición en el modelo analítico de cada rutina de aquellos parámetros del sistema que corresponden a rutinas de niveles inferiores que son invocadas en su código. Analizando estos modelos, se observa que para las diferentes rutinas de un nivel, sus respectivos conjuntos de rutinas de nivel inferior utilizadas son muy similares. Es decir, de manera general, nos encontramos con una gran reutilización del código entre los diferentes niveles de la jerarquía. Pues bien, en este apartado presentamos una propuesta para unir esta característica de la organización del software de álgebra lineal con el sistema de información de auto-optimización por rutina que hemos propuesto en el capítulo anterior.

La idea general consistiría en incrementar las relaciones existentes entre las diferentes rutinas de la jerárquica de la librerías, de manera que, a la hora de recabar la información de auto-optimización para una rutina durante su fase de instalación, se hará uso de la información de aquellas rutinas de niveles inferiores que utiliza (Figura 6.1). De esta manera, se pretende reducir el proceso de experimentación necesario para medir los parámetros del sistema.

El proceso de instalación se tendría que enfocar ahora desde un punto de vista global de toda la jerarquía de librerías. De manera que a la hora de instalar el software junto a su información de auto-optimización de un nivel dado, se partiría de que ya se ha realizado la instalación en los niveles inferiores; con lo que, conforme se van instalando las librerías de niveles superiores, se utiliza la información ya obtenida para generar la suya propia con el mínimo de experimentación, continuando este proceso hasta la cima de la jerarquía. De igual manera, se puede aplicar este proceso a nuevas rutinas que estuviesen en niveles superiores a los mostrados.

### 6.2.1 Modificaciones en la arquitectura y ciclo de vida de una *SOLAR*

Si analizamos el interior de una *SOLAR*, su arquitectura general no se vería modificada respecto a la descrita en el capítulo anterior. En su ciclo de vida, los cambios aparecerían en el proceso de instalación a la hora de obtener el valor de los *SP* de su *Model*.

Para cada *SOLAR* que pertenezca al nivel inferior de la jerarquía el proceso de instalación sería tal como se describió en el capítulo anterior, es decir, una medición experimental de los *SP* que aparecen en su *Model*. En este proceso, cada $sp_i$*_measurer* obtiene experimentalmente el valor de $sp_i$ para cada combinación de valores de tamaños de problema $\{n_1,...,n_w\}$ y parámetros algorítmicos $\{AP^1,...,AP^z\}$ almacenados en *Preinstallation_information*. Con estas mediciones obtenidas se va rellenando cada componente *Measured_*$sp_i$.

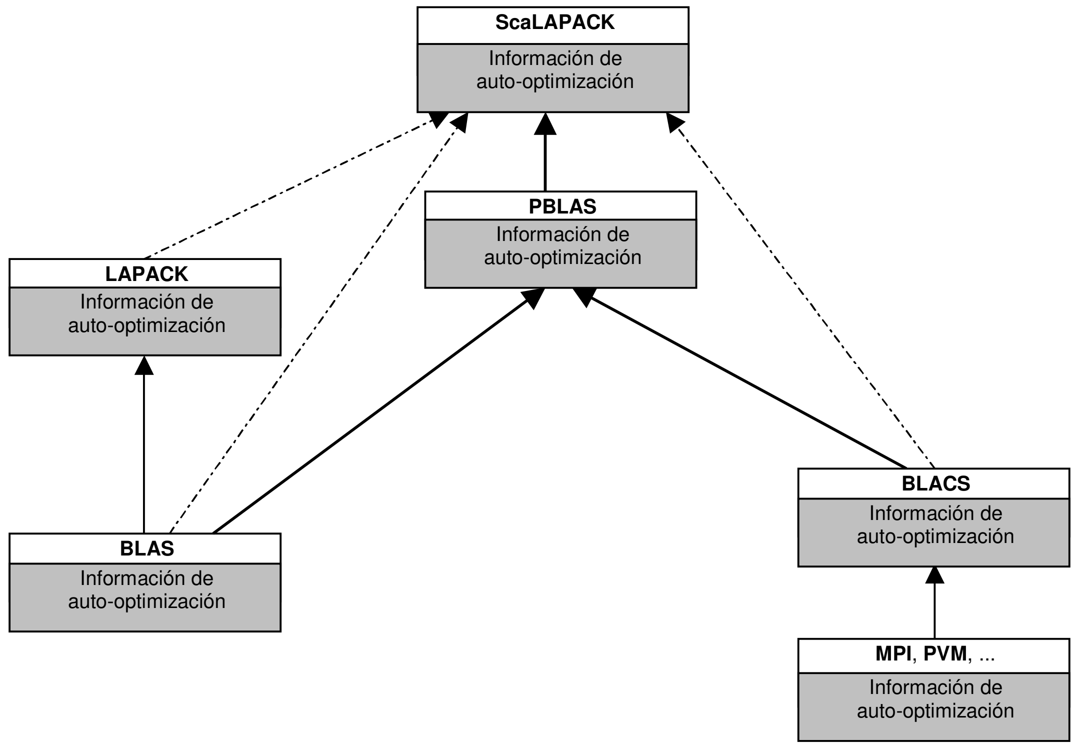


Figura 6.1. Jerarquía de librerías de álgebra lineal bajo ScaLAPACK con capacidad de auto-optimización.

Para las rutinas de los siguientes niveles el proceso de instalación sería diferente. Dada una rutina $S$, para cada uno de los $sp_i$ que aparecen en su *Model* el proceso seguiría el siguiente algoritmo:

1. El *SOLAR_manager*, $M$, se pone en contacto con el *SOLAR_manager* $M_i$ perteneciente a la rutina de nivel inferior $S_i$ a la que corresponde ese parámetro.
2. $M$ informa a $M_i$ de los datos que necesita, es decir, el valor de $sp_i$ para cada combinación de valores de tamaños de problema $\{n_1,...,n_w\}$ y parámetros algorítmicos $\{AP^1,...,AP^z\}$.
3. $M_i$ consulta la información que tiene en su componente *Measured_SP* para comprobar si puede responder a la consulta solicitada desde el nivel superior. Si cuenta con la información requerida por $M$ entonces el proceso continúa por el paso 6. En caso contrario, necesitará obtener esa información.
4. Si $S_i$ pertenece al nivel inferior de la jerarquía, el *SOLAR_manager* pone en marcha el *$sp_i$_measurer* para obtener experimentalmente la información solicitada tal y como se describió anteriormente. El proceso continúa en el paso 6.
5. Si $S_i$ no pertenece al nivel inferior de la jerarquía, la información solicitada no la obtendrá experimentalmente, sino que este proceso de consulta a los niveles inferiores se repetirá ahora para $S_i$. $M_i$ tendrá que ponerse en contacto con el *SOLAR_manager* de cada rutina de nivel inferior que utiliza en su código (como anteriormente $M$ hizo con $M_i$). A la vuelta de este proceso recursivo $M_i$ habrá obtenido la información que necesita.

6. $M_i$ envía a $M$ la información que éste le solicitó.
7. Con la información obtenida, $M$ rellena el componente *Measured_sp*$_i$.

A continuación, se describirá, a modo de ejemplo, cómo se realizaría ahora este proceso durante la instalación de dos rutinas de nivel medio como son las versiones secuenciales de las factorizaciones LU y QR. Como ejemplo de una rutina perteneciente a un nivel alto de la jerarquía se verá el proceso para la rutina *PDGEQRF,* la versión paralela de la factorización QR, que pertenece a la librería ScaLAPACK.

## Instalación de la rutina de factorización LU

Esta rutina pertenece al nivel de LAPACK, con lo que partimos de que ya se encuentra instalada en la plataforma la librería de nivel inferior BLAS, así como las diferentes rutinas auxiliares y las rutinas *drivers* de la propia librería LAPACK. Como ya se mostró anteriormente, esta rutina programada por nosotros, al igual que su equivalente *DGETRF* de la librería LAPACK, utilizaría la rutina auxiliar de LAPACK, *DGETF2* y la rutina *DTRSM*, del nivel 3 de BLAS, y la rutina *DGEMM*, también del nivel 3 de BLAS. De esta manera, en su modelo analítico aparecen los tres *SP* ($k_{3\_DGEMM}$, $k_{3\_DTRSM}$, $k_{2\_DGETF2}$) que corresponden al coste computacional de una operación básica realizada por cada una estas tres rutinas, y cuyos valores dependerán del tamaño de problema, $n$, y del bloque de cálculo que se utilice, $b$, que es el único *ap* de esta rutina.

De esta manera, a la hora de instalar esta rutina, su *SOLAR_manager* se pondría en contacto con los *SOLAR_managers* de esas tres rutinas, solicitándoles el valor de sus respectivos *SP* para las diferentes combinaciones de tamaños de problema $\{n_1,...,n_w\}$ y parámetros algorítmicos $\{b_1,...,b_z\}$ almacenados en *Preinstallation_information* (Figura 6.3).

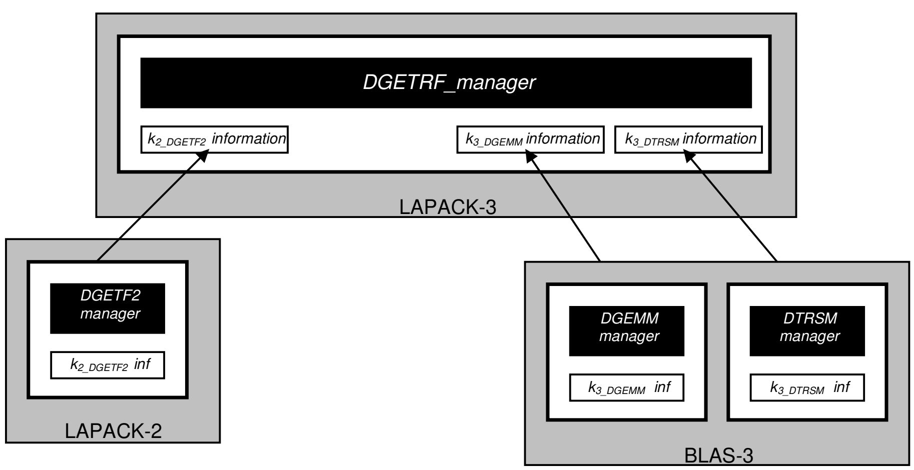


Figura 6.2. Un ejemplo de movimiento de información de optimización: la rutina secuencial de factorización LU.

Con lo que, conforme al algoritmo de esta rutina descrito en el capítulo 4, para cada valor de $n$ y de $b$, las peticiones a los *SOLAR_managers* de los niveles inferiores serían:

- Petición a *DGETF2_manager*: valor de $k_{2\_DGETF2}$ para las dimensiones $b \times b$.

- Petición a *DTRSM_manager*: valor de $k_{3\_DTRSM}$ para las dimensiones $c_i \times b$ y para las dimensiones $b \times c_i$, siendo $c_i = n-b, n-2b, ..., b$.
- Petición a *DGEMM_manager*: valor de $k_{3\_DGEMM}$ para las dimensiones $c_i \times b$ y $b \times c_i$ siendo $c_i = n-b, n-2b, ..., b$.

En estas *SOLARs* de niveles inferiores si no se tiene la información solicitada en sus respectivos *Measured_SP*, se calcularía experimentalmente, ampliando, en cierta manera, el proceso de instalación de estas rutinas básicas.

## Instalación de la rutina secuencial de factorización QR

Para la rutina secuencial de factorización QR, *DGEQRF*, el proceso sería semejante al mostrado anteriormente. Esta rutina de la librería LAPACK invoca en su código a *DGEQR2* y *DLARFT*, que son rutinas auxiliares de la propia librería LAPACK, y a *DGEMM* y *DTRMM*, que pertenecen al nivel 3 de la librería BLAS. Por lo tanto, en su proceso de instalación, su *SOLAR_manager* solicitará la información de auto-optimización necesaria a los respectivos *SOLAR_managers* de estas rutinas (Figura 6.3).

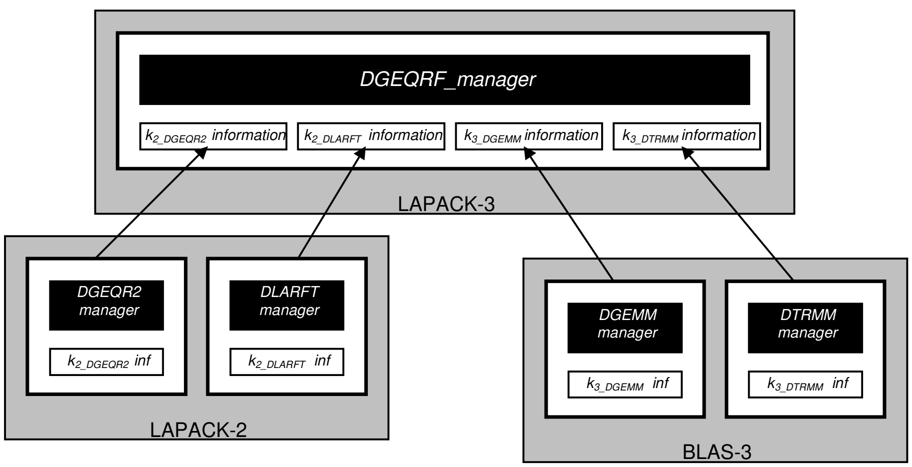


Figura 6.3. Un ejemplo de movimiento de información de optimización: la rutina secuencial de factorización QR.

De esta manera, conforme al algoritmo que sigue esta rutina, descrito en el capítulo 4, para cada combinación de los valores de $n$ y $b$ recogidos en *Preinstallation_information*, las peticiones a los *SOLAR_managers* de los niveles inferiores serían:

- Petición a *DGEQR2_manager*: valor de $k_{2\_DGEQR2}$ para las dimensiones $b \times b$.
- Petición a *DLARFT_manager*: valor de $k_{2\_DLARFT}$ para las dimensiones $c_i \times b$, donde $c_i = n-b, n-2b, ..., b$.
- Petición a *DGEMM_manager*: valor de $k_{3\_DGEMM}$ para las dimensiones $c_i \times b$ y $b \times c_i$ siendo $c_i = n-b, n-2b, ..., b$.
- Petición a *DTRMM_manager*: valor de $k_{3\_DTRMM}$ para las dimensiones $b \times b$.

Observamos que, tal cual como se pretendía, esta rutina y la mostrada anteriormente hacen uso de la misma información que les proporciona el *DGEMM_manager*, evitando tener que realizar la experimentación necesaria cada una de ellas por separado. Esta situación de reutilización de la información de optimización será algo muy común en las rutinas de álgebra lineal, permitiendo una importante reducción de los tiempos totales de instalación de las librerías.

## Instalación de la rutina paralela de factorización QR

La rutina paralela de factorización QR, *PDGEQRF*, pertenece a la librería *ScaLAPACK*. Esta librería, como ya se comentó en el capítulo 3, se sitúa en la cima de la jerarquía de librerías (Figura 3.10) y está formada básicamente por las versiones paralelas de las rutinas que aparecen en la librería LAPACK. En el proceso de instalación de una rutina de ScaLAPACK va a producirse una mayor intercomunicación con diferentes rutinas de los niveles inferiores que en las de LAPACK anteriormente mostradas. Existe dos razones para ello, la primera es que, al ser rutinas más complejas, en su *Model* van a parecer un mayor número de *SP* (como mínimo aparecen ahora los *SP* de comunicaciones); y la segunda es que va a ser frecuente que, para obtener la información necesaria de cada uno de estos *SP*, se descienda recursivamente más de un nivel en la jerarquía de librerías.

La rutina *PDGEQRF* invoca en su código, por un lado, a las rutinas auxiliares *PDGEQR2* y *PDLARFT*, que pertenecen a la propia librería ScaLAPACK, y por otro lado, a *PDGEMM* y *PDTRMM*, que pertenecen a la librería PBLAS. De esta manera, conforme al algoritmo que sigue esta rutina (descrito en el capítulo 4), para cada valor de $n$ y de $b$, las peticiones a los *SOLAR_managers* de los niveles inferiores serían:

- Petición a *PDGEQR2_manager*: valor de $k_{2\_DGEQR2}$ para las dimensiones $b \times b$.
- Petición a *PDLARFT_manager*: valor de $k_{2\_DLARFT}$ para las dimensiones $c_i \times b$, siendo $c_i = n-b, n-2b, \ldots, b$.
- Petición a *PDGEMM_manager*: valor de $k_{3\_DGEMM}$ para las dimensiones $c_i \times b$ y $b \times c_i$, siendo $c_i = n-b, n-2b, \ldots, b$.
- Petición a *PDTRMM_manager*: valor de $k_{3\_DTRMM}$ para las dimensiones $b \times b$.

Si estas rutinas de nivel intermedio no tienen la información requerida para estos *SP* tendrá que continuarse el proceso de solicitud de información hacia los niveles inferiores, hasta que se pueda obtener los valores solicitados para los diferentes *SP* de cómputo. Como podemos apreciar en la Figura 6.4, se reutilizaría la información generada en la instalación (previamente realizada) de la versión secuencial de esta rutina.

En cuanto a los *SP* de comunicaciones, la información sobre estos parámetros se solicitará a las correspondientes rutinas de la librería *BLACS* que, a su vez, podrán pedirla a las rutinas primitivas de comunicación de la versión de MPI o de PVM que se esté utilizando a bajo nivel.

Esta característica de reutilización de la información de auto-optimización se extendería por toda la jerarquía de álgebra lineal de igual forma a como se reutilizan las rutinas de bajo nivel en los códigos de las de niveles superiores. De esta manera el proceso completo de instalación de toda la jerarquía de software con capacidad de ajuste automático, así como de nuevas rutinas de niveles por encima de ella, se agilizaría notablemente.

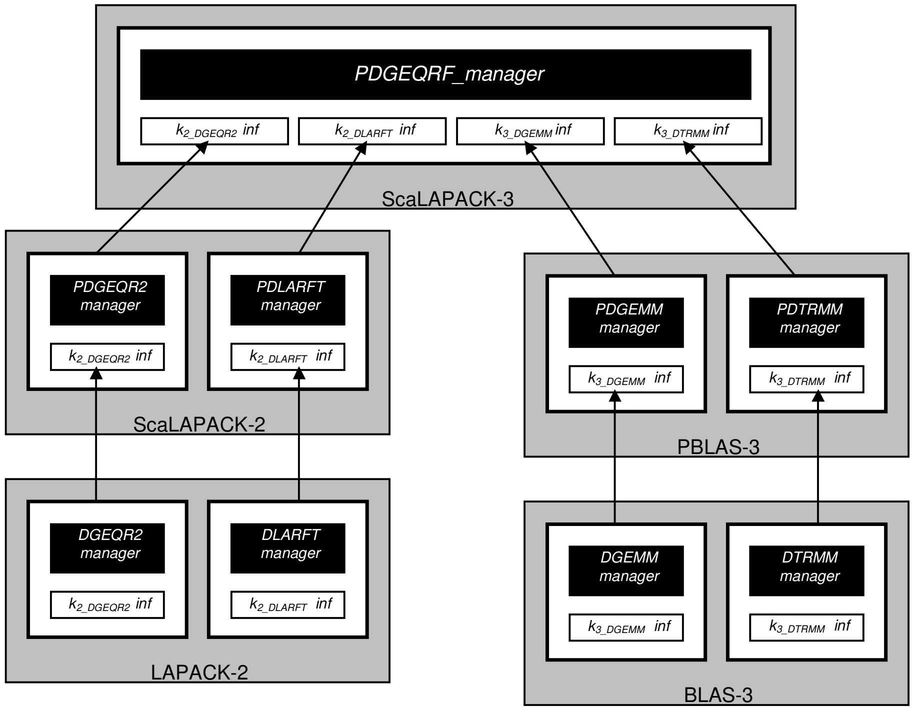


Figura 6.4. Un ejemplo de movimiento de información de optimización: la rutina paralela de factorización QR.

## 6.3 Polilibrerías

Al contar ahora con una visión global a nivel de la jerarquía de librerías de álgebra lineal surge una nueva aplicación de esta metodología de ajuste automático de software, consistente en abrir el abanico de librerías de bajo nivel a utilizar desde las de alto nivel. Algunas motivaciones para la introducción de las polilibrerías en el proceso de ajuste automático del software podrían ser:

- Dada una librería concreta, no para todas las plataformas existe una versión implementada a medida de manera óptima, sino que es muy frecuente encontrar instaladas diferentes versiones genéricas que habrá que evaluar para saber cuál es la más idónea usar en cada caso.
- Cuando las características de la plataforma cambian (añadiendo más memoria, cambiando la red de interconexión, ...) la librería que era la óptima puede dejar de serlo, con lo que será necesario volver a evaluar las diferentes librerías instaladas.
- No se puede afirmar de manera inequívoca que una librería es la óptima para una plataforma pues, como se mostrará experimentalmente, podemos encontrar que la mejor versión para cada rutina se encuentra en distintas librerías. Incluso, para una rutina dada, la

versión óptima puede cambiar dependiendo del tamaño de problema a resolver, del esquema de acceso a los datos, etc.

- En *clusters* heterogéneos que compartan el sistema de ficheros, como la plataforma cSUN que utilizamos en nuestros experimentos, la versión óptima de las rutinas secuenciales puede ser diferente para cada uno de los procesadores que integran esta plataforma.

De esta manera, por ejemplo, una rutina del nivel de LAPACK que requiera llamar a una rutina de multiplicación matricial (nivel de BLAS) podría decidir entre diferentes versiones genéricas de BLAS, u otras optimizadas para la plataforma donde nos encontremos, bien provistas por el vendedor o disponibles libremente [wHen][wGoto], o bien librerías equivalentes como ATLAS, que estén instaladas en la plataforma. De igual manera esta situación se puede dar en cualquier nivel de la jerarquía de librerías (Figura 3.10), reconfigurándose ésta como una nueva jerarquía de polibrerías (Figura 6.5). De esta forma, podemos tener varias versiones de LAPACK, o alguna librería equivalente como ESSL [wESSL] en los sistemas IBM. Esta situación puede darse también para las librerías de comunicaciones, pudiéndose escoger entre versiones de MPI y PVM genéricas o específicas de la plataforma. Incluso con rutinas de alto nivel (resolución de ecuaciones diferenciales [Man03], problemas inversos de valores propios [Alb02], problemas de mínimos cuadrados en matrices Toeplitz [PE00][ABV01], ...), que se encuentran por encima de esta jerarquía, se podrían utilizar rutinas a partir de diferentes combinaciones de librerías de los niveles altos, medios y bajos de la jerarquía. Por ejemplo, se puede resolver un problema de valores propios utilizando ScaLAPACK, LAPACK de la plataforma, ATLAS y el MPI de la plataforma; alternativamente, se podría utilizar PLAPACK, LAPACK de referencia, BLAS de la plataforma y MPICH.

Por otro lado, como se mostrará más adelante, de igual forma que se selecciona de qué librería utilizar una rutina básica (polilibrerías), se puede utilizar la misma metodología para escoger qué algoritmo utilizar entre varios posibles (polialgoritmos) para resolver un problema, e incluso combinar ambas técnicas junto a la selección de los *AP* vistos anteriormente.

Como ya se mostró en el capítulo 2, existen múltiples proyectos que también aplican estas técnicas de polibrerías y/o polialgoritmos (Brewer, SPIRAL, FFTW, SALT, ...). En este campo, nuestra principal aportación consiste en abarcar los diferentes tipos de optimizaciones mostrados (parametrización, distribución del trabajo, polilibrerías y polialgoritmos) de una manera unificada, a partir del modelo analítico del tiempo de ejecución de cada rutina acompañado con la medición experimental de los parámetros del sistema. Con este enfoque unificado, se puede realizar una búsqueda global en el espacio total de factores ajustables de las rutinas.

En este proceso de decisión, la librería de nivel inferior a utilizar se convierte en un nuevo parámetro algorítmico en el *Model* de cada rutina. Este nuevo *ap*, al igual que los vistos anteriormente, se tendrá en cuenta a la hora de medir experimentalmente los *SP*, es decir, como las operaciones de cómputo en cada procesador de una plataforma no tendrán que realizarse expresamente con una única versión de una librería, dada una *operación* de un nivel *i*, el parámetro de sistema $k_{i\_operacion}$ tendrá un valor diferente dependiendo de la librería básica de álgebra linear que se utilice para llevar a cabo la operación. De igual manera el valor de los *SP* de comunicaciones también dependerá de la librería básica de comunicaciones que se decida utilizar.

En el siguiente apartado de esta sección se mostrarán algunos resultados experimentales sobre la gestión de este nuevo parámetro algorítmico, con rutinas de diferentes niveles de la jerarquía y sobre distintas plataformas.

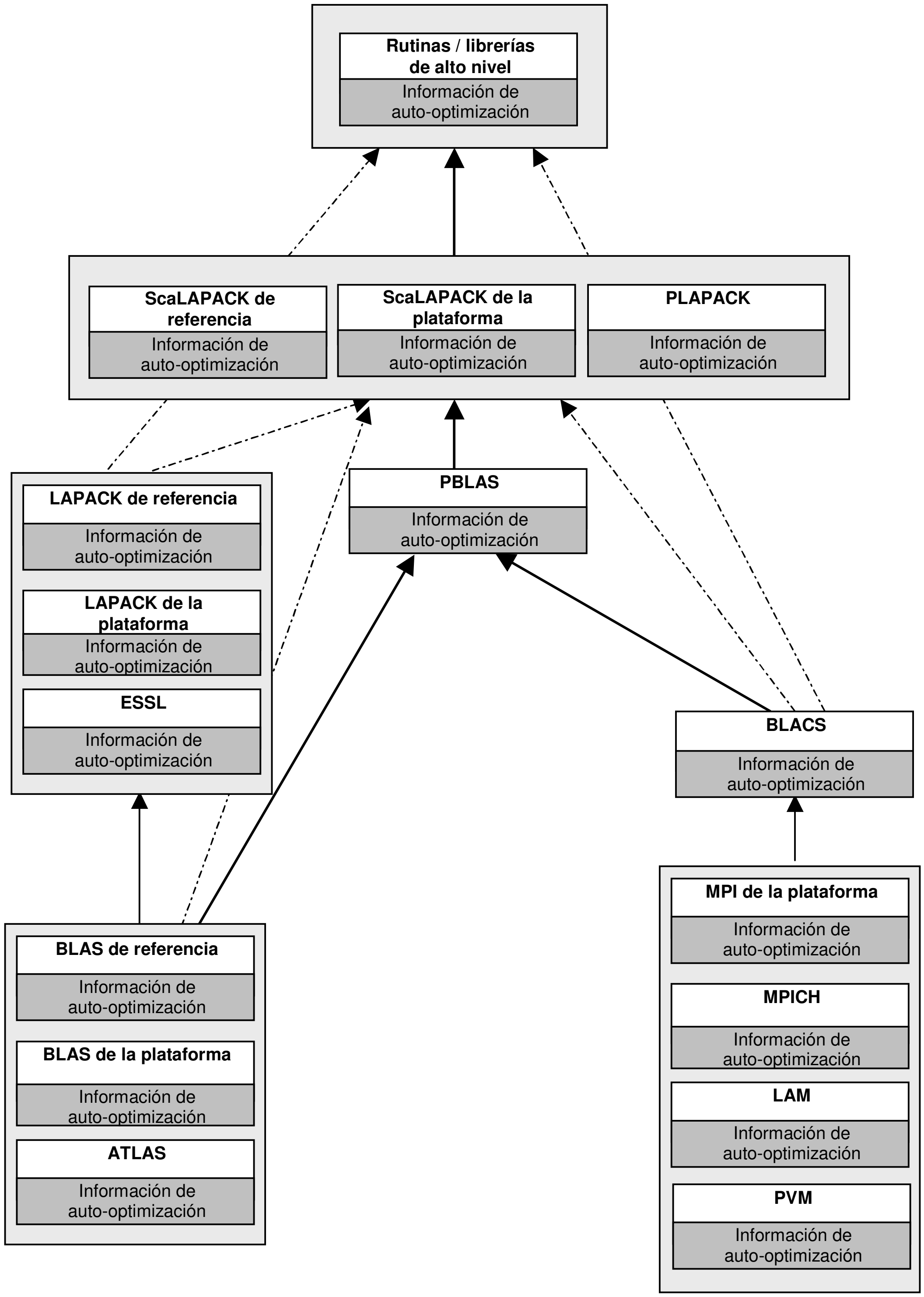


Figura 6.5. Un ejemplo de jerarquía de polilibrerías de álgebra lineal con capacidad de auto-optimización.

### 6.3.1 Resultados experimentales

En este apartado se pretende mostrar, mediante una serie de experimentos, la utilidad de introducir el manejo de polilibrerías en la jerarquía de software de álgebra lineal, resaltando las ventajas que se obtendrían. Igualmente se irán enumerando los problemas que se han encontrado en cada uno de los diferentes niveles de dicha jerarquía.

Las rutinas utilizadas en la experimentación han sido: la multiplicación secuencial de matrices, como un ejemplo de rutina de bajo nivel, las factorizaciones LU y QR, tanto en secuencial como en paralelo, como muestra del funcionamiento en rutinas de medio y alto nivel de la jerarquía, y finalmente, como ejemplo de rutinas de nivel superior a dicha jerarquía de librerías, se han utilizado dos rutinas cuyo comportamiento y modelado también se describieron en el capítulo 4, un algoritmo para resolver un problema de mínimos cuadrados de matrices Toeplitz y el método de "elevación y proyección" para resolver el problema inverso aditivo de valores propios.

#### Multiplicación secuencial matriz-matriz

Para esta rutina de bajo nivel se ha ampliado la funcionalidad de nuestra metodología al introducir como parámetros algorítmicos tanto la selección de la librería a utilizar (polilibrerías) como la elección del algoritmo a seguir (polialgoritmos). Los diferentes algoritmos utilizados son:

- La multiplicación directa, utilizando tal cual la rutina que aporta la librería básica seleccionada.
- Una multiplicación por bloques, que utiliza la rutina de la librería básica para multiplicar los bloques.
- Una multiplicación siguiendo el método de Strassen, que también utiliza la rutina de la librería básica en su implementación.

Según el algoritmo que se seleccione, puede aparecer otro parámetro algorítmico:

- El tamaño de bloque en el algoritmo por bloques.
- El número de subdivisiones de la matriz donde detener la recurrencia en el algoritmo de Strassen.

En la plataforma cSUN, se han considerado las siguientes librerías: un BLAS de referencia genérico (BLAS-ref), la librería científica de Sun (BLAS-maq) y tres versiones de ATLAS, una instalada completamente para Sun Ultra 1 (ATLAS-1) y dos versiones precompiladas de ATLAS 3.2.0 para Ultra 2 (ATLAS-2) y para Ultra 5 (ATLAS-5). Hemos utilizado las mismas librerías para los dos tipos de procesadores que forman este *cluster*, Sun Ultra 1 y Sun Ultra 5, debido a que comparten el sistema de ficheros. En la Tabla 6.1 para Sun Ultra 1 y en la Tabla 6.2 para Sun Ultra 5, se muestra para cada una de estas librerías cuál sería el mejor algoritmo y el *ap* asociado (tamaño de bloque en el algoritmo por bloques o número de subdivisiones de la matriz donde detener la recurrencia para Strassen) junto con el tiempo obtenido. Además se remarca en negrita la casilla correspondiente a la mejor librería para cada tamaño de problema.

El proceso de instalación se ha realizado para los tamaños de problema 400, 800, 1200 y 1600. Además se ha aplicado una metodología heurística de cara a reducir el tiempo de instalación, que supone experimentar para diferentes tamaños de problema con cada una de estas librerías, combinadas con los diferentes algoritmos propuestos y el restante parámetro algorítmico. Para el tamaño de problema más pequeño se ha experimentado con el método directo, con el algoritmo por bloques para los diferentes bloques recogidos en *Preinstallation_information* y de igual forma para el algoritmo de Strassen para diferentes tamaños base (tamaño de la submatriz donde detener la recurrencia) indicados también en *Preinstallation_information*. Pero para el siguiente tamaño de problema, el número de experimentos se ha reducido, de manera que se realiza una búsqueda local de los mejores *AP* en torno a los valores que han sido los óptimos para el tamaño inicial, repitiéndose este proceso para el resto de tamaños de problema.

| | | Tamaño del problema | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 200 | 400 | 600 | 800 | 1000 | 1200 | 1400 |
| | Tiempo | 0.144 | 1.134 | 3.649 | 8.032 | 15.536 | 31.842 | 63.493 |
| BLAS-maq | Método | Strassen | Strassen | Strassen | Strassen | Strassen | Bloques | Bloques |
| | Parámetro | 2 | 4 | 2 | 4 | 8 | 100 | 25 |
| | Tiempo | 0.226 | 1.596 | 5.318 | 11.622 | 23.184 | 53.125 | 90.258 |
| BLAS-ref | Método | Strassen | Strassen | Strassen | Strassen | Strassen | Strassen | Bloques |
| | Parámetro | 4 | 4 | 4 | 8 | 8 | 32 | 25 |
| | Tiempo | 0.089 | 0.704 | 2.236 | 5.201 | 10.450 | 17.861 | 42.990 |
| ATLAS-1 | Método | Directo | Directo | Directo | Directo | Directo | Directo | Bloques |
| | Parámetro | | | | | | | 25 |
| | Tiempo | 0.091 | 0.625 | 1.992 | 4.823 | **9.607** | 15.330 | 39.362 |
| ATLAS-2 | Método | Bloques | Directo | Directo | Strassen | **Strassen** | Directo | Directo |
| | Parámetro | 100 | | | 4 | **4** | | |
| | Tiempo | **0.085** | **0.616** | **1.987** | **4.758** | 9.894 | **15.292** | **24.825** |
| ATLAS-5 | Método | **Directo** | **Directo** | **Directo** | **Strassen** | Strassen | **Directo** | **Directo** |
| | Parámetro | | | | **2** | 8 | | |

Tabla 6.1. Tiempos de ejecución (segundos), algoritmo y parámetro seleccionados para la rutina de multiplicación matricial en una Sun Ultra 1.

| | | Tamaño del problema | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 200 | 400 | 600 | 800 | 1000 | 1200 | 1400 | 1600 |
| | Tiempo | 0.072 | 0.553 | 1.717 | 4.005 | 7.814 | 12.751 | 20.319 | 30.195 |
| BLAS-maq | Método | Directo | Bloques | Strassen | Strassen | Strassen | Strassen | Strassen | Strassen |
| | Parámetro | | 100 | 2 | 4 | 2 | 4 | 4 | 4 |
| | Tiempo | 0.100 | 0.787 | 2.674 | 6.024 | 11.898 | 19.604 | 35.942 | 43.590 |
| BLAS-ref | Método | Strassen | Strassen | Strassen | Strassen | Strassen | Strassen | Strassen | Strassen |
| | Parámetro | 2 | 4 | 16 | 16 | 8 | 32 | 4 | 16 |
| | Tiempo | 0.046 | 0.328 | 1.104 | 2.511 | 4.944 | 8.488 | 13.776 | 20.388 |
| ATLAS-1 | Método | Bloques | Directo | Directo | Directo | Directo | Bloques | Directo | Bloques |
| | Parámetro | 100 | | | | | 400 | | 400 |
| | Tiempo | 0.040 | 0.331 | 1.068 | **2.465** | 4.721 | **7.880** | **12.532** | **20.388** |
| ATLAS-2 | Método | Bloques | Directo | Directo | **Bloques** | Strassen | **Strassen** | **Strassen** | **Bloques** |
| | Parámetro | 100 | | | **400** | 2 | **2** | **2** | **400** |
| | Tiempo | **0.038** | **0.324** | **1.063** | **2.465** | **4.678** | **7.880** | 12.575 | 20.521 |
| ATLAS-5 | Método | **Directo** | **Strassen** | **Directo** | **Bloques** | **Strassen** | **Strassen** | Strassen | Strassen |
| | Parámetro | | **2** | | **400** | **2** | **2** | 2 | 8 |

Tabla 6.2 Tiempos de ejecución (segundos), algoritmo y parámetro seleccionados para la rutina de multiplicación matricial en una Sun Ultra 5.

En la Tabla 6.3, para un procesador Sun Ultra 1, y en la Tabla 6.4, para un procesador Sun Ultra 5, se muestra una comparación, para tamaños de problema distintos a los utilizados en el proceso de instalación, del tiempo obtenido con nuestra metodología frente al tiempo óptimo conocido a posteriori y el que se obtendría utilizando directamente la librería que ofrece mejor resultado de manera general (que se podría escoger gracias a la información obtenida anteriormente). Podemos apreciar, por un lado, que el tiempo de ejecución de la *SOLAR* es muy cercano al óptimo y, por otro, que la ejecución directa de la supuestamente mejor librería no siempre ofrece buenos resultados. Por otro lado, también observamos cómo no siempre resulta trivial etiquetar una librería como la supuestamente óptima para una plataforma. Así, por ejemplo, para la Sun Ultra 1 se podría haber pensado que la mejor librería general iba a ser la proporcionada por el vendedor (BLAS-maq) o, tal vez, ATLAS-1 y, sin embargo, ha resultado ser ATLAS-5.

| | | Tamaño del problema | | | |
|---|---|---|---|---|---|
| | | 200 | 600 | 1000 | 1400 |
| Óptimo | Tiempo | 0.085 | 1.987 | 9.607 | 24.825 |
| | Librería | ATLAS-5 | ATLAS-5 | ATLAS-2 | ATLAS-5 |
| | Algoritmo | Directo | Directo | Strassen | Directo |
| | Parámetro | | | 4 | |
| *SOLAR* | Tiempo | 0.085 | 2.030 | 11.211 | 24.825 |
| | Librería | ATLAS-5 | ATLAS-5 | ATLAS-5 | ATLAS-5 |
| | Algoritmo | Directo | Strassen | Directo | Directo |
| | Parámetro | | 2 | | |
| ATLAS-5 directo | Tiempo | 0.085 | 1.987 | 11.211 | 24.825 |
| ATLAS-1 directo | Tiempo | 0.089 | 2.236 | 10.450 | 42.990 |

Tabla 6.3. Comparación del tiempo óptimo (en segundos), del tiempo *SOLAR* , del tiempo con la mejor librería (conocida a posteriori) usada directamente y del tiempo con la supuestamente mejor librería a priori usada directamente, para la rutina secuencial de multiplicación matricial, con diferentes tamaños de problema y en una Sun Ultra 1.

| | | Tamaño del problema | | | | |
|---|---|---|---|---|---|---|
| | | 200 | 600 | 1000 | 1400 | 1600 |
| Óptimo | Tiempo | 0.038 | 1.063 | 4.678 | 12.532 | 20.031 |
| | Librería | ATLAS-5 | ATLAS-5 | ATLAS-5 | ATLAS-2 | ATLAS-2 |
| | Algoritmo | Directo | Directo | Strassen | Strassen | Bloques |
| | Parámetro | | | 2 | 2 | 400 |
| *SOLAR* | Tiempo | 0.041 | 1.112 | 4.678 | 12.575 | 26.569 |
| | Librería | ATLAS-5 | ATLAS-5 | ATLAS-5 | ATLAS-5 | ATLAS-5 |
| | Algoritmo | Strassen | Bloques | Strassen | Strassen | Strassen |
| | Parámetro | 2 | 400 | 2 | 2 | 2 |
| ATLAS-5 directo | Tiempo | 0.038 | 1.063 | 4.833 | 13.501 | 31.023 |

Tabla 6.4. Comparación del tiempo óptimo (en segundos), del tiempo *SOLAR* y del tiempo con la mejor librería (conocida a posteriori) usada directamente, para la rutina secuencial de multiplicación matricial, con diferentes tamaños de problema y en una Sun Ultra 5.

En un procesador R10000 de la plataforma Karnak, se ha utilizado la librería científica de Silicon Graphipc (BLAS-maq) y dos versiones precompiladas de ATLAS 3.2.0, una para SGI R10000 IP28 con 1MB de caché de segundo nivel (ATLAS-28) y la otra para SGI R10000 IP30 con 1MB de caché de segundo nivel (ATLAS-30). Siguiendo un proceso semejante al de la plataforma cSUN, se ha realizado la instalación para tamaños 400, 800 y 1200, y, posteriormente, se han comparado los tiempos de ejecución para otros tamaños de problema (Tabla 6.5). Observamos cómo la librería científica de esta plataforma es la que mejor resultado ofrece, pero no ejecutándola de manera directa, sino con el método de Strassen. Por su parte, la *SOLAR* consigue una buena selección de librería, algoritmo y parámetro auxiliar en la mayoría de los casos, con tiempos muy cercanos a los óptimos.

| | | Tamaño del problema | | | | |
|---|---|---|---|---|---|---|
| | | 200 | 600 | 1000 | 1400 | 1600 |
| Óptimo | Tiempo | 0.037 | 0.948 | 4.292 | 11.386 | 16.827 |
| | Librería | BLAS-maq | BLAS-maq | BLAS-maq | BLAS-maq | BLAS-maq |
| | Algoritmo | Directo | Strassen | Strassen | Strassen | Strassen |
| | Parámetro | | 2 | 4 | 8 | 8 |
| *SOLAR* | Tiempo | 0.037 | 0.962 | 4.292 | 11.758 | 17.824 |
| | Librería | BLAS-maq | BLAS-maq | BLAS-maq | BLAS-maq | BLAS-maq |
| | Algoritmo | Bloques | Strassen | Strassen | Strassen | Strassen |
| | Parámetro | 100 | 4 | 4 | 4 | 4 |
| BLAS-maq directo | Tiempo | 0.037 | 0.993 | 4.540 | 12.858 | 19.104 |

Tabla 6.5. Comparación del tiempo óptimo (en segundos), del tiempo *SOLAR* y del tiempo con la mejor librería usada directamente, para la rutina secuencial de multiplicación matricial, con diferentes tamaños de problema y en un R10000.

Como se ha podido observar con estos primeros experimentos en una rutina de bajo nivel, la posibilidad de escoger el mejor algoritmo y la mejor librería abre enormemente el abanico de posibilidades para resolver un problema, pudiéndose conseguir mejoras importantes de las prestaciones (alrededor de un 20% en los experimentos realizados) con una buena elección automática de estos nuevos parámetros algorítmicos.

## Rutina secuencial de factorización LU

Como ya se vio en capítulos anteriores, en la rutina de factorización LU la rutina de nivel inferior (nivel de BLAS) con más peso en el tiempo total de ejecución de la LU es la multiplicación matriz-matriz, *DGEMM*. Por esta razón, el estudio experimental de este apartado se ha orientado a la elección automática de la librería de nivel de BLAS de donde utilizar dicha rutina *DGEMM*.

Para la versión secuencial, las plataformas utilizadas han sido un Pentium III a 550 MHz. y un Pentium 4 a 1.5 GHz., de la plataforma TORC. Las librerías utilizadas han sido un BLAS de referencia, dos versiones supuestamente optimizadas para Pentium III y otra para Pentium II, así como un ATLAS precompilado para Pentium II.

De cara a determinar la influencia en el tiempo de ejecución de la librería seleccionada, se muestra el valor de $k_{3\_DGEMM}$ para diferentes valores de tamaño de bloque, *b*, con las tres librerías utilizadas que ofrecen las mejores prestaciones, en un Pentium III (Tabla 6.6) y un Pentium 4 (Tabla 6.7). Tal como se mostró en capítulos anteriores en otros sistemas, el valor de $k_{3\_DGEMM}$ no depende del tamaño del problema.

En la Tabla 6.8 (Pentium III) y en la Tabla 6.9 (Pentium 4) se muestra una comparación experimental entre el tiempo de ejecución óptimo con las librerías consideradas (conocido a posteriori), el tiempo de ejecución de la *SOLAR* (con una *LAR* programada por nosotros) y la rutina *DGETRF* de la librería LAPACK enlazada con cada una de las diferentes librerías de nivel inferior.

Los tiempos óptimos para los diferentes tamaños de problemas aparecen en negrita. Como podemos observar, los *AP* (librería y tamaño de bloque) escogidos por la *SOLAR* son, en casi todos los casos, la elección óptima, mejorando en torno a un 10% sobre la segunda mejor elección de librería. Por otro lado, el mejor tiempo de ejecución usando LAPACK se obtiene igualmente con la mejor librería seleccionada según nuestra metodología (BLAS-III en el Pentium III y BLAS-II en el Pentium 4), lo que viene a mostrar que ésta es igualmente útil para software ya existente.

| | Tamaño de bloque | | | | |
|---|---|---|---|---|---|
| | 16 | 32 | 64 | 128 | 256 |
| ATLAS | 0.0036 | 0.0030 | 0.0028 | 0.0027 | 0.0025 |
| BLAS-III | 0.0039 | 0.0029 | 0.0026 | 0.0023 | 0.0024 |
| BLAS-II | 0.0038 | 0.0029 | 0.0029 | 0.0030 | 0.0034 |

Tabla 6.6. Valores medidos experimentalmente de $k_{3\_DGEMM}$ para diferentes librerías en un Pentium III (en microsegundos).

| | Tamaño de bloque | | | | |
|---|---|---|---|---|---|
| | 16 | 32 | 64 | 128 | 256 |
| ATLAS | 0.0013 | 0.0011 | 0.0011 | 0.0011 | 0.0010 |
| BLAS-III | 0.0017 | 0.0014 | 0.0011 | 0.0011 | 0.0012 |
| BLAS-II | 0.0012 | 0.0010 | 0.0010 | 0.0011 | 0.0013 |

Tabla 6.7. Valores medidos experimentalmente de $k_{3\_DGEMM}$ para diferentes librerías en un Pentium 4 (en microsegundos).

| | *LAR* + Librerías básicas | | | | | | *SOLAR* + Librería básica escogida | | | LAPACK + Librerías básicas | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ATLAS | | BLAS-II | | BLAS-III | | | | | ATLAS | BLAS-II | BLAS-III |
| *n* | Tiempo | *b* | Tiempo | *b* | Tiempo | *b* | Tiempo | Librería | *b* | Tiempo | Tiempo | Tiempo |
| 512 | 0.32 | 64 | **0.28** | 32 | **0.28** | 64 | 0.32 | ATLAS | 64 | 0.37 | 0.35 | 0.33 |
| 1024 | 2.32 | 128 | 2.19 | 64 | **1.96** | 128 | 1.96 | BLAS-III | 128 | 2.45 | 2.44 | 2.23 |
| 1536 | 7.25 | 128 | 7.08 | 32 | **6.15** | 128 | 6.15 | BLAS-III | 128 | 7.83 | 7.82 | 7.11 |
| 2048 | 16.67 | 128 | 16.67 | 32 | **14.04** | 128 | 14.04 | BLAS-III | 128 | 18.02 | 18.14 | 16.29 |

Tabla 6.8. Comparativa de los tiempos de ejecución óptimos (en segundos), los obtenidos con *SOLAR* y los de LAPACK, para la rutina secuencial de factorización LU en un Pentium III.

| | *LAR* + Librerías básicas | | | | | | *SOLAR* + Librería básica escogida | | | LAPACK + Librerías básicas | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ATLAS | | BLAS-II | | BLAS-III | | | | | ATLAS | BLAS-II | BLAS-III |
| *n* | Tiempo | *b* | Tiempo | *b* | Tiempo | *b* | Tiempo | Librería | *b* | Tiempo | Tiempo | Tiempo |
| 512 | 0.11 | 32 | **0.10** | 32 | 0.12 | 64 | 0.10 | BLAS-II | 32 | 0.13 | 0.12 | 0.11 |
| 1024 | 0.86 | 32 | 0.84 | 32 | **0.83** | 128 | 0.84 | BLAS-II | 32 | 0.93 | 0.85 | 0.90 |
| 1536 | 2.83 | 64 | **2.44** | 32 | 2.62 | 128 | 2.44 | BLAS-II | 32 | 2.98 | 2.79 | 3.05 |
| 2048 | 6.54 | 64 | **5.75** | 32 | 6.06 | 128 | 5.75 | BLAS-II | 32 | 6.86 | 6.42 | 7.14 |

Tabla 6.9. Comparativa de los tiempos de ejecución óptimos (en segundos), los obtenidos con *SOLAR* y los de LAPACK, para la rutina secuencial de factorización LU en un Pentium 4.

### Rutina paralela de factorización LU

Para la versión paralela de esta rutina los experimentos se han realizado en cuatro Pentium III unidos con una red Myrinet de la plataforma TORC. Los valores de los *SP* de comunicaciones son $t_s$ = 125 µs y $t_w$ = 0.04 µs, mientras que los valores del principal *sp* de cómputo, $k_{3\_DGEMM}$, corresponden a los mostrados en la Tabla 6.6, pues se utilizan las mismas librerías de cómputo básicas que en el caso secuencial.

En la Tabla 6.10 se muestra, para cada librería y tamaño de problema, el mejor tiempo teórico según el *Model* de esta rutina, el tiempo de ejecución óptimo conocido a posteriori y el tiempo de ejecución con los *AP* elegidos por la *SOLAR*. Se resaltan en negrita los mejores tiempos absolutos para cada tamaño de problema, así como los que se obtendrían con la *SOLAR*.

| | ATLAS | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Teórico | | | | Óptimo | | | | *SOLAR* | | | |
| *n* | Tiempo | *b* | *r* | *c* | Tiempo | *b* | *r* | *c* | Tiempo | *b* | *r* | *c* |
| 512 | 0.13 | 32 | 2 | 2 | 0.12 | 32 | 2 | 2 | 0.12 | 32 | 2 | 2 |
| 1024 | 0.79 | 32 | 2 | 2 | 0.74 | 32 | 2 | 2 | 0.74 | 32 | 2 | 2 |
| 1536 | 2.36 | 32 | 2 | 2 | 2.21 | 64 | 2 | 2 | 2.27 | 32 | 2 | 2 |
| | BLAS-II | | | | | | | | | | | |
| | Teórico | | | | Óptimo | | | | *SOLAR* | | | |
| *n* | Tiempo | *b* | *r* | *c* | Tiempo | *b* | *r* | *c* | Tiempo | *b* | *r* | *c* |
| 512 | 0.13 | 32 | 2 | 2 | **0.11** | 32 | 2 | 2 | **0.11** | 32 | 2 | 2 |
| 1024 | 0.77 | 32 | 2 | 2 | 0.71 | 32 | 2 | 2 | 0.71 | 32 | 2 | 2 |
| 1536 | 2.30 | 32 | 2 | 2 | **2.13** | 32 | 2 | 2 | **2.13** | 32 | 2 | 2 |
| | BLAS-III | | | | | | | | | | | |
| | Teórico | | | | Óptimo | | | | *SOLAR* | | | |
| *n* | Tiempo | *b* | *r* | *c* | Tiempo | *b* | *r* | *c* | Tiempo | *b* | *r* | *c* |
| 512 | 0.13 | 32 | 2 | 2 | **0.11** | 32 | 2 | 2 | **0.11** | 32 | 2 | 2 |
| 1024 | 0.77 | 32 | 2 | 2 | **0.70** | 32 | 2 | 2 | **0.70** | 32 | 2 | 2 |
| 1536 | 2.30 | 32 | 2 | 2 | **2.13** | 32 | 2 | 2 | **2.13** | 32 | 2 | 2 |

Tabla 6.10. Comparativa de los tiempos teóricos (en segundos) con los tiempos de ejecución óptimos y los tiempos de ejecución obtenidos con la *SOLAR*, para la rutina paralela de factorización LU en cuatro Pentium III unidos con Myrinet.

La buena predicción de tiempos que se realiza gracias al *Model*, permite una buena elección de los *AP* (librería, topología lógica de los procesadores y tamaño de bloque). Se puede observar también que las diferencias de prestaciones obtenidas con las diferentes librerías de cómputo son menores que en la versión secuencial, debido a que el tiempo de las comunicaciones es el mismo en los tres casos.

## Rutina paralela de factorización QR

Para la rutina *PDGEQRF,* de la librería ScaLAPACK, se ha realizado un estudio experimental semejante al de la factorización LU, para ocho Pentium III unidos por FastEthernet de la plataforma TORC. Los valores de los *SP* de comunicaciones son, como ya se indicó en el capítulo anterior, $t_s$ = 60 µs y $t_w$ = 0.07 µs, mientras que los valores del principal *sp* de cómputo, $k_{3\_DGEMM}$, corresponden a los mostrados en la Tabla 6.6, pues se utilizan las mismas librerías de cómputo básicas que en la rutina anterior.

En la Tabla 6.11 se muestra la comparación de los *AP* que seleccionaría la *SOLAR* y el tiempo de ejecución que se obtiene, frente a las decisiones óptimas conocidas a posteriori. Se puede observar cómo los valores óptimos para los *AP* (tamaño de bloque, topología lógica y librería básica) varían notablemente según el tamaño de problema a resolver. Pese a esa situación, la *SOLAR* consigue un buen acercamiento a las elecciones óptimas en los diferentes casos.

| | | Tamaño del problema | | | | | |
|---|---|---|---|---|---|---|---|
| | | 128 | 256 | 384 | 1024 | 2048 | 3072 |
| *SOLAR* | Tiempo | 0.025 | 0.087 | 0.178 | 1.92 | 9.90 | 28.00 |
| | *r*×*c* | 1×2 | 1×8 | 1×8 | 1×8 | 1×8 | 1×8 |
| | *b* | 8/16 | 8 | 8 | 8 | 16 | 32 |
| | Librería | BLAS-II | BLAS-II | BLAS-II | BLAS-II | BLAS-II | BLAS-II |
| Óptimo | Tiempo | 0.025 | 0.086 | 0.176 | 1.92 | 9.40 | 25.11 |
| | *r*×*c* | 1×2 | 1×8 | 1x8 | 1×8 | 2×4 | 2×4 |
| | *b* | 16 | 16 | 8 | 8 | 32 | 32 |
| | Librería | BLAS-II | BLAS-III | BLAS-III | BLAS-II | BLAS-II | BLAS-II |

Tabla 6.11. Comparativa de los tiempos de ejecución óptimos (en segundos) y los tiempos de ejecución obtenidos con la *SOLAR*, para la rutina paralela de factorización QR en ocho Pentium III unidos por FastEthernet.

## Problema de mínimos cuadrados de matrices Toeplitz

Como ya se comentó en el capítulo 4, la versión secuencial de esta rutina corresponde a la implementación de Alonso *et al.* [ABV01], basada en el algoritmo propuesto en [PE00]. La versión paralela corresponde con el algoritmo propuesto e implementado por Alonso *et al.* [ABV01]. El coste de esta rutina es $O(mn)$, con $n \leq m$, y las diferentes rutinas básicas que se utilizarían para la implementación de este algoritmo podríamos agruparlas en cinco apartados para facilitar el estudio de su optimización:

- **TRSM**: consiste en la solución de sistemas triangulares de ecuaciones con múltiples vectores como lado derecho. Se utiliza la rutina *DTRSM* del nivel 3 de BLAS. El coste es $O(n^2)$.
- **TRSV**: consiste en la solución de un sistema triangular de ecuaciones, usando la rutina *DTRSV* del nivel 2 de BLAS. El coste es $O(n^2)$.
- **LINEAR**: consiste en aplicar una serie de rutinas de nivel 1: *DDOT*, *DAXPY*, *DSCAL*, *DLARFX* y *DORG2R*. El coste es lineal en $m$ o $n$.
- **ROTATION**: consiste en aplicar diferentes tipos de rotaciones a una matriz, usando las rutinas *DLARFX* (de LAPACK) y *DROT* (del nivel 1 de BLAS). El coste es $O(n^2)$.
- **GTOMV**: realiza una operación matriz-vector. Se utiliza una rutina del estilo *DAXPY* del nivel 1 de BLAS, pero programada a medida.

Observamos que esta rutina de alto nivel invoca en su código a rutinas de los diferentes niveles de la jerarquía de librerías de álgebra lineal. De esta manera la elección de la librería básica a utilizar se puede llevar a cabo para cada una de las diferentes rutinas que se invocan.

Para la versión secuencial, los experimentos se han realizado en un Pentium III y en un Pentium 4 de la plataforma TORC. Las librerías consideradas han sido: ATLAS, un BLAS de referencia (BLAS-ref), un BLAS para Pentium II Xeon (BLAS-II) y dos versiones de BLAS para Pentium III (BLAS-III). Los resultados obtenidos con BLAS-ref son siempre peores que con ATLAS y, por otro lado, la segunda versión de BLAS-III es siempre más rápida que la primera de ellas. Por estas razones, sólo se mostraran resultados comparativos entre ATLAS, BLAS-II y la segunda versión de BLAS-III.

En la Tabla 6.12 (Pentium III) y en la Tabla 6.13 (Pentium 4) se muestran los tiempos de ejecución obtenidos utilizando una única librería de entre las consideradas para toda la rutina, así como los tiempos que se obtienen si se seleccionan diferentes librerías para cada una de las partes (polilibrería óptima).

| *n* | ATLAS | BLAS-II | BLAS-III | Polilibrería óptima |
|---|---|---|---|---|
| 400 | 0.4394 | 0.5602 | 0.5696 | 0.4341 |
| 800 | 0.9840 | 1.2409 | 1.2799 | 0.9756 |
| 1200 | 1.9612 | 2.3592 | 2.4557 | 1.9488 |
| 1600 | 3.1655 | 3.6138 | 3.7845 | 3.1517 |
| 2000 | 4.2811 | 4.8081 | 5.0687 | 4.2560 |

Tabla 6.12. Comparativa de los tiempos de ejecución (en segundos) de la rutina de mínimos cuadrados de Toeplitz, utilizando diferentes rutinas, y el tiempo de ejecución usando la librería óptima en cada parte de la rutina, en un Pentium III, con $m$ =4000 y $n$ variable.

| *n* | ATLAS | BLAS-II | BLAS-III | Polilibrería óptima |
|---|---|---|---|---|
| 400 | 0.1548 | 0.1041 | 0.1180 | 0.1025 |
| 800 | 0.3507 | 0.2427 | 0.3046 | 0.2401 |
| 1200 | 0.5671 | 0.4056 | 0.5496 | 0.4050 |
| 1600 | 0.8341 | 0.6002 | 0.8590 | 0.5999 |
| 2000 | 1.1042 | 0.8156 | 1.2431 | 0.8154 |

Tabla 6.13. Comparativa de los tiempos de ejecución (en segundos) de la rutina de mínimos cuadrados de Toeplitz, utilizando diferentes rutinas, y el tiempo de ejecución usando la librería óptima en cada parte de la rutina, en un Pentium 4, con $m$ =4000 y $n$ variable.

La reducción del tiempo de ejecución que se consigue utilizando la mejor librería para cada parte de la rutina frente a utilizar una única librería es despreciable. Sin embargo, una decisión automática de qué librería usar sigue siendo interesante, porque para diferentes plataformas y/o algoritmos la librería óptima cambia. De esta manera, observamos que para Pentium III la mejor librería es ATLAS, mientras que para Pentium 4 es BLAS-II. Por otro lado, BLAS-II es una librería supuestamente optimizada para Pentium II, y BLAS-III para Pentium III, sin embargo, en los experimentos realizados, BLAS-II suele ofrecer mejores prestaciones que BLAS-III tanto en Pentium-III como en Pentium 4. Esto nos viene a demostrar, una vez más, la dificultad que supone determinar a priori cuál es la mejor librería para cada plataforma, haciendo necesario un sistema automático de decisión como el que proponemos.

De la versión paralela, en la Tabla 6.14 se muestra una comparación de los tiempos de ejecución con estas tres librerías en cuatro Pentium III con Myrinet en la plataforma TORC. Se puede observar que los tiempos óptimos (marcados en negrita) se producen para BLAS-II, mientras que para la versión secuencial la mejor librería es ATLAS. Este cambio de cuál es la mejor librería puede deberse a que en paralelo, para los mismos tamaños de problema, los tamaños de los vectores con los que trabaja cada procesador son más pequeños. De nuevo encontramos un ejemplo de la dificultad de establecer a priori cuál es la mejor librería para una plataforma, pues, como les ocurre a los demás *AP* tratados en este capítulo y en el anterior, la selección del mejor valor para este nuevo *ap* va a depender de varios factores, como las características de la plataforma, el número de procesadores utilizados, el tamaño del problema, la rutina a ejecutar, etc.

| | Número de procesadores utilizados | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| *n, m* | ATLAS | | | |
| 400 | **0.4566** | 0.5870 | 0.5280 | 0.4151 |
| 800 | **1.0109** | 1.4078 | 1.1537 | 1.1023 |
| 1200 | **1.9772** | 2.6694 | 2.3173 | 1.8172 |
| *n, m* | BLAS-II | | | |
| 400 | 0.6329 | **0.5554** | **0.5088** | **0.4073** |
| 800 | 1.3183 | **1.3887** | **1.1412** | **0.9650** |
| 1200 | 2.5290 | 2.5838 | **2.1818** | **1.7185** |
| *n, m* | BLAS-III | | | |
| 400 | 0.6329 | 0.5859 | 0.5180 | 0.4170 |
| 800 | 1.3183 | 1.4244 | 1.1862 | 1.0444 |
| 1200 | 2.5290 | **2.5745** | 2.2229 | 2.0067 |

Tabla 6.14. Comparativa de los tiempos de ejecución (en segundos) de la versión paralela de la rutina de mínimos cuadrados de Toeplitz, para distintas librerías, en una red de Pentium III con Myrinet.

### Método de "Elevación y Proyección"

El método de "elevación y proyección" es uno de los más usados para resolver el problema inverso aditivo de valores propios. En esta rutina, desarrollada por Alberti [Alb02], se utiliza la solución propuesta en [CC96].

Este método de resolución del problema inverso aditivo de valores propios, que se describió detalladamente en el capítulo 4, es otro ejemplo de rutina de alto nivel que invoca en su código a otras rutinas de diversos niveles de la jerarquía de librerías de álgebra lineal. Las diferentes rutinas básicas que se utilizan para la implementación de este algoritmo se pueden agrupar en varios apartados donde se usarán determinadas rutinas de BLAS y LAPACK, mientras que en su versión paralela se utilizarían las rutinas equivalentes de PBLAS y ScaLAPACK.

- **TRACE**:. Este cálculo se realiza solamente una vez, al inicio de la ejecución. Cada traza se calcula realizando *n* productos con la rutina *DOT* y sumando los resultados.
- El resto del algoritmo consiste en una serie de iteraciones hasta alcanzar la convergencia, con lo que el resto de grupos de rutinas se repetirán en cada iteración.
  - **ADK:** Multiplicaciones escalar-matriz y sumas matriz-matriz realizadas mediante las rutinas de nivel 1 *SCAL* y *AXPY*.

- o **EIGEN:** Obtención de una serie de valores y vectores propios usando la rutina *SYEV* de LAPACK.
- o **MATEIG:** Una multiplicación matricial, siendo una de las matrices de forma diagonal.
- o **MATMAT:** Una multiplicación de matrices densas.
- o **ZKAOA:** Formación de una matriz de cálculos intermedios.

Con lo que el coste total de la versión secuencial sería:

$$\begin{aligned} T_{EXEC} = iter\left(\frac{22}{3}k_{SYEV} + 2k_{3\_DGEMM} + k_{3\_DIAGGEMM}\right)n^3 + \\ iter\left(2k_{1\_DOT} + k_{1\_SCAL} + k_{1\_AXPY}\right)n^2L + 2k_{1\_DOT}n^2L^2 + k_{SUM}nL^2 \end{aligned} \qquad 6.1$$

donde aparecen cinco *SP* correspondientes a rutinas de diferentes niveles de la jerarquía (BLAS y LAPACK). Tres de los *SP* de BLAS corresponden a operaciones del subnivel 1 y dos de ellos al subnivel 3.

Las plataformas secuenciales utilizadas han sido: un Pentium III de TORC, una Sun Ultra 1 y una Sun Ultra 5 de COCI, y una R10000 de Karnak.

En la Tabla 6.15 se muestran, para un Pentium III, los tiempos de ejecución obtenidos en total y por cada parte de la rutina, para diferentes combinaciones de librerías utilizadas, para $n$=$m$=250, con $L$=25. Además se destaca el tiempo de ejecución óptimo sin el uso de *threads*, el óptimo usando librerías con *threads* y el óptimo absoluto utilizando la combinación de librerías más apropiada en cada parte de la rutina (en negrita).

| Combinación de librerías | Partes de la rutina | | | | | | TOTAL |
|---|---|---|---|---|---|---|---|
| | TRACE | ADK | EIGEN | MATEIG | MATMAT | ZKA0A | |
| CL_1 | 1.69 | **12.86** | **165.81** | 0.94 | 98.79 | 14.22 | 294.32 |
| CL_2 | 1.16 | 14.87 | 210.85 | 0.83 | 26.70 | 10.46 | **264.89** |
| CL_3 | 1.16 | 15.65 | 255.20 | 0.86 | 10.52 | 10.44 | 293.85 |
| CL_4 | 1.69 | 16.41 | 336.49 | 1.21 | 123.73 | 18.03 | 497.59 |
| Óptima sin *threads* | 1.16 | 12.86 | 165.81 | 0.83 | 10.52 | 10.44 | 201.64 |
| CL_5 | **1.10** | 13.92 | 266.63 | 0.66 | 14.13 | 12.34 | 308.80 |
| CL_6 | 1.16 | 15.68 | 254.34 | 0.79 | **6.66** | **9.99** | 288.66 |
| CL_7 | **1.10** | 13.71 | 249.59 | **0.62** | 13.74 | 11.90 | 290.68 |
| Óptima con *threads* | 1.10 | 13.71 | 249.59 | 0.62 | 6.66 | 9.99 | 281.70 |
| Óptima | 1.10 | 12.86 | 165.81 | 0.62 | 6.66 | 9.99 | 197.06 |

Tabla 6.15. Comparativa de los tiempos de ejecución (en segundos) de las distintas combinaciones de librerías en las diferentes partes de la rutina, en un Pentium III, con $n$ = $m$ =250, $L$ = 25.

Las combinaciones de librerías utilizadas han sido:

- **CL_1**: LAPACK y BLAS instaladas en el sistema.
- **CL_2**: LAPACK de referencia y una versión libre de BLAS, supuestamente optimizada para Pentium III.
- **CL_3**: LAPACK de referencia y una versión libre de BLAS, supuestamente optimizada para Pentium II.

- **CL_4**: LAPACK de referencia y BLAS instalada en el sistema.
- **CL_5**: LAPACK y BLAS instaladas en el sistema, con uso de *threads*.
- **CL_6**: LAPACK de referencia y una versión libre de BLAS, supuestamente optimizada para usar *threads* en un Pentium II.
- **CL_7**: LAPACK de referencia y BLAS instalada en el sistema, supuestamente optimizada para usar *threads*.

De nuevo se observa que la elección directa, sin ninguna experimentación, de la mejor combinación de librerías para toda la rutina (CL_2) no sería una labor sencilla, pues a priori podrían parecer mejores combinaciones la CL_1, que utiliza el software especialmente instalado en la plataforma, así como las diferentes combinaciones que incorporan el uso de *threads* (CL_5, CL_6 y CL_7).

Por otro lado, también se aprecia la importante reducción del tiempo de ejecución (en torno al 30% frente a la mejor, CL_2) si se selecciona una combinación óptima diferente para cada parte de la rutina.

En la Tabla 6.16 (Pentium III), en la Tabla 6.17 (Sun Ultra 1), en la Tabla 6.18 (Sun Ultra 5) y en Tabla 6.19 (R10000) se muestra la mejor combinación de librerías para cada parte de la rutina, con matrices cuadradas y $L=n/10$. Observamos cómo para diferentes plataformas, e incluso para una misma plataforma, al variar el tamaño del problema las combinaciones de librerías óptimas son bien diferentes, lo que justifica un método de selección automático como el propuesto.

| Parte de la rutina | Tamaño del problema | | | | |
|---|---|---|---|---|---|
| | 100 | 150 | 200 | 250 | 300 |
| TRACE | CL_3 | CL_5 | CL_5 | CL_5 | CL_5 |
| ADK | CL_5 | CL_7 | CL_7 | CL_1 | CL_5 |
| EIGEN | CL_5 | CL_7 | CL_1 | CL_1 | CL_1 |
| MATEIG | CL_6 | CL_6 | CL_5 | CL_7 | CL_5 |
| MATMAT | CL_6 | CL_6 | CL_6 | CL_6 | CL_6 |
| ZKA0A | CL_6 | CL_5 | CL_5 | CL_6 | CL_5 |

Tabla 6.16. Combinaciones de librerías seleccionadas para cada parte de la rutina, con matrices cuadradas, $L=n/10$, en un Pentium III.

Si se realiza la instalación de esta rutina experimentando para las tamaños de problema 150 y 250, y ejecutando posteriormente para los tamaños 100, 200 y 300, se obtienen los resultados mostrados en la Tabla 6.20, para un Pentium III, en la Tabla 6.21, para una Sun Ultra 1, en la Tabla 6.22, para una Sun Ultra 5, y en la Tabla 6.23, para un R10000. La comparación que se muestra es entre:

- El tiempo promedio entre los tiempos obtenidos con las combinaciones de librerías consideradas (Promedio).
- El tiempo obtenido usando la mejor combinación de librerías (conocida a posteriori) en toda la rutina (Óptima global).
- El tiempo obtenido con la selección de combinaciones de librerías para cada parte de la rutina que lleva a cabo la *SOLAR*.
- El tiempo obtenido con la mejor combinación de librerías (conocida a posteriori) para cada parte de la rutina (Óptima en cada parte).

| Parte de la rutina | Tamaño del problema | | | | |
|---|---|---|---|---|---|
| | 100 | 150 | 200 | 250 | 300 |
| TRACE | CL_3 | CL_3 | CL_3 | CL_3 | CL_3 |
| ADK | CL_3 | CL_3 | CL_3 | CL_1 | CL_1 |
| EIGEN | CL_3 | CL_3 | CL_3 | CL_3 | CL_1 |
| MATEIG | CL_3 | CL_3 | CL_3 | CL_1 | CL_3 |
| MATMAT | CL_2 | CL_2 | CL_2 | CL_4 | CL_2 |
| ZKA0A | CL_1 | CL_3 | CL_3 | CL_3 | CL_3 |

Tabla 6.17. Combinaciones de librerías seleccionadas para cada parte de la rutina, con matrices cuadradas, $L=n/10$, en una Sun Ultra 1.

| Parte de la rutina | Tamaño del problema | | | | |
|---|---|---|---|---|---|
| | 100 | 150 | 200 | 250 | 300 |
| TRACE | CL_3 | CL_1 | CL_1 | CL_1 | CL_1 |
| ADK | CL_1 | CL_1 | CL_1 | CL_1 | CL_1 |
| EIGEN | CL_1 | CL_1 | CL_1 | CL_1 | CL_1 |
| MATEIG | CL_1 | CL_1 | CL_3 | CL_1 | CL_1 |
| MATMAT | CL_2 | CL_2 | CL_2 | CL_2 | CL_2 |
| ZKA0A | CL_3 | CL_1 | CL_1 | CL_3 | CL_1 |

Tabla 6.18. Combinaciones de librerías seleccionadas para cada parte de la rutina, con matrices cuadradas, $L=n/10$, en una Sun Ultra 5.

| Parte de la rutina | Tamaño del problema | | | | |
|---|---|---|---|---|---|
| | 100 | 150 | 200 | 250 | 300 |
| TRACE | CL_2 | CL_2 | CL_3 | CL_4 | CL_4 |
| ADK | CL_4 | CL_2 | CL_3 | CL_1 | CL_4 |
| EIGEN | CL_3 | CL_3 | CL_3 | CL_3 | CL_3 |
| MATEIG | CL_2 | CL_2 | CL_4 | CL_4 | CL_1 |
| MATMAT | CL_3 | CL_3 | CL_1 | CL_3 | CL_1 |
| ZKA0A | CL_4 | CL_2 | CL_4 | CL_1 | CL_1 |

Tabla 6.19. Combinaciones de librerías seleccionadas para cada parte de la rutina, con matrices cuadradas, $L=n/10$, en un RS10000.

| | Tamaño del problema | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Promedio | 3.57 | 91.30 | 601.95 |
| Óptimo global | 3.06 | 70.34 | 417.10 |
| *SOLAR* | 3.10 | 68.75 | 408.99 |
| Óptima en cada parte | 3.02 | 66.54 | 406.11 |

Tabla 6.20. Comparación del tiempo de ejecución promedio (en segundos), el obtenido con la mejor combinación de librerías, el obtenido con la selección de la *SOLAR* y el obtenido con la óptima combinación en cada parte de la rutina, con matrices cuadradas, $L=n/10$, en un Pentium III.

| | Tamaño del problema | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Promedio | 13.86 | 220.38 | 1352.63 |
| Óptimo global | 11.60 | 185.49 | 1260.08 |
| *SOLAR* | 11.09 | 170.32 | 1159.63 |
| Óptima en cada parte | 11.04 | 169.79 | 1038.75 |

Tabla 6.21. Comparación del tiempo de ejecución promedio (en segundos), el obtenido con la mejor combinación de librerías, el obtenido con la selección de la *SOLAR* y el obtenido con la óptima combinación en cada parte de la rutina, con matrices cuadradas, $L=n/10$, en una Sun Ultra 1.

| | Tamaño del problema | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Promedio | 6.43 | 108.82 | 555.03 |
| Óptimo global | 5.56 | 95.17 | 483.94 |
| *SOLAR* | 5.33 | 87.56 | 426.43 |
| Óptima en cada parte | 5.23 | 87.51 | 426.14 |

Tabla 6.22. Comparación del tiempo de ejecución promedio (en segundos), el obtenido con la mejor combinación de librerías, el obtenido con la selección de la *SOLAR* y el obtenido con la óptima combinación en cada parte de la rutina, con matrices cuadradas, $L=n/10$, en una Sun Ultra 5.

| | Tamaño del problema | | |
|---|---|---|---|
| | 100 | 200 | 300 |
| Promedio | 6.44 | 86.79 | 362.61 |
| Óptimo global | 4.24 | 65.58 | 292.90 |
| *SOLAR* | 4.21 | 65.70 | 292.84 |
| Óptima en cada parte | 4.18 | 65.10 | 291.90 |

Tabla 6.23. Comparación del tiempo de ejecución promedio (en segundos), el obtenido con la mejor combinación de librerías, el obtenido con la selección de la *SOLAR* y el obtenido con la óptima combinación en cada parte de la rutina, con matrices cuadradas, $L=n/10$, en un RS10000.

Se observa que la *SOLAR*, por lo general, mejora considerablemente el tiempo promedio (en torno al 25%), así como al tiempo obtenido usando la mejor combinación de librerías para toda la rutina (en torno al 5%). Además se consigue una buena aproximación al mejor tiempo posible, que usa la combinación óptima en cada parte de la rutina. Esto último viene a demostrar que la *SOLAR* consigue realizar una buena selección de librerías para resolver cada una de las partes de la rutina, llevándose a cabo este proceso con la metodología general de elección de los valores para todos los parámetros algoritmos que aparecen en dicha rutina.

## 6.4 Resumen y Conclusiones

En este capítulo se ha afrontado la optimización automática de software de álgebra lineal desde una visión a nivel de las librerías que lo conforman. De esta manera, se ha mostrado cómo se ampliaría la metodología propuesta en el capítulo anterior.

Así, en una primera sección, con el objetivo de reducir el tiempo de instalación de las diferentes rutinas, se ha incrementado la relación ya existente entre las librerías de diferentes niveles de la jerarquía clásica de librerías de álgebra lineal. La propuesta ha consistido en un reaprovechamiento de la información de auto-optimización generada por las rutinas de los niveles inferiores cuando se instalan en la plataforma, para generar la información de las rutinas de niveles superiores. Este reaprovechamiento de información sigue el mismo patrón que la reutilización de código que viene implícita cuando desde una rutina de alto nivel se invoca a otra de nivel inferior.

Para implementar este proceso ha sido necesario incrementar la funcionalidad del motor de cada rutina, el *SOLAR_manager*, para que éste sea capaz de establecer una "conversación", con el consiguiente intercambio de datos, con los *SOLAR_managers* de las rutinas relacionadas. Se ha detallado, a modo de ejemplo, cómo se llevaría a cabo este proceso para varias rutinas de nivel medio y alto de la jerarquía de librerías.

En una segunda sección, se ha introducido el concepto de polilibrería en nuestra propuesta de ajuste automático. De esta manera, se ha mostrado cómo la elección de qué librería utilizar para llevar a cabo cada operación de una rutina puede suponer una importante mejora de las prestaciones. Esta selección de la librería básica se ha manejado como un nuevo parámetro algorítmico que viene a unirse a los introducidos en el capítulo anterior (tamaño de bloque de cómputo, número de procesadores a utilizar, topología lógica de los procesadores, ...).

Esta ampliación de la propuesta se ha aplicado a rutinas de los diferentes niveles de la jerarquía, desde rutinas básicas, como la multiplicación matricial, rutinas de nivel medio, como las de factorización matricial en secuencial y de nivel alto, como lo son las versiones paralelas de estas factorizaciones. Igualmente se ha mostrado su funcionalidad para rutinas complejas de un nivel superior a las de la jerarquía, que utilizan en su código rutinas de los diferentes niveles de ésta, lo que conlleva un manejo de diferentes combinaciones de librerías a utilizar, para una posterior selección de la óptima.

De igual manera, se ha mostrado cómo se puede utilizar esta misma metodología para seleccionar, además de la librería, el mejor algoritmo con el que resolver un problema dado (polialgoritmos). De esta manera, el algoritmo a utilizar sería tratado como otro nuevo parámetro algorítmico dentro del esquema general de funcionamiento del software de ajuste automático propuesto en este trabajo.

# Capítulo 7. Conclusiones y Trabajo Futuro

## 7.1 Conclusiones

En esta tesis se ha propuesto un entorno de trabajo de software de álgebra lineal capaz de adaptarse automáticamente a las condiciones de la arquitectura paralela donde se esté ejecutando, a fin de minimizar su tiempo de ejecución. Se han abarcado diferentes tipos de optimizaciones (parametrización, distribución del trabajo, polilibrerías y polialgoritmos) de una manera unificada, siendo el modelo analítico del tiempo de ejecución de cada rutina el eje principal que guía toda la metodología.

En primer lugar, se ha mostrado que el modelo teórico-experimental introducido mantiene en todo momento una imagen realista del comportamiento de la rutina sobre la plataforma. Este modelo cuenta con una estructura teórica que define el comportamiento general de la rutina, sobre la que se introducen, en forma de parámetros del sistema, las características de la plataforma, de su software básico y de sus condiciones de carga de trabajo en cada momento, habilitando su utilización de cara a tomar las oportunas decisiones de optimización.

Se ha mostrado cómo en plataformas donde la carga de trabajo no sufre grandes variaciones, el ajuste de la rutina se puede realizar en la fase de instalación, donde se miden experimentalmente los valores de los parámetros del sistema. Posteriormente, a la hora de ejecutar la rutina se puede utilizar el modelo para tomar las decisiones oportunas. Los resultados alcanzados muestran cómo las rutinas se adaptan perfectamente a las características de la plataforma, obteniéndose unas prestaciones cercanas a las óptimas.

En el caso de plataformas donde la carga de trabajo sufre grandes variaciones en el tiempo e incluso de manera heterogénea, la labor de ajuste del software se complica notablemente. En estas condiciones se necesita tener en cuenta la situación de la plataforma en el momento de la ejecución de la rutina, pero sin que el proceso de ajuste introduzca una sobrecarga en el tiempo total de ejecución de las rutinas. Con este objetivo se ha ideado un proceso de ajuste en dos fases, donde la parte más costosa del proceso (recogida de las características estáticas de la plataforma) se realiza durante la instalación, dejando para el momento de la ejecución una labor de mucho menor coste computacional (recoger los datos sobre la carga de la plataforma y realizar un ajuste lineal de los valores de los parámetros del sistema en función de esta carga). Pese a la dificultad que supone el proceso de ajuste en estas condiciones, se han obtenido unas mejoras significativas respecto a las prestaciones promedio de las rutinas, en distintas plataformas y con diferentes situaciones de carga de trabajo heterogénea.

Otro objetivo alcanzado en este trabajo ha sido la integración del sistema de información de auto-optimización que proponemos con la estructura jerárquica de librerías del software de álgebra lineal de uso más extendido en la actualidad (ScaLAPACK, LAPACK, PBLAS, BLACS, BLAS, ...). La propuesta ha consistido en un reaprovechamiento de la información generada por las rutinas pertenecientes a librerías de los niveles inferiores cuando se instalan en la plataforma, para generar la información de las rutinas de niveles superiores, siguiendo el mismo esquema de reutilización ya existente en este software cuando desde una rutina de alto nivel se invoca a otra de nivel inferior.

Se ha mostrado igualmente cómo la elección de qué librería utilizar para llevar a cabo cada operación de una rutina supone una importante mejora de las prestaciones. Esta selección de la librería básica se ha gestionado como un nuevo parámetro algorítmico del modelo. De igual manera, se ha mostrado cómo se puede utilizar esta misma metodología para seleccionar, además de la librería, el mejor algoritmo con el que resolver un problema dado.

Recapitulando, se ha mostrado cómo la metodología propuesta es válida para distintos tipos de plataformas: multiprocesadores de memoria compartida, multiprocesadores de memoria distribuida y conjuntos de máquinas independientes conectadas por una red local formando un *cluster*. Esto ha sido posible gracias a que, aunque estas plataformas tienen características físicas bien diferentes, cualquiera de ellas puede ser vista como una plataforma paralela genérica formada por un conjunto de procesadores que pueden funcionar de manera independiente, donde cada procesador puede acceder localmente a una porción de la memoria total de la plataforma y el conjunto de procesadores de la plataforma puede adquirir diferentes topologías lógicas (malla, anillo, hipercubo, ...). Además, estas plataformas se puede programar, obteniéndose un alto grado de eficiencia, siguiendo el paradigma de paso de mensajes.

Por último, cabría destacar que los valores óptimos de los parámetros algorítmicos pueden ser totalmente diferentes dependiendo de la rutina, y para una misma rutina pueden cambiar según la plataforma donde se encuentre y de su carga de trabajo e incluso para distintos tamaños del problema a resolver. Por esta razón, podemos concluir que resulta totalmente necesario un sistema de selección automática de estos valores como el que proponemos, que tenga en consideración todas estas circunstancias. De esta manera hemos podido alcanzar nuestro objetivo general de que el usuario obtenga las mejores prestaciones posibles con el software científico que maneja, sobre cualquier plataforma y en cualquier situación de carga de trabajo, siendo todo este proceso totalmente transparente para él.

## 7.2 Resultados de la tesis

Los orígenes de esta tesis se remontan al proyecto CICYT "Algoritmos paralelos para problemas densos de valores propios (TIC96/1062-C03-02)", en coordinación con un grupo de investigación de la Universidad Politécnica de Valencia y otro de la Universidad Jaume I de Castellón, dentro del Proyecto conjunto "Algoritmos paralelos para valores propios y ecuaciones en ingeniería", desarrollado desde 1 de Agosto de 1996 hasta el 30 de Septiembre de 1999. En este proyecto, centrado en el desarrollo de algoritmos de álgebra lineal de altas prestaciones, surgió la idea de dotar a este tipo software de capacidad de adaptación automática al entorno.

Con posterioridad, el desarrollo y culminación de los objetivos fundamentales de la tesis se obtuvieron en el marco de los proyectos siguientes:

- Proyecto CICYT "Diseño, Evaluación y Optimización de Algoritmos Paralelos para Problemas Numéricos con Matrices Estructuradas sobre Redes de Computadores (TIC2000-1683-C03-03)", coordinado con un grupo de investigación de la Universidad Politécnica de Valencia, desarrollado desde el 1 de Diciembre de 2000 hasta el 31 de Diciembre de 2003.

- Proyecto de la Fundación Séneca, Consejería de Cultura y Educación de la Región de Murcia, "Automatic Optimization of Parallel Routines (PI-34/00788/FS/01)", desarrollado desde el 1 de Enero de 2002 hasta el 31 de Diciembre de 2003.

- Proyecto CICYT "Desarrollo y Optimización de Código Paralelo para Sistemas de Audio 3D (TIC2003-08238-C02-02)", coordinado con un grupo de investigación de la Universidad Politécnica de Valencia, desarrollado desde el 1 de Diciembre de 2003 hasta el 30 de Noviembre de 2006.

Para la realización de los experimentos de esta tesis se han utilizado diversos recursos hardware y software del Centro de Paralelismo de Barcelona (CEPBA), del Centro de Supercomputación de Cataluña (CESCA) y del *Innovative Computing Laboratory* (ICL) de la Universidad de Tennessee.

Asimismo, merece la pena señalar la realización de tres estancias predoctorales. La primera de ellas fue en el Departamento de Arquitectura de Computadores, de la Universidad Politécnica de Cataluña, tutorizada por el Profesor Miguel Valero-García, durante Octubre-Noviembre de 1999, donde se inició un estudio preliminar de algunas posibles soluciones para modelar el comportamiento de las rutinas paralelas de álgebra lineal. La segunda y tercera estancia se efectuaron en el ICL de la Universidad de Tennessee, tutorizadas por el Profesor Jack Dongarra, durante Octubre-Noviembre de 2001 y Octubre-Noviembre de 2003. En estas estancias se consolidaron algunas de las principales aportaciones de esta tesis. Como resultado de la primera de estos dos estancias en el ICL, se escribió un *technical report* [CG02], así como una publicación para un congreso internacional [CGG+03] en coordinación con miembros del ICL.

A continuación se enumeran las publicaciones que muestran los resultados relacionados con los objetivos marcados en esta tesis, con una breve descripción de su contenido. En primer lugar, en el campo de investigación de desarrollo de algoritmos de álgebra lineal de altas prestaciones, se han publicado los siguientes artículos:

| | |
|---|---|
| Título | **One-sided block Jacobi methods for the Symmetric Eigenvalue Problem** |
| Autores | D. Giménez, J. Cuenca, R. M. Ralha, A. J. Viamonte |
| Tipo de participación | Poster |
| Congreso Científico | VECPAR98, 3rd international meeting on vector and parallel processing |
| Lugar de celebración | Oporto, Portugal, 1998 |
| Publicación | Actas del Congreso |
| Resumen | Partiendo de la base de trabajos realizados con anterioridad [CCG95][CGZ95][GHV+97][CGM98], se propone un nuevo algoritmo de Jacobi unilateral por bloques para el problema simétrico de valores propios. |
| Capítulos de la Tesis | 4 |
| Referencia | [GCR+98] |

| | |
|---|---|
| Título | **Implementation of parallel one-sided block Jacobi methods for the symmetric eigenvalue problem** |
| Autores | J. Cuenca, D. Giménez |
| Tipo de participación | Comunicación |
| Congreso Científico | ParCo99 |
| Lugar de celebración | Delft, Holanda, 1999 |
| Publicación | Capítulo del libro "Parallel Computing Fundamentals & Applications", Imperial College Press, 2000 |
| Resumen | Desarrollo, implementación y medición de prestaciones de dos algoritmos paralelos de Jacobi unilaterales por bloques para el problema simétrico de valores propios. |
| Capítulos de la Tesis | 4 |
| Referencia | [CG00] |
| Comentarios | Artículo también publicado en las X Jornadas de Paralelismo [CG99] |

Por otro lado, las publicaciones enmarcadas dentro del tema central de la tesis, modelado y optimización automática de software de álgebra lineal, serían:

| | |
|---|---|
| Título | **Optimización automática de rutinas paralelas de álgebra lineal** |
| Autores | G. Carrillo, J. Cuenca, D. Giménez, J. González |
| Tipo de participación | Comunicación |
| Congreso Científico | IX Jornadas de Paralelismo |
| Lugar de celebración | Granada, 2000 |
| Publicación | Actas del congreso |
| Resumen | Diversas aproximaciones iniciales a la problemática de la optimización automática, tanto en plataformas homogéneas como heterogéneas |
| Capítulos de la Tesis | 4, 5 |
| Referencia | [CCG+00] |

| | |
|---|---|
| Título | **Modeling the Behaviour of Linear Algebra Algorithms with Message-passing** |
| Autores | J. Cuenca, D. Giménez, J. González |
| Tipo de participación | Comunicación |
| Congreso Científico | 9th EUROMICRO Workshop on Parallel and Distributed Processing (PDP 2001) |
| Lugar de celebración | Mantova, Italia, 2001 |
| Publicación | Actas del congreso, IEEE Press |
| Resumen | Primeros estudios sobre modelado de rutinas. Resultados experimentales con diferentes rutinas de Jacobi para el problema simétrico de valores propios |
| Capítulos de la Tesis | 4 |
| Referencia | [CGG01] |
| Comentarios | Otra versión de este artículo, con una orientación más cercana a la plataforma utilizada, el SGI Origin 2000, sería [CGG00] |

| | |
|---|---|
| Título | **Automatic parametrization of parallel linear algebra routines** |
| Autores | D. Giménez, J. Cuenca, J. González |
| Tipo de participación | Charla invitada |
| Congreso Científico | Algèbre Linéaire et Arithmétique: Calcul Numérique, Symbolique et Parallèle |
| Lugar de celebración | Rabat, Marruecos, 2001 |
| Publicación | Actas del Congreso |
| Resumen | Primeras aproximaciones a una arquitectura software de ajuste automático del software de álgebra lineal |
| Capítulos de la Tesis | 5 |
| Referencia | [GCG01] |

| | |
|---|---|
| Título | **Towards the Design of an Automatically Tuned Linear Algebra Library** |
| Autores | J. Cuenca, D. Giménez, J. González |
| Tipo de participación | Comunicación |
| Congreso Científico | 10th EUROMICRO Workshop on Parallel, Distributed and Networked Processing (PDP 2002) |
| Lugar de celebración | Las Palmas de Gran Canaria, 2002 |
| Publicación | Actas del congreso, IEEE Press |
| Resumen | Primera propuesta completa de un sistema software de ajuste automático de rutinas de álgebra lineal. |
| Capítulos de la Tesis | 4, 5 |
| Referencia | [CGG02a] |

| | |
|---|---|
| Título | **Automatic Optimisation of Parallel Linear Algebra Routines in Systems with Variable Load** |
| Autores | J. Cuenca, D. Giménez, J. González, J. Dongarra, K. Roche |
| Tipo de participación | Comunicación |
| Congreso Científico | 11th EUROMICRO Workshop on Parallel, Distributed and Networked Processing (PDP 2003) |
| Lugar de celebración | Génova, Italia |
| Publicación | Actas del Congreso, IEEE Press |
| Resumen | Modificación de la arquitectura del sistema de información en rutinas auto-ajustables de cara a considerar plataformas con carga de trabajo variable |
| Capítulos de la Tesis | 5 |
| Referencia | [CGG+03] |
| Comentarios | Aunque este congreso se realizó en el 2003, este trabajo se realizó en el año anterior, siendo también publicado en las XIII Jornadas del Paralelismo [CGG+02] |

| | |
|---|---|
| Título | **Architecture of an Automatic Tuned Linear Algebra Library** |
| Autores | J. Cuenca, D. Giménez, J. González |
| Tipo de participación | Póster |
| Congreso Científico | 2nd International Workshop on Parallel Matrix Algorithms and Applications (PMAA'02) |
| Lugar de celebración | Neuchatel, Suiza, 2002 |
| Publicación | Actas del Congreso |
| Resumen | Presentación esquemática de la arquitectura completa del sistema de información en rutinas auto-ajustable, considerando plataformas con carga de trabajo constante y variable. Además, se introduce el sistema de comunicación entre rutinas de diferentes librerías para la optimización común |
| Capítulos de la Tesis | 5, 6 |
| Referencia | [CGG02] |
| Comentarios | Posteriormente, a partir de este póster, se escribió un artículo publicado en Febrero de 2004 en la revista Parallel Computing [CGG04] |

| | |
|---|---|
| Título | **Empirical Modelling of Parallel Linear Algebra Routines** |
| Autores | J. Cuenca, L. P. García, D. Giménez, J. González, A. Vidal |
| Tipo de participación | Póster |
| Congreso Científico | Fifth International Conference on Parallel Processing and Applied Mathematics |
| Lugar de celebración | Czestochowa, Polonia, 2002 |
| Publicación | LNCS, Springer-Verlag (en proceso de publicación) |
| Resumen | Diversas propuestas sobre cómo incorporar las medidas experimentales de los parámetros del sistema en el modelo analítico de las rutinas |
| Capítulos de la Tesis | 4 |
| Referencia | [CGG+03a] |

| | |
|---|---|
| Título | **Designing Polylibraries to Speed Up Linear Algebra Computations** |
| Autores | P. Alberti, P. Alonso, A. Vidal, J. Cuenca, D. Giménez |
| Tipo de participación | Comunicación |
| Congreso Científico | The 2nd Workshop on Hardware/Software Support for High Performance Scientific and Engineering Computing. PACT-SHPSEC-03 |
| Lugar de celebración | Nueva Orleans, EE.UU., 2003 |
| Publicación | En proceso de publicación en la revista "International Journal on High Performance Computing and Networking (IJHPCN)", 2004 |
| Resumen | Introducción de la gestión de polilibrerías en nuestra metodología de ajuste automático de software |
| Capítulos de la Tesis | 6 |
| Referencia | [AAV+03a] |
| Comentarios | Este artículo también se publicó en las XIV Jornadas de Paralelismo [AAV+03b]. Además, se puede encontrar una versión más extendida en el *technical report* [AAV+03] |

| | |
|---|---|
| Título | **Architecture of an Automatic Tuned Linear Algebra Library** |
| Autores | J. Cuenca, D. Giménez, J. González |
| Revista internacional | Parallel Computing, Elsevier,<br>Publicada en Febrero de 2004 |
| Resumen | A partir del póster [CGG02] se escribió este artículo, donde se describe detalladamente la arquitectura completa del sistema de información en rutinas auto-ajustables, considerando plataformas con carga de trabajo constante y variable. Incluye la descripción del sistema de comunicación entre rutinas de diferentes librerías para intercambiar información de optimización |
| Capítulos de la Tesis | 2, 4, 5, 6 |
| Referencia | [CGG04] |

## 7.3 Trabajo futuro

A continuación, presentamos algunas de las líneas relacionadas y complementarias con el trabajo realizado en esta tesis y que consideramos interesantes a corto y medio plazo. Podríamos agrupar estas líneas futuras en dos apartados:

- En un primer grupo tenemos posibles trabajos que consistirían en modificar o ampliar la funcionalidad de algunas partes del sistema software propuesto:
    - Desarrollo de una herramienta para la generación automática del modelo analítico de una rutina. Esta herramienta funcionaría ayudada por el propio diseñador de la rutina,

que le indicaría el esquema del algoritmo, así como las rutinas de nivel inferior que se invocan.

- o Desarrollo de un generador automático de *SOLAR_managers*. Esta herramienta, tomando como punto de partida el esqueleto de un *SOLAR_manager* genérico, lo completaría en función del modelo de la rutina.

- o Introducción en el sistema software de la actualización automática de la información recogida en *Preinstallation_information* a partir de los datos históricos que se vayan recogiendo en las sucesivas ejecuciones de la rutina. De igual manera, se iría ampliando la información recogida en *Measured_SP*. El sistema software se retroalimentaría continuamente con los resultados sobre prestaciones obtenidos gracias a sus propias decisiones, transformándose la fase de instalación en una continúa recalibración durante toda la vida de la rutina.

- Un segundo grupo lo conformarían las líneas de desarrollo que afectan al sistema software en su conjunto (en algunas de la cuáles ya se está empezando a trabajar en la actualidad). Entre ellas destacaríamos:

  - o Implementación, a partir de los prototipos manejados en esta tesis, de una versión de sublibrería de software de álgebra lineal auto-ajustable que fuese totalmente funcional y que estuviera disponible para que cualquier usuario pudiera utilizarla a través de un interfaz adecuado.

  - o Estudio de la aplicabilidad de la metodología descrita a otros tipos de software, como, por ejemplo, rutinas de programación dinámica [GM04]. De igual manera, se podría aplicar esta metodología a esqueletos algorítmicos en lugar de a rutinas concretas.

  - o Ampliación de la metodología para sistemas heterogéneos. En la segunda parte del capítulo 5 se muestra cómo se adapta nuestro sistema a plataformas homogéneas que tienen carga de trabajo heterogénea. Pues bien, el trabajo ahora consistiría, en primer lugar, en generalizar esta aproximación, de cara a considerar también características heterogéneas de los nodos y de las redes de interconexión. Para este primer paso, realmente no sería necesario modificar la metodología de trabajo tal como se presentó, ya que un sistema heterogéneo se puede considerar equivalente a una plataforma homogénea con carga de trabajo heterogénea. La segunda parte de este trabajo, consistiría en incluir en el sistema software la capacidad de gestión necesaria para asignar más de un proceso por procesador.

  - o Ampliación de esta metodología para sistemas distribuidos, para lo que sería necesario el diseño del *middleware* apropiado, al modo en que se está trabajando en LFC [CDL+03].

  - o Ampliación del ciclo de vida de las rutinas, de manera que durante su ejecución se puedan tomar nuevas decisiones sobre los valores de sus parámetros algorítmicos. Esta nueva propuesta sería de utilidad cuando se aplicara a rutinas con un alto tiempo de ejecución por dos razones. En primer lugar, para que dé tiempo a que la situación de la plataforma cambie con respecto al estado de ésta al inicio de la ejecución de la rutina. En segundo lugar, para que el coste de posibles ajustes dinámicos, que pueden incluir redistribución de datos entre los procesadores, sea inferior a la reducción del tiempo de ejecución que se pretende con los nuevos valores de los parámetros algorítmicos.

# Bibliografía


[AAB+96] J. M. Arruabarrena, A. Arruabarrena, R. Beivide and J. A. Gregorio, "Assessing the performance of the new IBM-SP2 communication subsystem", IEEE Concurrency, 4 (4), pp. 12-22, 1996.

[AAV+03] P. Alberti, P. Alonso, A. Vidal, J. Cuenca, L. P. García, D. Giménez, "Designing polylibraries to speed up linear algebra computations", Technical Report UM-LSI 1-2003, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Murcia, Murcia, Spain, 2003.

[AAV+03a] P. Alberti, P. Alonso, A. Vidal, J. Cuenca and D. Giménez, "Designing polylibraries to speed up linear algebra computations", 2nd Workshop on Hardware/Software Support for High Performance Scientific and Engineering Computing, PACT-SHPSEC-03, special issue on International Journal of High Performance Computing and Networking (IJHPCN) (to be published), New Orleans, Louisiana, USA, 2003.

[AAV+03b] P. Alberti, P. Alonso, A. Vidal, J. Cuenca and D. Giménez, "Designing polylibraries to speed up linear algebra computations", XIV Jornadas de Paralelismo, Leganés, Madrid, Spain, pp. 3-8, 2003.

[ABB+90] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mc Kenney and D. Sorensen, "LAPACK: A portable Linear Algebra Library for High-Performance Computers", Tech. Report CS-90-105, (LAPACK Working Note #20), Univ. Of Tennessee, Knoxville, 1990.

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, "LAPACK Users' Guide", Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, Third Edition, 1999.

[ABV01] P. Alonso, J. M. Badía and A. M. Vidal, "A parallel algorithm for solving the Toeplitz least squares problem", Vector and Parallel Processing (VECPAR 2000), J.M.L.M. Palma, J. Dongarra and V. Hernandez, editors, LNCS 1981, Springer-Verlag, pp. 316-329, 2001.

[ACS90] A. Aggarwal, A. K. Chandra and M. Snir, "Communication complexity of PRAMs", Theoretical Computer Science, 71(1), pp. 3-28, 1990.

[AGK+00] B. S. Andersen, F. Gustavson, A. Karaivanov, J. Wasśniewski and P. Y. Yalamov, "LAWRA, Linear algebra with recursive algorithms", on proceedings of PARA 2000, T. Sørevik, F. Manne, R. Moe, and A. H. Gebremedhin, editors, LNCS 1947, Springer-Verlag, Bergen, Norway, pp. 38-50, 2000.

[AGM+03] F. Almeida, D. González, L. M. Moreno, C. Rodríguez and J. Toledo, "On the prediction of master-slave algorithms over heterogeneous clusters", proceedings of the 11th Euromicro Workshop of Parallel, Distributed and Network-based Processing (EUROMICRO-PDP 2003), IEEE Computer Society Press, Genova, Italy, pp. 433-440, 2003.

[AIA+90] H. H. Ammar, S. M. R. Islam, M. Ammar and S. Deng, "Performance Modeling of Parallel Algorithms", International Conference on Parallel Processing, pp. 68-71, 1990.

[AIS+95] A. Alexandrov, M. F. Ionescu, K. E. Schauser and C. Scheiman, "LogGP: Incorporating long messages into the LogP model", proceedings SPAA'95, Santa Barbara, CA USA, pp. 95-105, 1995.

[AKL+98] D. Arapov, A. Kalinov, A. Lastovetsky and I. Dedovskith, "Experiments with mpC: Efficient solving regular problems on heterogeneous networks of computers via irregularization", 5th Internatonial Symposium on Solving Irregularly Structured Problems in Parallel, IRREGULAR'98, LNCS 1457, Berkley, CA, USA, pp.332-343, 1998.

[Alb02] P. Alberti, "Solución paralela de mínimos cuadrados para el problema inverso aditivo de valores propios", Master's thesis, Universidad Politénica de Valencia, Spain, 2002.

[AMM+95] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias and M. Snir, "SP2 system architecture", IBM Systems Journal, 34(2), www.research.ibm.com/journal/sj34-2.html, pp. 152- 184, 1995.

[And00] G. R. Andrews, "Foundations of Multithreaded, Parallel, and Distributed Programming", Addison Wesley Longman, 2000.

[AWG01] B. S. Andersen, J. Waśniewski and F. G. Gustavson, "A recursive formulation of Cholesky factorization of a matrix in packed storage", ACM Transactions on Mathematical Software, 27 (2), pp. 214-244, 2001.

[BAC+97] J. Bilmes, K. Asanovic, C. W. Chin and J. Demmel, "PHiPAC: a portable, high-performance, ANSI C coding methodology", proceedings of International Conference on Computational Science (ICCS), ACM, Vienna, Austria, pp. 340-347, 1997.

[BBP+01] O. Beaumont, V. Boudet, A. Petitet, F. Rastello and Y. Robert, "A proposal for heterogeneous cluster ScaLAPACK (dense linear solver)", IEEE Transactions on Computers, 50 (10), pp. 1052-1070, 2001.

[BBR+01] O. Beaumont, V. Boudet, F. Rastello and Y. Robert, "Matrix multiplication on heterogeneous platforms", IEEE Trans. Parallel Distributed Systems, 12 (10), pp.1033-1051, 2001.

[BCC+01] F. Berman, A. Chien, K. Cooper, J. J. Dongarra, I. Foster, D. Gannon, L. Jonson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon and R. Wolski, "The GrADS project: Software support for high-level grid application development", Rice University, Houston, TX USA, 2001.

[BCC+97] L. S. Blackford, J. Choi, A. Clearly, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker and R. C. Whaley, "ScaLAPACK Users' Guide", Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[BDT01] S. Browne, J. Dongarra and A. Trefethen, "Numerical Libraries and Tools for

Scalable Parallel Cluster Computing", International Journal of High Performance Applications and Supercomputing, 15(2), pp. 175-180, 2001.

[BGL+04] V. Blanco, J. A. González, C. León, C. Rodríguez, G. Rodríguez and M. Printista, “Predicting the performance of parallel programs”, Parallel Computing ((to be published)), Elsevier Science, 2004.

[BGP+94] M. Barnett, S. Gupta, D. Payne, L. Shuler, R. van de Geijn and J. Watts, “Building a high-performance collective communication library”, proceedings of the 1994 conference on Supercomputing, Washington, D.C., USA, pp. 107-116, 1994.

[BK92] A. Bar-Noy and S. Kipnis, “Designing broadcasting algorithms in the postal model for message-passing systems”, 4th Annual Symposium on Parallel Algorithms and Architectures, ACM, pp. 11-22, 1992.

[BL82] R. P. Brent and F. T. Luk, “A systolic architecture for almost linear-time solution of the symmetric eigenvalue problem”, Technical Report TR-CS-82-10, Department of Computer Science, Australian National University, Camberra, 1982.

[BLR+01] O. Beaumont, A. Legrand, F. Rastello and Yves Robert, “Dense linear algebra kernels on heterogeneous platforms: Redistribution issues”, Parallel Computing, 28, pp. 155-185, 2001.

[BPR+99] V. Boudet, A. Petitet, F. Rastello and Y. Robert, “Data allocation strategies for dense linear algebra kernels on heterogeneous two-dimensional grids”, in Parallel and Distributed Computing and Systems conference (PDCS'99), IASTED Press., Marina del Rey, CA, USA, pp. 561-569, 1999.

[Bre94] E. A. Brewer, “Portable high-performance supercomputing: High-level platform-dependent optimization”, PhD Thesis, Departament of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1994.

[Bre95] E.A. Brewer, “High-level Optimization Via Automated Statistical Modeling”, proceedings of Principles and Practice of Parallel Programming, proceedings of the Fifth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPoPP), Santa Barbara, California, pp. 80-91, 1995.

[BRP99] P. B. Bhat, C. S. Raghavendra and V. K. Prasanna, “Efficient collective communication in distributed heterogeneous systems”, 19th IEEE International Conference on Distributed Computing Systems, Austin, Texas, pp. 15-24, 1999.

[BSB+93] P. V. Bangalore, A. Skjellum, C. Baldwin and S. G. Smith, “Dense and iterative concurrent linear algebra in the Multicomputer Toolbox”, proceedings of Scalable Parallel Libraries Conference (SPLC), Starkville, MS, USA, pp. 132-141, 1993.

[BSP+99] M. Banikazemi, J. Sampathkumar, S. Prabhu, D. K. Panda and P. Sadayappan, “Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations”, 8th Heterogeneous Computing Workshop, San Juan, Puerto Rico, pp. 125-134, 1999

[Cam97] D. K. G. Campbell, “A survey of models of parallel computation”, TR YCS-278, Department of Computer Science, University of York, UK, 1997.

[CBL+92] M. Crovella, R. Bianchini, T. LeBlanc, E. Markatos and R. Wisniewski, "Using Communication-to-Computation Ratio in parallel program design and performance prediction", 4th International Symposium on Parallel & Distributed Processing, pp. 238-245, 1992.

[CC96] X. Chen and M. T. Chu, "On the least squares solution of inverse eigenvalue problems", SIAM Journal on Numerical Analysis, 33 (6), pp. 2417-2430, 1996.

[CCG+00] G. Carrillo, J. Cuenca, D. Giménez and J. González, " Optimización automática de rutinas paralelas de álgebra lineal", IX Jornadas de Paralelismo, Granada, Spain, pp. 71-76, 2000.

[CCG95] M. T. Cámara, J. Cuenca and D. Giménez, "Aceleración de la convergencia en métodos de Jacobi para el Problema Simétrico de Valores Propios", IV Congreso de Matemática Aplicada, Vic, Barcelona, Spain, pp. 177-178, 1995.

[CDG00] A. Clematis, G. Dodero and V. Gianuzzi, "Efficient use of parallel libraries on heterogeneous network of worksations", Journal of Systems Architecture, 46, pp. 641-653, 2000.

[CDG01] A. Clematis, G. Dodero and V. Gianuzzi, "A practical approach to efficient use of heterogeneous PC networks for parallel mathematical computation", proceedings of the HPCN'01, Louis O. Hertzberger, Alfons G. Hoekstra, Roy Williams, Editors, LNCS 2110, Springer-Verlag, Amsterdam, pp. 464-473, 2001.

[CDG99] A. Clematis, G. Dodero and V. Gianuzzi, "A resource management tool for heterogeneous networks", proocedings of the 7th Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 1999), IEEE Computer Society Press, Madeira, Portugal, pp. 367-373, 1999.

[CDL+03] Z. Chen, J. Dongarra, P. Luszczek and Kenneth Roche, "Self adapting software for numerical linear algebra and LAPACK for clusters", Parallel Computing, 29 (11-12), Elsevier Science, pp. 1723-1743, 2003.

[CDO+96] J. Choi, J. Dongarra, L. S. Ostrouchov, A. Petitet, D. Walker and R. C. Whaley, "The Design and implementation of the ScaLAPACK LU, QR and Cholesky factorization routines", Scientific Programming, 5, pp. 173-184, 1996.

[CDW92] J. Choi, J. J. Dongarra and D. W. Walker, "Level 3 BLAS for distributed memory concurrent computers", proceedings of Environment and Tools for Parallel Scientific Computing Workshop, Sain Hilaire du Touvet, France, Elsevier Science, pp. 17-29, 1992.

[CG00] J. Cuenca and D. Giménez, "Implementation of parallel one-sided block Jacobi methods for the symmetric eigenvalue problem", Parallel Computing Fundamentals & Applications, Imperial College Press, pp 291-298, 2000.

[CG02] J. Cuenca and D. Giménez, "Some considerations about the Automatic Optimisation of Parallel Linear Algebra Routines", Technical Report UM-LSI 1-2002, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Murcia, Murcia, Spain, 2002.

[CG99] J. Cuenca and D. Giménez, "Implementation of parallel one-sided block Jacobi methods for the symmetric eigenvalue problem", X Jornadas de Paralelismo, Murcia, Spain, pp. 85-87, 1999.

[CGG+02] J. Cuenca, D. Giménez, J. González, K. Roche and J. Dongarra, "Automatic Optimisation of Parallel Linear Algebra Routines in Systems with Variable Load", XIII Jornadas de Paralelismo, Lérida, Spain, pp 129-134, 2002.

[CGG+03] J. Cuenca, D. Giménez, J. González, J. Dongarra and K. Roche, "Automatic optimisation of parallel linear algebra routines in systems with variable load", proceedings of the Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 2003), IEEE Computer Society Press, Genove, Italy, January, pp. 401-408, 2003.

[CGG+03a] J. Cuenca, L. García, D. Giménez, J. González and A. Vidal, "Empirical Modelling of Parallel Linear Algebra Routines", fifth International Conference on Parallel Processing and Applied Mathematics, LNCS Springer-Verlag (to be published), Czestochowa, Poland, 2003.

[CGG00] J. Cuenca, D. Giménez and J. González, "Modelling the behaviour of Linear Algebra Algorithms: a study with Jacobi methods on Origin 2000", SGI Users' Conference, Krakow, Poland, 2000.

[CGG01] J. Cuenca, D. Giménez, and J. González, "Modelling the Behaviour of Linear Algebra Algorithms with Message-Passing", proceedings of the Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 2001), Mantova, Italy, pp. 282-289, 2001.

[CGG02] J. Cuenca, D. Giménez and J. González, "Architecture of an Automatic Tuned Linear Algebra Library", poster on the 2nd International Workshop on Parallel Matrix Algorithms and Applications (PMAA'02), Neuchatel, Switerland, 2002.

[CGG02a] J. Cuenca, D. Giménez and J. González, "Towards the Design of an Automatically tuned Linear Algebra Library", proceedings of the Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 2002), IEEE Computer Society Press, Gran Canaria Island, Spain, pp. 401-408, 2002.

[CGG04] J. Cuenca, D. Giménez and J. González, "Architecture of an Automatic Tuned Linear Algebra Library", Parallel Computing, 30 (2), Elsevier Science, pp. 187-220, 2004.

[CGM98] J. Cuenca, D. Giménez, M. J. Majado, N. Marín and I. Verdú, "LANLAPACK: solving eigenvalue problems on networks of processors", Technical Report UM-LSI 2-98, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Murcia, Murcia, Spain, 1998.

[CGZ95] J. Cuenca, D. Jiménez and J. Zapata, "Uso de núcleos computacionales en el método de Jacobi para el cálculo de valores propios en Multiprocesadores", IV Congreso de Matemática Aplicada, Vic, Barcelona, Spain, pp. 225-226, 1995.

[Cha63] B. A. Chartres, "Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store", the Computer Journal, 5, pp. 52-60, 1963.

[CKP+93] D. Culler, R. Karp, David Patterson, A. Sahay, K.E. Schauser, E. Santos, R.

Subramonian and T. von Eicken, “LogP: Towards a realistic model of parallel computation”, in fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 1-12, 1993.

[CL94] M. E. Crovella and T. J. LeBlanc, “ Parallel performance prediction using Lost Cycles Analysis”, proceedings of Supercomputing ’94, pp. 600-610, 1994.

[CMS03] J. R. McCombs, R. T. Mills and A. Stathopoulos, “Dynamic load balancing of an iterative eigensolver on grids of heterogeneous clusters”, in International Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France, 2003.

[CQ93] M. J. Clement and M. J. Quinn, “Analytical Performance Prediction on Multicomputers”, proceedings of Supercomputing ’93, Portland, Oregon, USA, pp. 886-894, 1993.

[Cro94] M. E. Crovella, “Performance Prediction and Tuning of Parallel Programs”, PhD Thesis, Department of Computer Science, University of Rochester, NY USA, 1994.

[CS03] J. R. McCombs and A. Stathopoulos, “Parallel, multigrain iterative solvers for hiding network latencies on MPPs and neetworks of clusters”, Parallel Computing, 29 (9), Elsevier Science, pp. 1237-1259, 2003.

[CZ89] R. Cole and O. Zajicek, “The APRAM: Incorporating asynchrony into the PRAM model”, proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures, pp. 169-178, 1989.

[DDD+90] J. Dongarra, J. Du Croz, I. S. Duff and S. Hammarling, “A set of Level 3 Basic Linear Algebra Subprograms”, ACM Transactions on Mathematical Software, 16(1), pp. 1-17, 1990.

[DDH+88] J. J. Dongarra, J. Du Croz, S. Hammarling and R. Hanson, “An extended set of Fortran basic linear algebra sobroutines”, ACM Transactions on Mathematical Software, 14(1), pp. 1-17, 1988.

[DDS+98] J. J. Dongarra, I. S. Duff, D. C. Sorensen and H. A. van de Vorst, “Numerical Linear Algebra for High-Performance Computers”, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.

[DE02] J. J. Dongarra and V. Eijkhout, “Self-adapting numerical software for next generation applications”, ICL Technical Repot, ICL-UT-02-07, 2002.

[Dem89] J. Demmel, “LAPACK: A portable linear algebra library for supercomputers”, proceedings of the 1989 IEEE Control System Society Workshop on Computer-Aided Control System Design, 1989.

[DGK+01] P. A. Dinda, T. Gross, R. Karrer, B. Lowekamp, N. Miller, P. Steenkiste and D. Sutherland, “The architecture of the Remos system”, 10th IEEE Symp. on High Performance Distributed Computing, pp. 383-394, 2001.

[DK96] K. Dackland, and B. Kågström, “A Hierarchical Approach for Performance Analysis of ScaLAPACK-based Routines Using the Distributed Linear Algebra Machine”, Applied Parallel Computing: Computations in Physics, Chemistry and Engineering Science, in Dongarra et. al. editors, LNCS 1184, Springer-Verlag, pp.

186-195, 1996.

[DLO03] E. M. Daoudi, A. Lakhouaja, H. Outada, “Overlapping Computation/Communication in the Parallel One-Sided Jacobi Method”, proceedings of EUROPAR 03, LNCS 2790, Springer-Verlag, Klagenfurt, Austria, pp. 844-849, 2003.

[dlTK91] P. de la Torre and C. P. Kruskal, “Towards a single model of efficient computation in real parallel machines”, Parallel Architectures and Languages Europe (PARLE ’91), LNCS, Springer-Verlag, 1991.

[dlTK96] P. de la Torre and C. P. Kruskal, “Submachine locality in the bulk synchronous setting”, proceedings of EUROPAR 96, LNCS 1124, Springer-Verlag, pp. 352-358, 1996.

[DM98] L. Dagum and R. Menon, “OpenMP: An industry-standard API for shared-memory programming”, IEEE Computational Science & Engineering, 5 (1), pp. 46-54, 1998

[DMP02] F. Dehne, S. Mardegan and G. Prencipe, “Distribution sweeping on clustered machines with hierarchical memories”, proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), 2002.

[DV92] J. Demmel and K. Veselić, “Jacobi method is more accurate than QR”, SIAM J. Matrix Anal. Appl. 13, pp. 1204-1245, 1992.

[DW95] J. Dongarra and R.C. Whaley, “ A users guide to BLACS v1.0”, TR UT CS-95-281, LAPACK working note 94, University of Tennessee, USA, 1995.

[DW95a] J. Dongarra and D. Walker, “Software libraries for linear algebra computations on high performance computers”, SIAM review, 37 (2), pp. 151-180, 1995.

[EG00] E. Elmroth and F. Gustvason, “A high performance algorithm for the linear least squares problem on SMP system”, proceedings of the PARA 2000, T. Sørevik, F. Manne, R. Moe, and A.H. Gebremedhin, editors, Bergen, Norway, LNCS 1947, Springer-Verlag, pp. 53-63, 2000.

[EP90] P. J. Eberlein and Haesun Park, “Efficient implementation of Jacobi algorithms and Jacobi sets on distributed memory architectures”, Journal of Parallel and Distributed Computing 8, pp. 358-366, 1990.

[FFF+01] C. Fabianek, F. Franchetti, M. Frigo, H. Karner, L. Meirer, C. W. Ueberhuber, “Survey of self-adapting FFT software”, AURORA TR2002-01, Institute for Applied and Numerical Mathematics, Technical University of Vienna, Austria, 2002.

[FH60] G. E. Forsythe and P. Henrici, “The cyclic Jacobi method for computing the principal values of a complex matrix”, Trans. Amer. Math. Soc. 94, pp. 1-23, 1960.

[FJ97] M. Frigo and S. G. Johnson, “The fastest Fourier transform in the west”, Tech. Report MIT-LCS-TR-728, MIT Laboratory for Computer Science, Cambridge, MA, USA, 1997.

[FJ98] M. Frigo and S. G. Johnson, “FFTW: An adaptive software architecture for the FFT”, proceedings of the ICASSP Conference, IEEE Computer Society Press, pp.

1381-1384, 1998.

[FK01] M. Frigo and S. Kral, "The advance FFT program generator GENFFT", Tech. Report AURORA TR2001-03, Vienna University of Technology, Vienna, Austria, 2001.

[FK89] H. P. Flatt and K. Kennedy, "Performance of parallel processors", Parallel Computing, Elsevier Science Publishers, 12, pp. 1-12, 1989.

[Fly66] M. Flynn, "Very high-speed computing system", proceedings of the IEEE, 54, pp. 1901-1909, 1966.

[Fos95] I. Foster, "Designing and building parallel program", www-unix.mcs.anl.gov/dbpp/text/book.html, Addison-Wesley, 1995.

[FW78] S. Fortune and J. Wyllie, "Parallelism in random access machines", proceedings of the 10th Symposium on Theory of Computing. ACM, pp. 114-118, 1978.

[GAM+03] D. González, F. Almeida, L. Moreno and C. Rodríguez, "Towards the automatic optimal mapping of pipeline algorithms", Parallel Computing, 29 (2), Elsevier Science, pp. 241-254, 2003.

[GBD+96] A. Geist, A. Beguelín, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, "PVM: Parallel Virtual Machine - a users' guide and tutorial for networked parallel computing", 3rd Edition, Massachusetts Institute of Technology, 1996.

[GCG01] D. Giménez, J. Cuenca and J. González, "Automatic parameterization of parallel linear algebra routines", Algèbre Linéaire et Arithmétique: Calcul Numérique, Symbolique et Parallèle, Rabat, Morroco, pp. 63-81, 2001.

[GCR+98] D. Giménez, J. Cuenca, R. M. Ralha and A. J. Viamonte, "One Sided Block Jacobi Methods for the Symmetric Eigenvalue Problem", proceedings of the Third International Meeting on Vector and Parallel Processing (VECPAR 98), Porto, Portugal, pp. 687-692, 1998.

[GGK+03] A. Grama, A. Gupta, G. Karypis and V. Kumar, "Introduction to parallel computing", 2nd edition, Addison Wesley Longman, 2003.

[GH01] S. Goedecker and A. Hoisie, "Performance optimization of numerical intensive codes", in Dongarra et. al. editors, Siam Philadelphia, Software-Enviroment-Tools series, 2001.

[GHH97] V. Getov, E. Hernández and T. Hey, "Message-passing performance of parallel computers", TR-CSPE-04, Department of Electronics and Computer Science, University of Westminster, London UK, 1997.

[GHJ+98] F. Gustvason, A. Henriksson, I. Jonsson, B. Kågström and P. Ling, "Recursive blocked data formats and BLAS's for dense linear algebra algorithms", in proceedings of the PARA'98, Umeå, Sweden, LNCS 1541, Springer-Verlag, pp. 195-206, 1998.

[GHJ+98a] F. Gustvason, A. Henriksson, I. Jonsson, B. Kågström and P. Ling, "Superscalar GEMM-based level 3 BLAS – The ongoing evolution of portable and high-performance library", in proceedings of the PARA'98, Umeå, Sweden, LNCS 1541, Springer-Verlag, pp. 207-215, 1998.

[GHV+97] D. Giménez, V. Hernández, R. van de Geijn and A. M.Vidal. "A block Jacobi method on a mesh of processors", Concurrency: Practice and Experience 9(5), pp. 391-411, 1997.

[GHvG01] J. Gunnels, G. Henry and R. van de Geijn, "A family of high-performance matrix algorithms, " in Computational Science - 2001, Part I, LNCS 2073, Springer-Verlag, pp. 51-60, 2001.

[Gim95] D. Giménez, "Métodos de Jacobi para la resolución del problema simétrico de valores propios en multicomputadores", PhD Thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 1995.

[GJ00] F. Gustavson and I. Jonsson, "High performance Cholesky factorization via blocking and recursion that uses minimal storage", Workshop on Applied Parallel Computing (PARA 2000), T. Sørevik, F. Manne, R. Moe, and A. H. Gebremedhin, editors, LNCS 1947, Springer-Verlag, pp 82-102, 2000.

[GJM+91] K. Gallivan, W. Jalby, A. Malony and H. Wishoff, "Performance prediction for parallel numerical algorithms", International Journal of High Speed Computing, World Scientific Publishing Company, 1(3), pp. 31-62, 1991.

[GKU99] W. N. Gansterer, D. F. Kvasnicka and C. W. Ueberhuber, "Blocking techniques in numerical software", ACPC'99, in Zinterhof, Vajtersic, Uhl Editors, LNCS 1557, Springer-Verlag, pp. 127-139, 1999.

[GL97] W. D. Gropp and E. Lusk, "Why are PVM and MPI so different?", in Recent Advances in Parallel Virtual Machine and Message Passing Interface", LNCS, 1332, Springer-Verlag, pp. 3-10, 1997.

[GLP+99] J. A. González, C. León, F. Piccoli, M. Printista, J. L. Roda, C. Rodríguez and F. Sande, "Oblivious BSP", Euro-par 2000, LNCS 1900, Springer-Verlag, pp. 682-685, 2000.

[GLR+02] J. A. González, C. León, J. L. Roda, C. Rodríguez, J. M. Rodríguez, F. Sande and M. Printista, "Model oriented profiling of parallel programs", proceedings of the 10th Euromicro Workshop of Parallel, Distributed and Network-based Processing (EUROMICRO-PDP 2002), IEEE Computer Society Press, Canary Islands, Spain, pp. 39-47, 2002.

[GM04] D. Giménez and J. P. Martínez, "Automatic optimization in parallel dynamic programming schemes", admitted in VECPAR04, 28-30 June 2004, Valencia, Spain.

[Gon03] A. González Escribano, "Synchronization Architecture in Parallel Programming Models", PhD Thesis, Dpto. Informática, University of Valladolid, Spain, 2003.

[Gus97] F. Gustavson, "Recursion leads to automatic variable blocking for dense linear-algebra algorithms", IBM J. Res. Develop, 41(6), pp. 737-755, 1997.

[GVL96] G. Golub and C. Van Loan, "Matrix Computations", John Hopkins Press, 3rd edition, 1996.

[Hen01] G. Henry, "Flexible high-performance matrix multiply via a self-modifying runtime code", FLAME Working Note #7, The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-2001-46, 2001.

[HJ88] R. W. Hockney and C. R. Jesshope, "Parallel Computers 2: Architecture, Programming and Algorithms", Adam Hilger /IOP Publishing, Bristol & Philadelphia, second edition, 1988.

[HN02] J. R. Herrero and J. J. Navarro, "Building libraries for small matrix kernels", XIII Jornadas de Paralelismo, Lérida, Spain, pp. 235-238, 2002.

[Hoc82] R. W. Hockney, "Characterization of Parallel Computers and Algorithms", Computer Physics Communications, 26, pp. 285-290, 1982.

[HR92] T. Heywood and S. Ranka, "A practical hierarchical model of parallel computation I, the model", Journal of Parallel and Distributed Computing, 16 (3), pp. 212-232, 1992.

[IFH01] F. Ino, N. Fujimoto and K. Hagihara, "LogGPS, A parallel computational model for synchronization análisis", proceedings of PPoPP'01, Snowbird, Utah USA, pp. 133-142, 2001.

[JC02] S. Juhász and H. Charaf, "Execution time prediction for parallel data processing tasks", proceedings of the 10th Euromicro Workshop of Parallel, Distributed and Network-based Processing (EUROMICRO-PDP 2002), IEEE Computer Society Press, Canary Islands, Spain, pp. 31-39, 2002.

[JW96] B. H. H. Juurlink and H. A. G. Wijshoff, "The E-BSP model: Incorporating unbalanced communication and general locality into the BSP model", proceedings of EUROPAR 96, LNCS 1124, Springer-Verlag, pp. 339-347, 1996.

[JW98] B. H. H. Juurlink and H. A. G. Wijshoff, "A quantitative comparison of parallel computation models", ACM Transactions on Computer Systems, 16 (3), pp. 271-318, 1998.

[Kat00] T. Katagiri, "A study on large scale eigensolvers for distributed memory parallel machines", PhD Thesis, Graduate School, University of Tokyo, Japan, 2000.

[KBG00] T. Kielmann, H. E. Bal and S. Gorlatch, "Bandwidth-efficient collective communication for clustered wide area systems", IPDPS, Cancún, Mexico, pp. 492-499, 2000.

[KGG+94] V. Kumar, A. Grama, A. Gupta and G. Karypis, "Introduction to parallel computing, design and analisis of algorithms", The Benjamin/Cummings Publishing Company, 1994.

[KKK00] T. Katagiri, H. Kuroda and Y. Kanada, "A methodology for automatically tuned parallel tridiagonalization on distributed memory vector-parallel machines", proceedings of Vector and Parallel processing 2000 (VECPAR2000), Faculdade de Engenharia da Universidade do Porto, Portugal, pp. 265-277, 2000.

[KKK01] H. Kuroda, T. Katagiri and Y. Kanada, “Parallel numerical library project –How ILIB is to be developed”, Workshop on Scalable Solver Software (SSS2001), Tokyo, Japan, 2001.

[KKK99] H. Kuroda, T. Katagiri and Y. Kanada, “Performance of automatically tuned parallel GMRES(m) method on distributed memory machines”, proceedings of Hakken Kagaku Team, University of Tokyo, Japan, pp. 11-19, 1999.

[KKP+02] A. Kalinov, A. Kossatchev, M. Posypkin and V.Shishkov, “Using ASM specification for mpC parallel programming language compiler”, 4th International Workshop on Action Semantics and Related Frameworks (AS 2002), 2002.

[KKP01] K. Karganov, K. Khorenko and M.Posypkin, “A mpC parallel development environment”, proceedings of the International Conference on Parallel and Distributed Procesing Techniques and Applications (PDPTA'2001), CSREA Press, Las Vegas, Nevada, pp.1464-1470, 2001.

[KL01] A. Kalinov and A. Lastovetsky, “Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers”, Journal of Parallel and Distributed Computing 61, pp. 520-535, 2001.

[KLV98] B. Kågström, P. Ling and C. Van Loan, “GEMM-based level 3 BLAS: high performance model implementations and performance evaluation benchmark”, ACM Trans. Math. Software, 24 (3), pp. 268-302, 1997.

[Las03] A. Lastovetsky, “Parallel computing on heterogeneous networks”, Wiley series on parallel and distributed computing, Zomaya editor, Wiley-interscience, 2003.

[LHK+79] C. Lawson, R. Hanson, D. Kincaid and F. Krogh, “Basic linear algebra subprograms for Fortran usage”, ACM Transactions on Mathematical Software, 5, pp. 308-323, 1979.

[LL97] J. Laudon and D. Lenoski, “The SGI Origin: A ccNUMA highly scalable server”, proceedings of the 24th International Symposium on Computer Architecture (ISCA’97), Denver, Colorado, USA, pp. 241-251, 1997.

[LP90] L. Liu and J. K. Peir, “A performance evaluation methodology for coupled multiple supercomputers”, International Conference on Parallel Processing, pp. 198-202, 1990.

[LSF95] J. Li, A. Skjellum and R. D. Falgout, “A poly-algorithm for parallel dense matrix multiplication on two-dimensional process grid topologies”, BLAS Technical Workshop, University of Tennessee, Knoxville, USA, 1995.

[MAG+01] L. M. Moreno, F. Almeida, D. González and C. Rodríguez, “Adaptive execution of pipelines”, PVM/MPI 2001, LNCS 2131, Springer-Verlag, pp. 217-224, 2001.

[MAG+99] D. Morales, F. Almeida, F. García, J. González, J. Roda and C. Rodríguez, “A skeleton for parallel dynamic programming”, Euro-Par ’99, LNCS 1685, Springer-Verlag, pp. 877-887, 1999.

[Mal03] A. J. Malheiro Viamonte, “Métodos unilaterais de Jacobi para a computação paralela de valores e vectores própios de matrizes simétricas”, PhD Thesis, Universidade do Minho, 2003.

[Man03] J. M. Mantas, “Desarrollo Basado en Componentes de Resolutores de Ecuaciones Diferenciales para Multicomputadores”, Ph.D. thesis, Universidad de Granada, 2003.

[MCL+01] S. Moore, D. Cronk, K. London and J. Dongarra, “Review of performance analysis tools for MPI Parallel Programs”, 8th European PVM/MPI Users´ Group Meeting, LNCS 2131, Springer Verlag, , Greece, pp. 241-248, 2001.

[MF01] C. A. Moritz and M. I. Frank, “LoGPC: Modelling network contention in message-passing programs”, IEEE Transaction on Parallel and Distributed Systems, 12(4), pp. 404-415, 2001.

[MJJ+00] J. Moura, J. Johnson, R. Johnson, P. Nagvara, D. Padua, V. Prasanna, M. Pueschel and M. Veloso, The SPIRAL WWW Home Page, www.ece.cmu.edu/spiral/, 2000.

[MMJ00] D. Mirkovich, R. Mahasoom and S. L. Johnson, “An adaptive software library for fast fourier transforms”, proceedings of the 2000 International Conference on Supercomputing, ACM Press, New York, pp. 215-224, 2000.

[Moo65] G. E. Moore, “Cramming more components onto integrated circuits”, Electronics 38(8), pp. 114-117, 1965.

[MSS01] R. T. Mills, A. Stathopoulos and E. Smirni, “Algorithmic modifications to the Jacobi-Davidson parallel eigensolver to dynamically balance external CPU and memory load”, International Conference on Supercomputing (ICS01), Sorrento, Italy, pp. 18-21, 2001.

[NDD93] J. M. Nash, P. M. Dew and M. E. Dyer, “Scalable parallel application design”, General Purpose Parallel Computing, British Computer Society Parallel Processing Specialist Group, University of Westminster, pp. 47-49, 1993.

[NP85] A. Norton and G. P. Pfister, “A methodology for predicting multiprocessor performance”, proceedings of ICPP ’85, University Park, PA, USA, pp. 772-781, 1985.

[PBD+01] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche and S. Vadhiyar, “Numerical Libraries And The Grid”, International Journal of High Performance Computing Applications, Vol. 15, SAGE Publications, pp 359-374, 2001.

[PE00] H. Park and L. Elden, “Schur-type methods for solving least squares problems with Toeplitz structure”, SIAM J. Sci. Comput. 22 (2), pp. 406-430, 2000.

[RD02] K. J. Roche and J. J. Dongarra, ”Deploying parallel numerical library routines to cluster computing in a self adapting fashion”, Parallel Computing: Advances and Current Issues, Imperial College Press, London, 2002.

[RG87] D. Rood and D. Grunwald, “The performance of multicomputer interconnection networks”, IEEE Computer 20, pp. 63-73, 1987.

[RR01] T. Rauber and G. Rünger, "A hierarchical computation model for distributed shared-memory machines", proceedings of 9th Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 2001), Mantova, Italy, pp. 57-64, 2001.

[RR97] T. Rauber and G. Rünger, "PVM and MPI communication operations on the IBM-SP2: modelling and comparison", Concurrency: Practice and Experience 9(3), pp. 181-202, Wiley, 1997.

[RSL+99] J. L. Roda, F. Sande, C. León, J. A. González and C. Rodríguez, "The collective computing model", 7th Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 1999), Funchal, Portugal, 1999.

[RVG98] D. Royo, M. Valero-García and A. González, "A Jacobi-based algorithm for computing Symmetric Eigenvalues and Eigenvectors in a Two-dimensional Mesh", Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 1998), pp. 463-469, Madrid, Spain, 1998.

[Sam71] A. H. Sameh, "On Jacobi and Jacobi-like algorithms for a parallel computer", Math. Comput. 25, pp. 579-590, 1971.

[Sch17] J. Schur, "Über potenzreihen, die im Innern des Einkitskreise beschänkt sind", Journal für die reine und angewandte Mathematik, 147, pp. 205-232, 1917.

[SFK97] D. Sima, T. Fountain and P. Kacsuk, "Advanced computer architectures, a design space approach", Addison Wesley Longman, 1997.

[SK02] F. J. Seinstra and D. Koelma, "Incorporating memory layout in the modeling of message passing programs", 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP 2002), Canary Islands, Spain, pp. 293-300, 2002.

[SLS+94] A. Skjellum, A. P. Leung, C. H. Still, S. G. Smith, Robert D. Falgout and C. H. Baldwing, "The Multicomputer Toolbox- First-generation scalable libraries", proceedings of HICSS-27, pp. 644-654, 1994.

[SOH+96] M. Snir, Steve W. Otto, S. Huss-Lederman, D. W. Walker and J. Dongarra, "MPI: the complete reference", MIT Press, 1996.

[TD01] D. Tessera and A. Dubey, "Communication policies performance: A case study", proceedings of 9th Euromicro Workshop on Parallel and Distributed Processing (EUROMICRO-PDP 2001), Mantova, Italy, pp. 491-497, 2001.

[Val90] L. G. Valiant, "A bridging model for parallel computation", Communications of the ACM, 33 (8), pp. 103-111, 1990.

[VDB01] R. Vuduc, J. W. Demmel and J. Bilmes, "Statistical models for automatic performance tuning", International Conference on Computational Science, San Francisco, CA USA, pp. 117-126, 2001.

[vdG97] R. van de Geijn, "Using PLAPACK (Scientific and Engineering Computation", MIT Press, 1997.

[VDY+02] R. Vuduc, J. W. Demmel, K. A. Yelick, S. Kamil, R. Nishtala and B. Lee, "Performance optimization and bounds for sparse matrix-vector multiply", proceedings of the IEEE/ACM Conference on Supercomputing, Baltimore, MD USA, pp. 705-714, 2002.

[VFD00] S. S. Vadhiyar, G. E. Fagg and J. J. Dongarra, "Automatically tuned collective communications", proceedings of the SC'2000, Dallas, TX USA, pp. 3-13, 2000.

[VKH+02] R. Vuduc, S. Kamil, J. Hsu, R. Nishtala, J. W. Demmel and K. A. Yelick, "Automatic performance tuning and analysis of sparse triangular solve", ICS 2002: Workshop on performance optimization via high-level languages and libraries, 2002.

[WA99] B. Wilkinson and M. Allen, "Parallel Programming. Techniques and applications using networked workstations and parallel computers", Prentice Hall, 1999.

[WOH84] R. A. Whiteside, N. S. Ostlund and P. G. Hibbard, "A parallel Jacobi diagonalization algorithm for a loop multiple processor system", IEEE Trans. Computer 33(5), pp. 409-413, 1984.

[WPD01] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project", Parallel Computing, 27 (1-2), pp. 3-35, 2001.

[WRN90] M. Willebeek-LeMair, A. P. Reeves and C. H. Ning, "Characterization of multicomputer systems: A transfer ratio approach", International Conference on Parallel Processing, pp. 171-178, 1990.

[WSH99] R. Wolski, N. T. Spring and J. Hayes, "The Network Weather Service: a distributed resource performance forecasting service for metacomputing", Journal of Future Generation Computing Systems, 15(5-6), pp. 757-768, 1999.

[Wu99] X. Wu, "Performance evaluation, prediction and visualization of parallel systems", The Kluwer international series on Asian studies in computer and information science, K.Y. Cai editor, Kluwer academic publishers, 1999.

[XZS96] Z. Xu, X. Zhang and L. Sun, "Semi-empirical multiprocessor performance predictions", Journal of Parallel and Distributed Computing 39(1), Academic Press, pp. 14-28, 1996.

[ZC03] Y. Zhang and Y. Chen, "Block size selection of parallel LU and QR on PVP-based and RISC-based supercomputers", Institute of Software, Chinese Academy of Sciences, 2003.

[Zha00] Y. Zhang, "Block Size Selection of Parallel LU Factorization", proceedings HPC-ASIA2000, IEEE Computer Society, 1, pp. 247-249, Beijing, 2000.

[Zha99] Y. Zhang, "DRAM(h,k): A parallel computation model for high performance numerical computing", Institute of Software, Chinese Academy of Sciences, 1999.

[Zha99a] Y. Zhang, "Block size selection in ScaLAPACK", Institute of Software, Chinese Academy of Sciences, 1999.

[ZLP95] M. J. Zaki, W. Li and S. Parthasarathy, "Customized dynamic load balancing for a network of worksations", TR 602, Department of Computer Science, University of Rochester, NY USA, 1995.

# Direcciones Web


[wATL] The ATLAS WWW Home Page, math-atlas.sourceforge.net/

[wBLACS] The BLACS WWW Home Page, www.netlib.org/blacs/

[wBLAS] The BLAS WWW Home Page, www.netlib.org/blas/blast-forum/

[wCEPBA] The CEPBA WWW Home Page, www.cepba.upc.es

[wCESCA] The CESCA WWW Home Page, www.cesca.es

[wCOCI] The COCI WWW Home Page, dis.um.es/COCI/

[wDEEP] The DEEP WWW Home Page, www.psrv.com/deep.html

[wDIME] The DIMEMAS WWW Home Page, www.cepba.upc.es/tools.htm

[wEISP] The EISPACK WWW Home Page, www.netlib.org/eispack/

[wESSL] The ESSL WWW Home page, www.research.ibm.com/mathsci/ams/ams_ESSL.htm

[wFTM] The FT-MPI WWW Home Page, www.cs.utk.edu/~fagg/FT_MPI/

[wGoto] The "High-Performance BLAS by Kazushige Goto" www Home page, www.cs.utexas.edu/users/flame/goto/

[wHen] Greg Henry WWW Home page, www.cs.utk.edu/~ghenry

[wICL] The ICL WWW Home Page, icl.cs.utk.edu/projects/torc/

[wLAM] The LAM-MPI WWW Home Page, www.lam-mpi.org/

[wLAPAC] The LAPACK WWW Home Page, www.netlib.org/lapack/

[wLAWRA] The LAWRA www Home Page, lawra.uni-c.dk/lawra.

[wLINP] The LINPACK WWW Home Page, www.netlib.org/linpack/

[wMPE] The MPE WWW HomePage, www-unix.mcs.anl.gov/mpi/mpich

[wMPI] The MPI WWW Home Page, www.mpi-forum.org, www-unix.mcs.anl.gov/mpi/

[wMPI2] The MPI-2 WWW Home Page, www.mpi-forum.org/docs/mpi-20-html

[wMPICH] The MPICH WWW Home Page, www-unix.mcs.anl.gov/mpi/mpich/indexold.html

[wMPIi] The MPI implementations WWW Home Page, www-unix.mcs.anl.gov/mpi/implementations.html

[wNWS] The NWS WWW Home Page, nws.cs.ucsb.edu/

[wOMP] The OpenMP WWW Home Page, www.openmp.org

[wPablo] The Pablo WWW Home Page, vibes.cs.uiuc.edu/

[wPAPI] The PAPI WWW Home Page, icl.cs.utk.edu/projects/papi/overview.html

[wPARA] The PARAVER WWW Home Page, www.cepba.upc.es/tools.htm

[wPard] The Paradyn WWW Home Page, rib.cs.utk.edu/cgi-bin/object.pl?rh=223&class=Asset&oh=83&html=1

[wPBLAS] The PBLAS WWW Home Page, www.netlib.org/scalapack/pblas_qref.html

[wPGPR] The PGPROF WWW Home Page, scv.bu.edu/SCV/Archive/linux-cluster/manpages/pgprof.html

[wPLAP] The PLAPACK WWW Home Page, www.cs.utexas.edu/users/plapack/

[wPthre] The Pthreads WWW Home Page, www.humanfactor.com/pthreads/

[wPVM] The PVM WWW Home Page, www.csm.ornl.gov/pvm/pvm_home.html

[wREMO] The REMOS WWW Home Page, www-2.cs.cmu.edu/afs/cs.cmu.edu/project/cmcl/www/remulac/remos.html

[wSANS] The SANS WWW Home Page, icl.cs.utk.edu/iclprojects/pages/sans.html

[wScaLA] The ScaLAPACK WWW Home Page, www.netlib.org/scalapack/

[wSGI] The SGI WWW Home Page, www.sgi.com

[wSun] The Sun worksations WWW Home Page, www.sun.com/desktop/products/#ultrasparcworkstations

[wTAU] The TAU WWW Home Page, www.cs.uoregon.edu/research/paracomp/tau/

[wTools] The NHSE Parallel Processing Tools WWW Home Page, rib.cs.utk.edu/cgi-bin/catalog.pl?rh=226&term=4!1

[wTOP] The TOP 500 WWW Home page, www.top500.org

[wULL] The PCG at La Laguna University, WWW Home Page, nereida.deioc.ull.es

[wVAMP] The VAMPIR WWW Home Page, www.nhse.org/ptlib/sw_forums/VAMPIR/wwwboard.html

[wZha02] L. Zhang, “PVM and MPI: A comparison of features”, 2002, www.fbi-medialab.fh-karlsruhe.de/lectures/Hypercomputer/content/PVM-MPI.html