Optimización automática de rutinas paralelas de álgebra lineal*

Ginés Carrillo[1] Javier Cuenca[2] Domingo Giménez[3] José González[2]

Resumen— **En este trabajo se analiza el diseño de rutinas de álgebra lineal para usuarios no expertos en paralelismo. Estas rutinas deben obtener automáticamente valores de diversos parámetros (número de procesadores, tamaño de bloque, distribución de los datos, topología lógica de los procesos) que permitan hacer una ejecución cercana a la óptima. Se estudian dos tipos de técnicas: una técnica basada en el diseño de rutinas de instalación asociadas a una determinada rutina de álgebra lineal, y una segunda técnica basada en el modelado de la rutina cuya ejecución se pretende automatizar. La primera técnica se analiza con una rutina de eliminación Gaussiana en una red LAN con distintos grados de heterogeneidad, y la segunda técnica se analiza con métodos de Jacobi unilaterales por bloques para la resolución del problema simétrico de valores propios en un Origin 2000.**

Palabras clave— **Optimización automática, Álgebra lineal paralela.**

I. Introducción

DESDE que Flynn en 1972 clasificó los sistemas paralelos con su célebre taxonomía [1], estos han evolucionado considerablemente hasta el punto en que nos encontramos en que, además de los sistemas paralelos tradicionales que podían encuadrarse más o menos forzadamente en alguno de los grupos establecidos por Flynn o alguna modificación de ellos, con sistemas heterogéneos y distribuidos que están empezando a ocupar un lugar muy importante en el campo de la computación de altas prestaciones [2]. El abaratamiento de los costes y la posibilidad de conexión y reutilización de los equipos ha propiciado el posible acceso de un mayor número de usuarios a sistemas de programación paralela, pero al mismo tiempo los sistemas han pasado a ser más complejos, al poder ser heterogéneos y poder variar la carga dinámicamente, lo que hace que el desarrollo de programas para los nuevos sistemas sea más difícil de lo que ya lo era hace algunos años en el caso de los multiprocesadores y multicomputadores. Es verdad que han surgido estándares para poder trabajar sobre estos sistemas (OpenMP [3], Pthreads [4], PVM [5], MPI [6], ...), pero, aunque el uso de estos estándares permite el desarrollo de algoritmos portables, estos no se comportarán de manera eficiente en todos los sistemas. El obtener programas eficientes (o al menos tan eficientes como sea posible) para los diferentes sistemas no es una tarea fácil, y no se puede esperar que sea realizada por un ususario no experto en paralelismo, aunque se le proporcionen herramientas para hacerlo. En este sentido, sería necesario el desarrollo de software para distintos campos de trabajo en distintos sistemas. En particular estamos interesados en el desarrollo de software de álgebra lineal, que es un campo muy utilizado en la resolución de problemas en diferentes campos de la ciencia y la ingeniería [7]. Nuestra idea es desarrollar rutinas que tengan en cuenta las características del sistema para decidir la forma en que se ejecutan: el tamaño de los bloques, la distribución de los datos, la topologa lógica de los procesos, el número de procesadores a utilizar, ..., de manera que el usuario ejecute la rutina y sea ésta la que obtenga los parámetros con los que se asegure un tiempo de ejecución cercano al óptimo. Estamos trabajando en dos sentidos:

- Desarrollando rutinas de instalación con las que se determinen los parámetros a utilizar por la rutina de álgebra lineal. Esta técnica se analiza en la sección 2 de este trabajo con una rutina de eliminación Gaussiana sobre una red LAN heterogénea [8].
- Estudiando modelos que nos proporcionen un valor adecuado de los parámetros mencionados. Esta técnica se analiza en la sección 3 con métodos de Jacobi unilaterales por bloques con topologías lógicas de anillo y malla bidimensional sobre un Origin 2000 [9].

Para distintos tipos de sistemas las técnicas a utilizar serán distintas, y las dos que se analizan en este trabajo no cubren todos los tipos de sistemas paralelos que podemos encontrar en la actualidad. Simplificando mucho, los tipos de sistemas se podrían clasificar en lo referente a sus elementos de proceso en **Homogéneos** y **Heterogéneos**, y lo mismo en cuanto a las redes de interconexión en el sistema; y en cuanto a la ejecución se podrían clasificar en **Estáticos** y **Dinámicos**, también en lo correspondiente a elementos de proceso y red de interconexión. De esta manera tendríamos sistemas de 16 tipos distintos, que precisarían estilos de programación distintos, como podemos ver en la tabla I.

Vemos que la programación se va dificultando desde el extremo superior izquierdo de la tabla (paralela) hasta el inferior derecho (distribuida), y aunque en la tabla se han señalado distintos tipos de programación dentro de las distintas casillas, es posible que sea preferible utilizar un tipo más sencillo

*Parte de los experimentos cuyos resultados se muestran en este trabajo han sido realizados en el Origin 2000 del Centro Europeo de Paralelismo de Barcelona (CEPBA).

[1]Departament de Tecnología. Universitat Pompeu Fabra. La Rambla, 30-32. 08002 Barcelona. gines.carrillo@tecn.upf.es

[2]Departamento de Ingeniería y Tecnología de Computadores. Univ de Murcia. Aptdo 4021. 30001 Murcia. {javiercm,joseg}@ditec.um.es

[3]Departamento de Informática: Lenguajes y Sistemas Informáticos. Univ de Murcia. Aptdo 4021. 30001 Murcia. domingo@dif.um.es

TABLA I
TIPO DE PROGRAMACIÓN EN LOS DISTINTOS SISTEMAS DE COMPUTACIÓN PARALELA.

	Red			
Proc.	Hom. Est.	Hom. Din.	Het. Est.	Het. Din.
Ho. E.	paralela	paralela heterogénea	paralela	paralela heterogénea
Ho. D.	adaptativa	adaptativa	adaptativa	adaptativa
He. E.	heterogénea	heterogénea adaptativa	heterogénea	heterogénea adaptativa
He. D.	adaptativa	adaptativa	adaptativa	adaptativa distribuida

porque no merezca la pena, para nuestro problema, el esfuerzo de programación adicional de cara a una posible reducción en el tiempo de ejecución que es difícil de cuantificar y que quizás sea pequeña.

Las dos técnicas que analizamos corresponden a los tipos de programación paralela y heterogénea, pero no a la adaptativa, habiéndose mostrado la técnica de las rutinas de instalación más adecuada para sistemas heterogéneos, aunque la técnica de modelado puede dar buenos resultados en sistemas más controlados o para tamaños de problema suficientemente grandes, donde la variación en el coste de las comunicaciones tiene menos importancia al ser su coste de orden inferior al de la computación.

II. Técnica de las rutinas de instalación

En gran parte de las aplicaciones científicas y en ingeniería se requiere la resolución de problemas de álgebra lineal, y este es el motivo por el que estamos trabajando en el diseño de rutinas de álgebra lineal especialmente diseñadas para sistemas LAN (Local Area Networks) [10].

Hay otros trabajos en los que se analiza el diseño de rutinas de álgebra lineal para redes heterogéneas [11], [12], [13]. En algunos casos la distribución de los datos en el sistema se obtiene de manera dinámica [11], y en otros casos de manera estática [12]. Nuestra finalidad es obtener rutinas cuya ejecución sea próxima a la óptima. Una posibilidad es desarrollar rutinas de instalación para las rutinas de álgebra lineal, de manera que el usuario pueda ejecutar la rutina de álgebra lineal en un procesador y esta rutina obtenga los parámetros a utilizar en la resolución del problema de la información generada por la rutina de instalación. Los parámetros a determinar por la rutina de instalación serían: el número de procesadores, la distribución de los datos, el tamaño de bloque en algoritmos por bloques, la topología lógica de los procesos, ... Además, la rutina de instalación se puede volver a ejecutar cada vez que cambien las condiciones del sistema: más memoria en algunos procesadores, modificaciones en la gestión del sistema de ficheros, eliminación o adición de procesadores, ...

Como el sistema puede estar formado por procesadores con diferentes capacidades, puede ser conveniente desarrollar las rutinas de álgebra lineal y las correspondientes rutinas de instalación para sistemas heterogéneos. Analizaremos la metodología de diseño de las rutinas de instalación para sistemas homogéneos y heterogéneos utilizando una rutina de eliminación Gaussiana y variando la heterogeneidad del sistema.

La matriz se considera dividida en bloques de filas adyacentes y estos bloques se asignan al sistema de procesadores usando un "rowwise block-cyclic-striped mapping" [14].

Si el sistema es homogéneo los bloques pueden ser todos del mismo tamaño (b). Para una matriz de un cierto tamaño (n), la rutina de instalación deberá obtener el número de procesadores (p) y el tamaño de bloque, con los que se obtiene el menor tiempo de ejecución.

En un sistema heterogéneo el tamaño de bloque no es el mismo para los diferentes procesadores. La rutina de instalación también obtiene el número óptimo de procesadores y de tamaño de bloque genérico, pero en la rutina de eliminación el tamaño de bloque en cada procesador se obtiene con la fórmula:

$$b_i = \frac{v_i}{\sum_{i=1}^{p} v_i} pb \qquad (1)$$

donde v_i es la velocidad del procesador i y b_i el tamaño de bloque asignado al procesador i. Esta es la forma en que trabaja otro software para asignación dinámica [11], pero en nuestro caso no hemos hecho asignación dinámica pues esto sería muy costoso en tiempo de ejecución.

Los valores de p y b se pueden obtener utilizando una fórmula teórica del tiempo de ejecución (que depende de la rutina) y de los parámetros de coste de una operación aritmética, de inicio de una comunicación y de transmisión de un dato (que dependen del sistema). Pero para obtener el coste teórico se hacen una serie de asunciones que son válidas cuando el tamaño del problema aumenta, pero no para problemas de menor tamaño, especialmente en sistemas LANs, debido a las carcterísticas de la red de interconexión.

Otra posibilidad es obtener los parámetros por medio de una serie de ejecuciones. En este caso el administrador del sistema decide los tamaños mínimo y máximo del problema y el incremento, y se realizan experimentos para estos tamaños. Realizar todos los posibles experimentos para todos los tamaños sería muy costoso, por lo que se obtienen p y b para el tamaño menor y se usan los valores obtenidos para un tamaño de matriz como punto de partida para el siguiente tamaño.

En el caso homogéneo se genera un fichero con los distintos tamaños de matriz y el número de procesadores y tamaño de bloques asociado, y la rutina de álgebra lineal consulta este fichero para obtener los valores de p y b de la entrada con tamaño de matriz más cercano al del problema que se quiere resolver.

En el caso heterogéneo se genera el mismo fichero y otro adicional con la velocidad relativa de los procesadores y éstos ordenados de mayor a menor velocidad. La rutina de álgebra lineal toma p y b del primer fichero, los procesadores a utilizar en la ejecución del segundo fichero, y el tamaño de bloque para cada procesador se obtiene utilizando la fórmula 1.

A. Resultados experimentales

Se muestran resultados obtenidos con diferentes rutinas de instalación para la rutina de eliminación Gaussiana. Las entradas de la matriz se han generado aleatoriamente, y se ha usado una red de estaciones SUN conectadas con ethernet. Se han considerado tres redes:

- Una red con cinco SUN Ultra 1 con la misma capacidad computacional, pero una de ellas maneja el sistema de ficheros. A esta red la llamaremos HOM.
- Una red obtenida añadiendo una SUN Ultra 5 (más rápida que los otros procesadores) a HOM. Le llamaremos HIB.
- Una red con tres procesadores: la SUN Ultra 1 que maneja el sistema de ficheros, otra SUN Ultra 1 y la SUN Ultra 5. Este es el sistema más heterogéneo, por lo que lo llamaremos HET.

En las figuras 1, 2 y 3 se comparan los resultados obtenidos con las distintas rutinas de instalación. Las figuras muestran el cociente entre los tiempos de ejecución obtenidos con el número óptimo de procesadores y tamaño de bloque dados por la rutina de instalación y el tiempo de ejecución óptimo obtenido experimentalmente. Las rutinas de instalación utilizadas son: HOMEXP (homogénea-experimental), HETEXP (heterogénea-experimental) y THEOR (teórica).

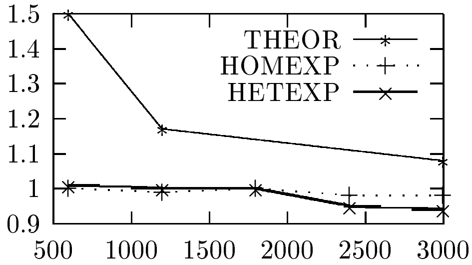

Fig. 1. Comparación entre las distintas rutinas de instalación en HOM.

Se pueden hacer algunas consideraciones:

- La rutina teórica no hace una buena predicción para tamaños pequeños, pero cuando el tamaño crece la predicción es buena. En algún caso (HET) la predicción teórica es tan buena como la experimental. Se puede utilizar la predicción teórica para matrices de gran dimensión, sobre todo teniendo en cuenta que para estas matrices la realización de los experimentos en la instalación sería muy costosa.
- La rutina homogénea trabaja bien cuando la red no es demasiado heterogénea, y la heterogénea es preferible cuando aumenta la heterogeneidad de la red.
- La rutina heterogénea trabaja bien en todos los casos, por lo que es la preferible.

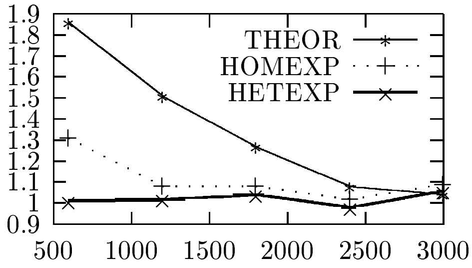

Fig. 2. Comparación entre las distintas rutinas de instalación en HIB.

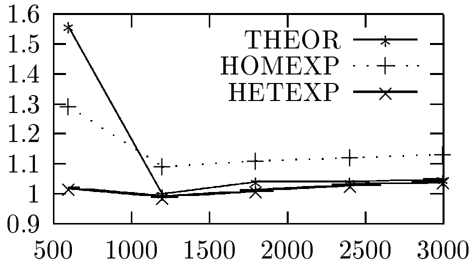

Fig. 3. Comparación entre las distintas rutinas de instalación en HET.

III. Técnica del modelado

Intentamos desarrollar un modelo analítico para algoritmos de álgebra lineal, de manera que pueda usarse para estimar el tiempo de ejecución y los valores óptimos de los parámetros antes de la ejecución de la rutina.

Históricamente el comportamiento de los algoritmos se ha modelado de diferentes maneras, con diferentes metodologías y con intenciones muy diferentes. Así, hay metodologías, como BSP y logP, concevidas como base para el análisis y diseño de algoritmos que podrían implementarse en una gran variedad de máquinas [15], [16]. Estas metodologías tienen una buena base teórica, pero la obtención de algoritmos eficientes para las máquinas actuales utilizando estas metodologías no está clara. Es necesario desarrollar un modelo donde se reflejen las características básicas de la máquina que influyen en las prestaciones del algoritmo.

Por otro lado, hay algunas técnicas en las que se lleva a cabo un estudio en profundidad de las características físicas de la plataforma hardware, lo que nos permitiría estimar el tiempo de ejecución de una rutina:

- Hay modelos donde se estudia en detalle el comportamiento de la jerarquía de memorias, y donde el tiempo de ejecución se predice en base al número de fallos de caché [17]. Sin embargo, en la mayoría de las máquinas actuales, con procesadores superescalares, el tiempo de ejecución no es una función directa del número de fallos de caché, porque es posible ocultar la latencia de memoria solapando fallos de caché con la ejecución de otras instrucciones. Además, el tiempo de ejecución depende también de otros factores, como el tamaño de página y el sistema de comunicación.

- Hay otras técnicas que generan automáticamente rutinas de álgebra lineal eficientes [18], [19]. Estas técnicas estudian las carcterísticas específicas de las rutinas en la máquina: jerarquía de memoria, unidades funcionales, registros, ... Pero tampoco tienen en cuenta el sistema de comunicación.

El modelo analítico que presentamos está basado en un modelo de arquitectura válido para una gran variedad de plataformas hardware, porque nuestra intención no es modelar exactamente las características físicas de la máquina, sino analizar con algunos parámetros (coste aritmético, de inicio de una comunicación y de transmisión de un dato) el efecto de las características de la máquina en el algoritmo. Utilizamos un modelo de programación por paso de mensaje porque este nos permitirá desarrollar rutinas que puedan ser ejecutadas en sistemas de memoria distribuida y de memoria compartida.

El modelo que arquitectura que usamos ya fue utilizado en [20] para estudiar algunas versiones del método de Jacobi. En el modelo se supone $p = 2^d$ procesadores en una red bidimensional $2^r \times 2^c$, con $r + c = d$. Se asume enlaces full-duplex y monopuerto (cada nodo puede enviar/recibir un único mensaje por la red en un momento dado). También se asume un modelo de comunicación síncrono, y que el sistema de comunicaciones utiliza un modelo wormhole switching [21], de modo que para mensajes grandes el tiempo de envío se puede considerar independiente de la distancia entre nodos.

El efecto de las características de la máquina en las prestaciones del algoritmo se puede especificar con unos pocos parámetros que definen el coste de las comunicaciones y la computación: para comunicaciones t_s representa el tiempo de inicio de la comunicación, y t_e el tiempo de envío de un dato, con lo que el envío o recepción de un mensaje de n números en coma flotante tomará un tiempo $t_s + nt_e$. Cada nodo toma un tiempo t_c en llevar a cabo una operación en coma flotante, pero si se usan rutinas de BLAS de nivel 1, 2 o 3, el coste de una operación en coma flotante será k_1, k_2 o k_3, respectivamente.

El algoritmo utilizado para estudiar esta metodología es un método de Jacobi unilateral por bloques para resolver el problema simétrico de valores propios. Este método nos permitirá estudiar todos los aspectos del análisis de rutinas paralelas de álgebra lineal, porque tendremos algoritmos por bloques (se necesita calcular el tamaño óptimo del bloque), y hay versiones para distintas topologías (se tiene que obtener el número de procesadores y la topología a utilizar).

Explicaremos brevemente el funcionamiento de un método de Jacobi. Cuando se trata de calcular los valores y vectores propios de una matriz simétrica A, el método trabaja formando una secuencia de matrices $\{A_l\}$, donde $A_{l+1} = Q_l A_l Q_l^t$, $l = 1, 2, 3, \ldots$, con $A_1 = A$, y Q_l es una rotación plana que anula un elemento no diagonal de A. La secuencia $\{A_l\}$ converge a una matriz diagonal $D = Q_k Q_{k-1} \ldots Q_2 Q_1 A Q_1^t Q_2^t \ldots Q_{k-1}^t Q_k^t$. Los elementos diagonales de D son los valores propios de A, y las columnas de $V = Q_1^t Q_2^t \ldots Q_{k-1}^t Q_k^t$ son los vectores propios de A.

Un método unilateral se obtiene trabajando sobre las matrices $B_0 = A$ y $W_0 = I$, obteniendo $B_{r+1} = Q_r B_r$, y $W_{r+1} = Q_r W_r$, con lo que $A = B_r W_r^t = Q_{r-1} Q_{r-2} \ldots Q_1 \left(B_0 W_0^t\right) Q_1^t \ldots Q_{r-2}^t Q_{r-1}^t$ [22]. Para anular a_{ij} es necesario calcular a_{ii}, a_{jj} y a_{ij}, pues el algoritmo trabaja sobre las matrices B_r y W_r, y no sobre A_r directamente. Estos elementos se pueden obtener con tres productos escalares, pero si los elementos diagonales se almacenan en un vector auxiliar sólo es necesario el cálculo de a_{ij}.

Para explotar mejor la jerarquía de memorias el algoritmo se puede modificar para obtener un algoritmo por bloques. Para esto se combinan las ideas anteriores con las de una algoritmo bilateral por bloques [23]. Las matrices B y W, de tamaño $n \times n$, se dividen en bloques de tamaño $s \times s$, y los bloques de $A = BW$ se tratan utilizando la ordenación par-impar. Inicialmente, los $\frac{n}{2s}$ bloques correspondientes al primer conjunto de Jacobi se tratan haciendo un barrido bilateral sobre bloques de tamaño $2s \times 2s$ de la matriz A, y las rotaciones se acumulan en matrices de rotaciones. Estas operaciones se hacen usando la rutina `drot` de BLAS 1. Después, las matrices B y W se actualizan multiplicando las matrices de rotaciones de dimensión $2s \times 2s$ por los correspondiente bloques de B y W, de tamaño $2s \times n$, usando la rutina `dgemm` de BLAS 3. A continuación se lleva a cabo un movimiento de filas de bloques en B y W, para agrupar los bloques de acuerdo con el siguiente conjunto de Jacobi. Antes de trabajar sobre un bloque es necesario calcular A_{ii}, A_{jj} y A_{ij}, porque el trabajo no se hace directamente sobre A. Estos bloques también se calculan con multiplicaciones matriciales (dimensión $s \times n$ por $n \times s$). Si los bloques diagonales se almacenan sólo es necesario calcular A_{ij}.

Una versión paralela para anillo (o malla con una única columna de procesadores) se presenta en [24], y nos referiremos a ella como versión 1D. Se asigna a cada procesador k bloques consecutivos de tamaño $2s \times n$ $(n = 2skp)$ de las matrices B y W. No es necesario comunicar las matrices de rotaciones a otros procesadores porque cada procesador actualiza las filas de bloques que contiene. Las únicas comunicaciones necesarias son las que hay que llevar a cabo para agrupar los bloques en los procesadores de acuerdo con los sucesivos conjuntos de Jacobi. En estas comunicaciones se envían de un procesador a otro vecino bloques de tamaño $s \times n$ de B y W y $s \times s$ de la diagonal de A.

El coste aritmético por barrido es:

$$t_{ari\_1D} = 9k_3 \frac{n^3}{p} + 12k_1 s \frac{n^2}{p} \qquad flops \qquad (2)$$

y el coste de las comunicaciones por barrido es:

$$t_{com\_1D} = 2\frac{n}{s}t_s + 4n^2t_e \quad (3)$$

Un algoritmo para malla bidimensional y que no trabaja por bloques se propone en [20]. Lo que se pretende con él es reducir el coste de las comunicaciones. Se usa una malla lógica $2^r \times 2^c$, y la actualización de las matrices, que antes se hacía con un nodo, se hace ahora con una fila de procesos. Para obtener un algoritmo por bloques se asigna a una fila de procesadores una serie de bloques consecutivos (k bloques de tamaño $\frac{n}{2s2^r}$). Al algoritmo obtenido le llamamos 2D.

Con esta topología, reducimos el número de filas en la malla en comparación con el algoritmo 1D. Como el número de pasos por barrido depende del número de filas en la malla, podemos obtener un menor coste debido a que tenemos menos intercambios de filas de bloques, y reducimos el coste de lo que a partir de ahora llamaremos comunicaciones verticales (CV). Sin embargo, como los nodos de cada fila deben cooperar para calcular y aplicar las transformaciones, aparece un nuevo tipo de comunicaciones, a las que llamaremos comunicaciones horizontales (CH), cuyo coste depende del número de columnas de la malla; estas comunicaciones son necesarias para calcular los bloques A_{ij} y para difundir las matrices de rotaciones por filas de procesadores para que cada fila de procesadores actualice las filas de bloques que contiene.

El coste aritmético por barrido es:

$$t_{ari\_2D} = 9k_3\frac{n^3}{p} + (24s + c - 1)k_1\frac{n^2}{2^{r+1}} \quad flops \quad (4)$$

El coste de las comunicaciones verticales por barrido es:

$$t_{VC\_2D} = 2\left(t_s + \frac{n}{2^{c-1}}st_e\right)\frac{n}{s} \quad (5)$$

y el coste de las comunicaciones horizontales por barrido es:

$$t_{HC\_2D} = (2c-1)\left(t_s + s^2t_e\right)\frac{n^2}{s^2 2^{r+1}} \quad (6)$$

Vemos que en el algoritmo 2D el coste aritmético es algo mayor que en el 1D, pero esto se puede ver compensado por un menor coste en el envío de los datos (término en t_e) dependiendo de los valores de los parámetros en el sistema en que se esté trabajando.

A. Resultados experimentales

El sistema utilizado para estudiar nuestro modelo analítico ha sido un SGI Origin 2000 con procesadores R10000 a 250 MHz. Aunque el sistema es de memoria virtual compartida utilizamos programación por paso de mensajes con MPI. Los experimentos se han llevado a cabo compartiendo los recursos con otros usuarios, pues esta es la forma normal de utilizar la máquina y el contexto en que se ejecutarían las rutinas que pretendemos desarrollar.

TABLA II

Valores de k_3 (en microsegundos) como función del tamaño de bloque.

Tamaño de bloque	32	64	128
k_3	0.005	0.004	0.003

Para estimar el tiempo de ejecución se han calculado los parámetros que definen las características de la arquitectura y que aparecen en el estudio teórico del algoritmo (k_3, k_1, t_s y t_e). La estimación de cada uno de los parámetros se ha hecho con una rutina que lleva a cabo alguna operación básica. Inicialmente hemos tomado un valor medio de cada parámetro, tal como se hace en estudios similares [23],[25]. Para k_1 se ha obtenido un valor de 0.01 μs utilizando `drot`. Para k_3 se ha obtenido 0.004 μs usando `dgemm`. Los parámetros de la comunicación se han obtenido con rutinas comunicando todos los nodos en la red, obteniéndose $t_s = 70$ μs y $t_e = 0.1$ μs. Además, se ha realizado una estimación de los parámetros en función del tamaño de los problemas y de la configuración de la red, y se ha encontrado una dependencia de k_3 con respecto al tamaño de bloque (tabla II), por lo que se han utilizado esos valores de k_3 en el estudio.

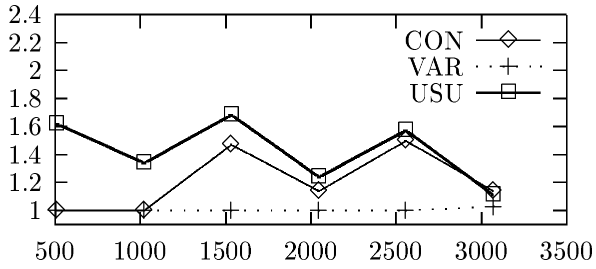

Fig. 4. Cociente de los tiempos de ejecución con distintos métodos de elección de los parámetros con respecto al tiempo óptimo. Con 4 procesadores.

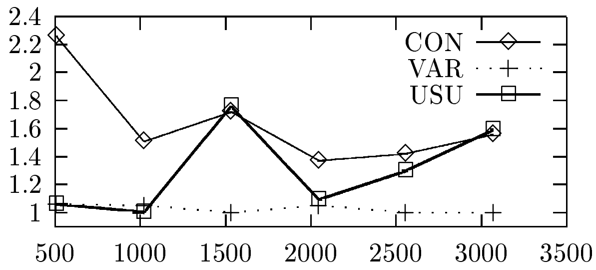

Fig. 5. Cociente de los tiempos de ejecución con distintos métodos de elección de los parámetros con respecto al tiempo óptimo. Con 8 procesadores.

Como hemos mencionado previamente, el objetivo de nuestro modelo no es predecir exactamente el tiempo de ejecución, sino que sea útil para hacer una buena elección de los parámetros. La compartición de recursos con otros usuarios hace difícil el hacer una buena estimación del tiempo de ejecución, pero no impide el hacer una buena estimación de los parámetros. En las figuras 4 y 5 se muestran los cocientes de diferentes tiempos de ejecución con respecto a los tiempos óptimos para distintos tamaños de problema, con 4 y 8 procesadores, respectivamente. Se muestran los cocientes de los tiempos obtenidos con los parámetros óptimos considerando k_3 constante (CON) o variable (VAR), y con la media de las diferentes configuraciones de la red y tamaños de bloque 32, 64 y 128 (USU), con lo que se pretende obtener una estimación del tiempo que podría tardar una ejecución realizada por un usuario no experto. Vemos que en los dos casos la predicción realizada por el modelo cuando k_3 se toma variable es muy satisfactoria, pues el cociente está siempre muy cercano a 1.

IV. Conclusiones

Se ha mostrado que es posible realizar una buena elección de los parámetros de una rutina paralela de álgebra lineal de manera automática. Esto permitirá desarrollar librerías dirigidas a usuarios no expertos de manera que las rutinas de la librería obtengan valores de los parámetros con los que la ejecución produzca unos tiempos de ejecución cercanos a los óptimos.

Se han analizado dos técnicas para realizar dicha elección: diseño de rutinas de instalación y modelado. En ambos casos hay que determinar cuáles son los parámetros del sistema que influyen en la ejecución de la rutina que estamos intentando optimizar, y cómo los parámetros de la rutina se ven influenciados por los del sistema. Aunque cuáles sean los parámetros varía de un sistema a otro y de una rutina a otra, lo que pretendemos mostrar es una metodología de trabajo que pueda ser aplicable en otros casos.

En el futuro pretendemos aplicar estas técnicas a otras rutinas y sistemas; para validar el método de trabajo y finalmente diseñar una librería de álgebra lineal paralela con optimización automática.

Agradecimientos

Agradecemos a Miguel Valero García la información proporcionada relativa a modelos.

Referencias

[1] M. J. Flynn, “Some computer organizations and their effectiveness,” *IEEE Transactions on Computers*, vol. 21, no. 9, pp. 948–960, 1972.
[2] E. Strohmaier, J. J. Dongarra, H. W. Meuer, H. D. Simon, “The marketplace of high-performance computing,” *Parallel Computing*, vol. 25, pp. 1517–1544, 1999.
[3] “OpenMP Fortran Application Program Interface 1.0,” 1997.
[4] D. R. Buthenhof, *Programming with POSIX Threads*, Addison-Wesley, 1996.
[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM Parallel Virtual Machine*, The MIT Press, 1996.
[6] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, J. Dongarra, *MPI The Complete Reference*, The MIT Press, 1996.
[7] A. Edelmann, “Large Dense Numerical Linear Algebra in 1993: The Parallel Computing Influence,” *The International Journal of Supercomputer Applicationsg*, vol. 7, no. 2, pp. 113–128, 1993.
[8] D. Giménez, G. Carrillo, “Installation routines for linear algebra libraries on LANs,” in *VECPAR 2000*, 2000.
[9] J. Cuenca, D. Giménez, J. González, “Modeling the behaviour of Linear Algebra Algorithms: a study with One-sided Jacobi Methods on Origin 2000,” 2000. In Preparation.
[10] D. Giménez, C. Jiménez, M. J. Majado, N. Marín, A. Martín, “Solving Eigenvalue Problems on Networks of Processors,” in *Vector and Parallel Processing-VECPAR-98*, J. M. L. M. Palma, J. Dongarra and V. Hernández, Ed., number 1573 in Lecture Notes in Computer Science, pp. 85–99. Springer-Verlag, 1999.
[11] A. Kalinov, A. Lastovetsky, “Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers,” in *High Performance Computing and Networking*, P. Sloot, M. Bubak, A. Hoekstra and B. Hertzberger, Ed., number 1593 in Lecture Notes in Computer Science, pp. 191–200. Springer-Verlag, 1999.
[12] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, F. Vivien, “Algorithmic issues on heterogeneous computing platforms,” *Parallel Processing Letters*, vol. 9, no. 2, pp. 197–213, 1999.
[13] V. Boudet, A. Petitet, F. Rastello, Y. Robert, “Data Allocation Strategies for Dense Linear Algebra Kernels on Heterogeneous Two-dimensional Grids,” in *IASTED Parallel and Distributed Computing and Systems*, 1999.
[14] V. Kumar, A. Grama, A. Gupta, G. Karypis, *Introduction to Parallel Computing. Design and Analysis of Algorithms*, The Benjamin Cummings Publishing Company, 1994.
[15] L. G. Valiant, “A Bridging Model for parallel Computation,” *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.
[16] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, T. von Eicken, “LogP. A Practical Model of Parallel Computation,” *Communications of the ACM*, vol. 39, no. 11, pp. 78–85, 1996.
[17] B. B. Fraguela, R. Doallo, E. Zapata, “Memory hierarchy performance prediction for blocked sparse algorithms,” *Parallel Processing Letters*, vol. 9, no. 3, pp. 347–360, 1990.
[18] J. Bilmes, K. Asanovic, C. W. Chin, J. Demmel, “Optimizing Matrix Multiply using PHIPAC: a Portable, High-Performance, ANSI C Coding Methodology,” in *Proceedings of the International Conference on Supercomputing*. 1997, pp. 340–347, ACM.
[19] R.C. Whaley, J.J. Dongarra, “Automatically Tuned Linear Algebra Software (ATLAS),” Tech. Rep. LAPACK Working Note 131, 1997. `www.netlib.org/atlas`.
[20] D. Royo, M. Valero-García, A. González, “A Jacobi-based algorithm for computing Symmetric Eigenvalues and Eigenvectors in a Two-dimensional Mesh,” in *Euromicro Workshop on Parallel and Distributes Processing*. 1998, pp. 463–469, IEEE.
[21] L. M. Ni, P. K. McKinley, “A Survey of wormhole routing techniques in direct networks,” *IEEE Computer*, vol. 26, no. 2, pp. 62–76, 1993.
[22] B. A. Chartres, “Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store,” *The Computer Journal*, vol. 5, pp. 52–60, 1963.
[23] D. Giménez, V. Hernández, R. Van de Geijn, A. M.Vidal, “A block Jacobi method on a mesh of processors,” *Concurrency: Practice and Experience*, vol. 9, no. 5, pp. 391–411, 1997.
[24] D. Giménez, J. Cuenca, R. M. Ralha and A. J. Viamonte, “One Sided Block Jacobi Methods for the Symmetric Eigenvalue Problem,” in *Proceedings of the third International Meeting on Vector and Parallel Processing*, 1998, pp. 687–692.
[25] K. Dackland, B. Kågström, “A Hierarchical Approach for Performance Analysis of ScaLAPACK-based Routines Using the Distributed Linear Algebra Machine,” in *Applied Parallel Computing: Computations in Physics, Chemistry and Engineering Science*, Dongarra et. al., Ed., Lecture Notes in Computer Science, pp. 187–195. Springer-Verlag, 1996.