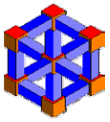
Incremento del reuso de datos de códigos dispersos en arquitecturas SMT

J. C. Pichel, D. B. Heras, J. C. Cabaleiro y F. F. Rivera

José Carlos Cabaleiro Domínguez
Grupo de Arquitectura de Computadores
Dpto. Electrónica y Computación
Universidad de Santiago de Compostela

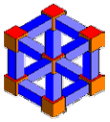
Índice

- Contexto y motivación
- Introducción
- Modelo y mejora de localidad
- Arquitecturas *multithreading*
- Mejora de localidad en SMTs
- Resultados
- Conclusiones y trabajo futuro



Índice

- **Contexto y motivación**
- Introducción
- Modelo y mejora de localidad
- Arquitecturas *multithreading*
- Mejora de localidad en SMTs
- Resultados
- Conclusiones y trabajo futuro

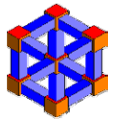


Línea de investigación general

- **Mejora del rendimiento de códigos irregulares**
- Grupo de Arquitectura de Computadores
- Soporte:
  - Soluciones *middleware* y hardware en computación de altas prestaciones: Aplicación a códigos multimedia y de simulación. Ministerio de Ciencia y Tecnología (MCYT)

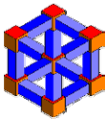
    Ref. **TIN2004-07797-C02**.
  - Soporte hardware y software para computación de altas prestaciones: Ministerio de Ciencia y Tecnología (MCYT)

    Ref. **TIN2007-67357-C03**.



Línea de investigación específica

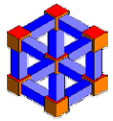
- ***Optimización del rendimiento de códigos irregulares a través de la reordenación de los accesos en tiempo de ejecución***
- Equipo de trabajo:
  - Dr. Francisco Fernández Rivera.
  - Dr. José Carlos Cabaleiro Domínguez.
  - Dra. Dora Blanco Heras.
  - D. Juan A. Lorenzo del Castillo.
  - Dr. David Expósito Singh. UC3M.
  - Dr. Juan Carlos Pichel Campos. UC3M.



Resultados académicos

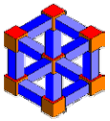
- Tesis doctoral (2000):
  - ***Modelado y mejora de la localidad en códigos irregulares***

    Autora: Dora Blanco Heras
- Tesis doctoral (2003):
  - ***Técnicas de compilación para la paralelización de códigos irregulares***
  - Autor: David Expósito Singh
- Tesis doctoral (2006):
  - ***Técnicas de optimización de la localidad para códigos irregulares sobre arquitecturas multiprocesador y multithreading***
  - Autor: Juan Carlos Pichel Campos



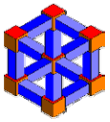
Motivación

- La optimización de códigos irregulares es un problema abierto, y requiere un esfuerzo importante
- Existen muchas aplicaciones en las que se requiere abordar este problema
- Los sistemas mono y multiprocesador, *multithreading*, *multicore*, MPSoC, ... se pueden ver beneficiados
- Existen problemas relacionados que se pueden abordar con este planteamiento: optimizar el balanceo de la carga, eliminar el efecto de la falsa compartición, minimizar el efecto de las invalidaciones, minimizar los costes de los mecanismos de coherencia, ...



Índice

- Contexto y motivación
- **Introducción**
- Modelo y mejora de localidad
- Arquitecturas *multithreading*
- Mejora de localidad en SMTs
- Resultados
- Conclusiones y trabajo futuro

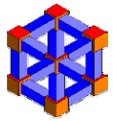


Contexto general del problema

- ¿Qué entendemos por **código irregular**?

  Códigos en los que el acceso a la memoria (datos) es desconocido en tiempo de compilación: indirecciones, punteros, ...

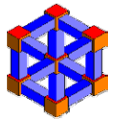
- Se han propuesto una gran variedad de mecanismos eficientes para la optimización de códigos regulares, pero no para códigos irregulares



Código irregular

```
do i=1,N
      ... = A[x[i]]  ...
      .
      .
      A[y[i]] = ...
end do
```

Problema: x[i] e y[i] son desconocidos hasta la ejecución



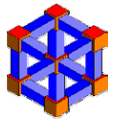
Planteamiento general

Objetivo general:

- Incrementar la localidad de los accesos para aprovechar la estructura jerárquica de la memoria

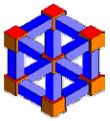
Formas de abordar el problema en la literatura:

1. Reorganizaciones de código
   - *Blocking*, *tiling*, ...
2. Reordenación de accesos a datos
   - Para obtener patrones en banda
   - Para reducir *fill-in*
   - Para mejorar la localidad



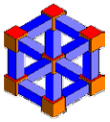
Nuestro grupo

- Metodología de reordenamiento de los accesos a los datos
- Aplicación a distintas arquitecturas:
  - Uniprocesadores
  - Multiprocesadores de memoria distribuida
  - Multiprocesadores de memoria compartida
  - Multithreading
- En este trabajo
  - Técnica de reordenamiento de datos de códigos irregulares cuando se ejecutan en arquitecturas SMT
  - Como caso de estudio, consideramos el producto de una matriz dispersa por un conjunto de vectores



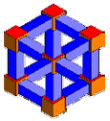
Índice

- Contexto y motivación
- Introducción
- **Modelo y mejora de localidad**
- Arquitecturas *multithreading*
- Mejora de localidad en SMTs
- Resultados
- Conclusiones y trabajo futuro

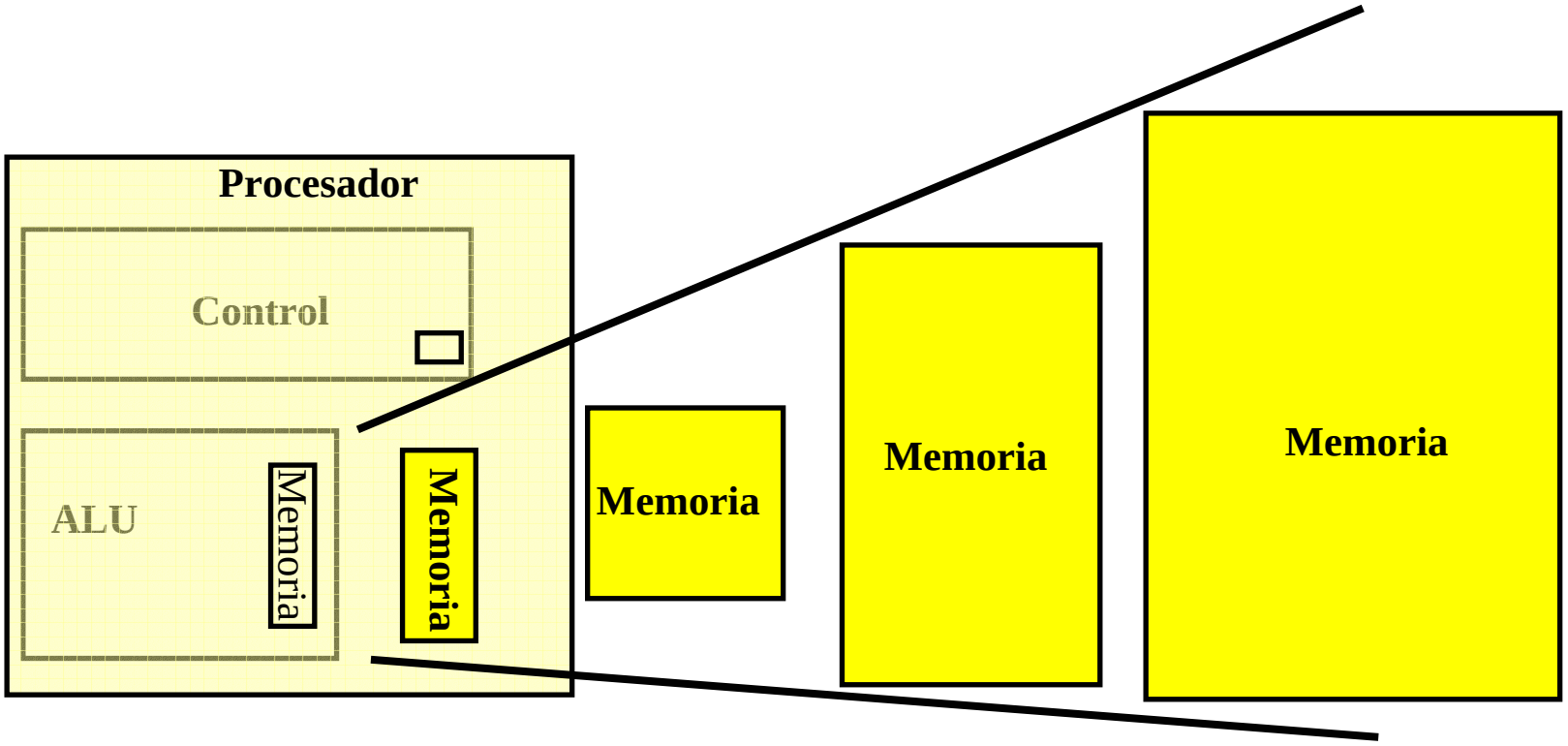


Principio de localidad

- Los programas acceden a una parte relativamente pequeña de su espacio de direcciones en un intervalo de tiempo
- **TEMPORAL:** Se referencia en el futuro próximo lo que se ha referenciado en el pasado reciente
- **ESPACIAL:** Se referencian direcciones próximas en el espacio de direcciones



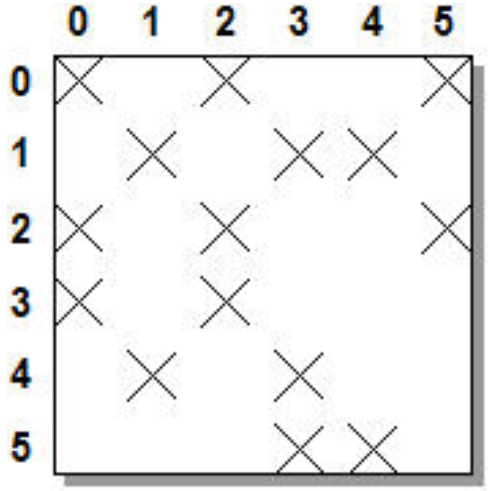
Jerarquía de memoria



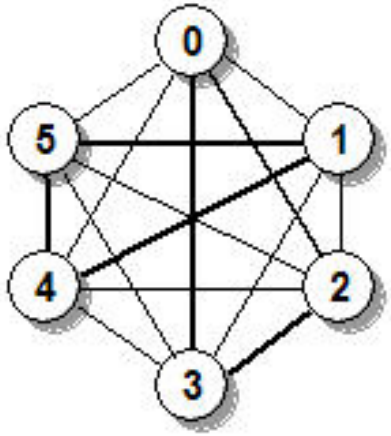
Caracterización de la localidad: definición de distancia

- Dado un patrón de acceso, proponemos diversas medidas para caracterizar su localidad, en particular consideramos:

$$d_1(x,y) = n_{elems}(x) + n_{elems}(y) - 2a_{elems}(x,y)$$

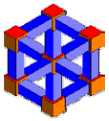


Matriz de entrada



	1	2	3	4	5
0	—	0	1	—	—
1		—	—	1	1
2			1	—	—
3				—	—
4					2

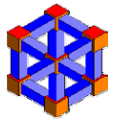
d_1



Distancia total

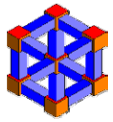
- Todas las funciones de distancias, concretamente d_1, son **normas**
- Su definición es aplicable tanto a filas como a columnas
- Se ha extendido a ventanas de localidad
- Definimos la distancia total de una matriz dispersa como:

$$D_1 = \sum_{i=0}^{N-2} d_1(i,i+1)$$



Mejora de localidad basada en distancias

- Cuantificación de la localidad como el inverso de la suma de distancias entre cada par de filas/columnas
- Existe una clara analogía con el TSP. Es un problema NP-completo
- Las distancias entre cada par de filas/columnas, dan lugar a una matriz de distancias
- La matriz de distancias no es densa
- Versión de Applegate et al. (1998) del algoritmo de Lin-Kernighan



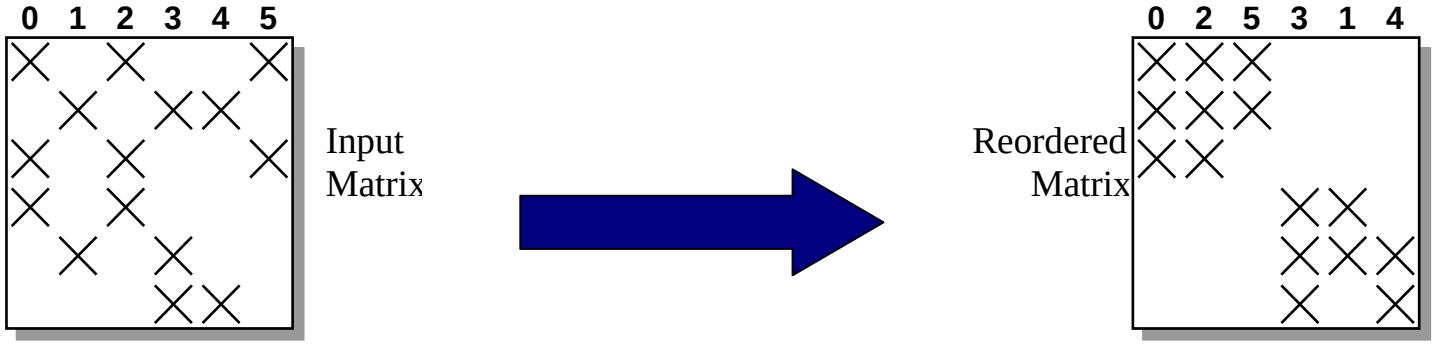
TSP


Hellín
Jumilla
Elda
Monóvar
Novelda
Aspe
Pinoso
Algueña
Elche de la Sierra
Letur
Férez
Calasparra
Moratalla
Caravaca de la Cruz
Archena
Abanilla
Cox
Almoradí
Algorfa
Molina de Segura
Beniel
Bullas
MURCIA
Torrevieja
Alhama de Murcia
Fuente Álamo de Murcia
Lorca
Vélez-Rubio
Puerto Lumbreras
Mazarrón
CARTAGENA
Taberno
Huércal-Overa
Cuevas del Almanzora
Arboleas
ELX
Crevillente
Pola
Aigües
Agost
Salinas
N-344
N-332
A-7
A-30
A-31
A-70
A-91
AP-7
E-15

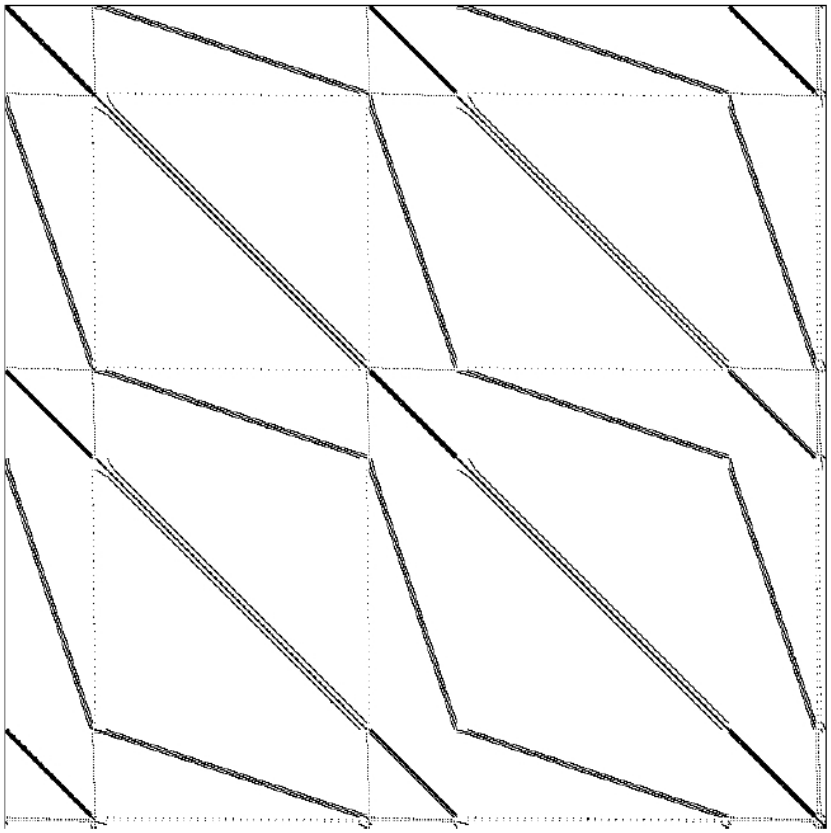
TSP

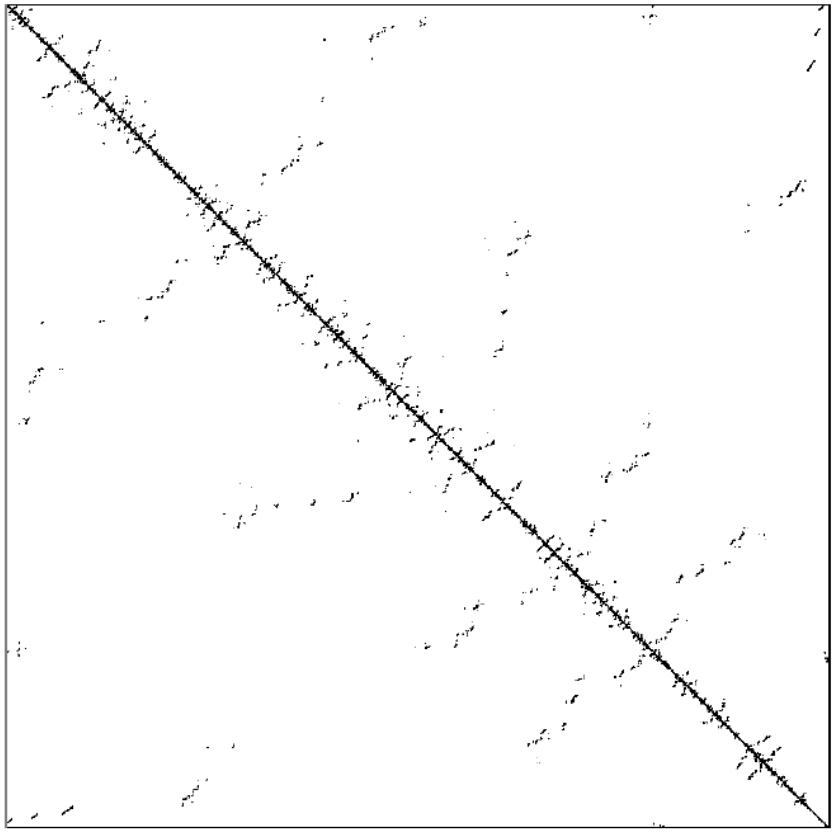

Hellín
Jumilla
Salinas
Elda
Monóvar
Novelda
Aspe
Pinoso
Algueña
Elche de la Sierra
Letur
Férez
Calasparra
Moratalla
Caravaca de la Cruz
Bullas
Abarán
Blanca
Archena
Molina de Segura
Mula
Pliego
Abanilla
Cox
Almoradí
Algorfa
Torrevieja
Pilar de la Horadada
Sn. Javier
Alhama de Murcia
Totana
Fuente Álamo de Murcia
Lorca
Vélez-Rubio
Puerto Lumbreras
Mazarrón
Taberno
Huércal-Overa
Cuevas del Almanzora
Arboleas
Águilas
ELX
MURCIA
CARTAGENA
S. Pola
Aigües
N-344
N-332
A-30
A-31
A-70
AP-7
A-7
E-15
A-91
MU-602

Ejemplo sencillo



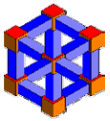
Ejemplo



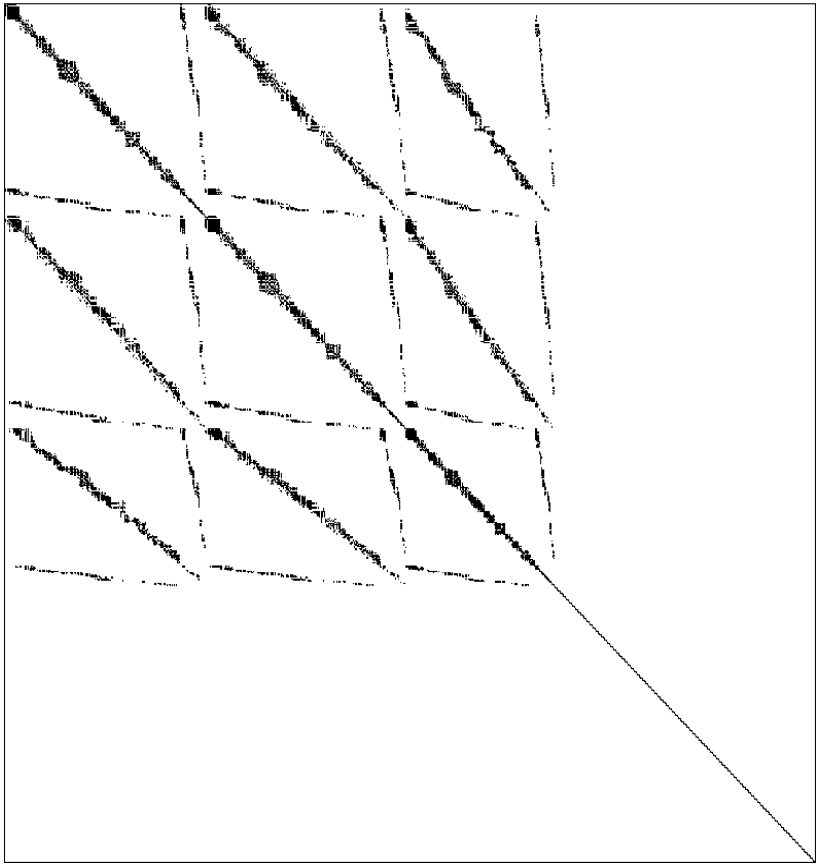


Original

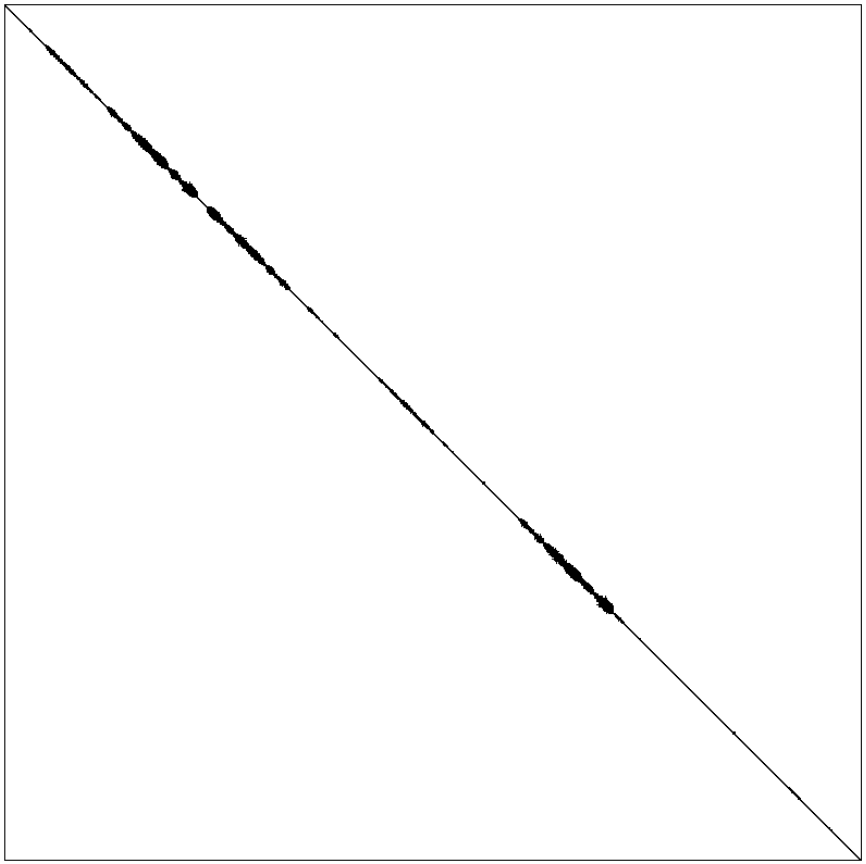
Reordenada



Otro ejemplo



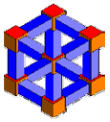
Original



Reordenada

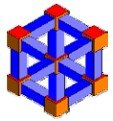
Índice

- Contexto y motivación
- Introducción
- Modelo y mejora de localidad
- **Arquitecturas *multithreading***
- Mejora de localidad en SMTs
- Resultados
- Conclusiones y trabajo futuro



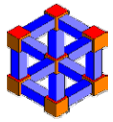
Arquitecturas *multithreading*

- Multithreading Simultáneo (SMT):
  - Ejecución simultánea de varios threads en las unidades funcionales en un mismo ciclo de reloj
  - Explotar TLP al mismo tiempo que el ILP
  - Implementación sin añadir complejidad al diseño de un procesador actual:
    - Nuevo mecanismo de búsqueda de instrucciones
    - Banco de registros de mayor tamaño
    - Se replican pocas estructuras (contadores de programa, pilas de retorno a las subrutinas, ...)
    - Las demás estructuras se comparten dinámicamente (unidades funcionales, cachés, TLBs, ...)
  - Ejemplos: procesadores de Intel con tecnología HT, IBM Power 5, Alpha 21464



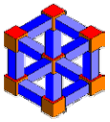
Arquitecturas *multithreading*

- Niveles de la memoria caché compartidos:
  - Cada thread genera accesos propios → Cuello de botella
  - Comportamiento difiere del de un sistema multiprocesador:
    - Compartición de datos beneficiosa (incluso la falsa compartición)
- Tecnología Hyper-Threading (Intel):
  - Un procesador aparece como dos procesadores lógicos:
  - Diferencias con la arquitectura SMT:
    - Algunas estructuras se dividen estáticamente:
      - Aísla de manera más eficiente a los threads
      - Efectos negativos de particionamiento estático se minimizan con sólo dos threads



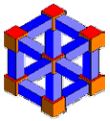
Índice

- Contexto y motivación
- Introducción
- Modelo y mejora de localidad
- Arquitecturas *multithreading*
- **Mejora de localidad en SMTs**
- Resultados
- Conclusiones y trabajo futuro



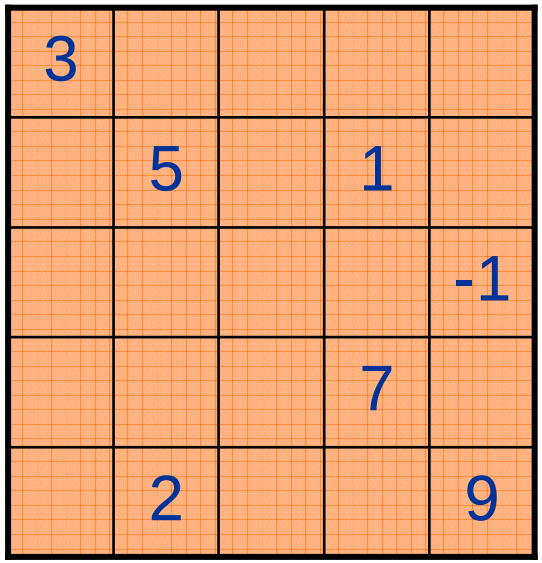
Objetivo

- Mejorar la localidad de los datos de códigos dispersos en arquitecturas SMT y, con ello, el tiempo de ejecución
  - Se parte de un modelo de localidad
  - Técnica de reordenamiento de datos de códigos irregulares cuando se ejecutan en arquitecturas SMT
  - Como caso de estudio, consideramos el producto de una matriz dispersa por un conjunto de vectores
  - Evaluación de la propuesta

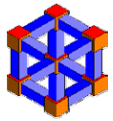


Formato CRS para matrices dispersas

- Vectores:
  - PTR — Puntero de inicio de fila
  - INDEX — Índices de columna
  - DA — Datos



PTR = {0, 1, 3, 4, 5, 7}
INDEX = {0, 1, 3, 4, 3, 1, 4}
DA = {3, 5, 1, -1, 7, 2, 9}

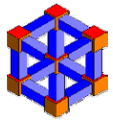


Caso de estudio

```
for (i=0; i < N; i++) {
   for (j=0; j < M; j++) {
      reg=0;
      for (k=PTR[i]; k < PTR[i+1]; k++) {
         reg = reg + DA[k]*B[j][INDEX[k]];
      }
      R[i][j]=reg;
   }
}
```

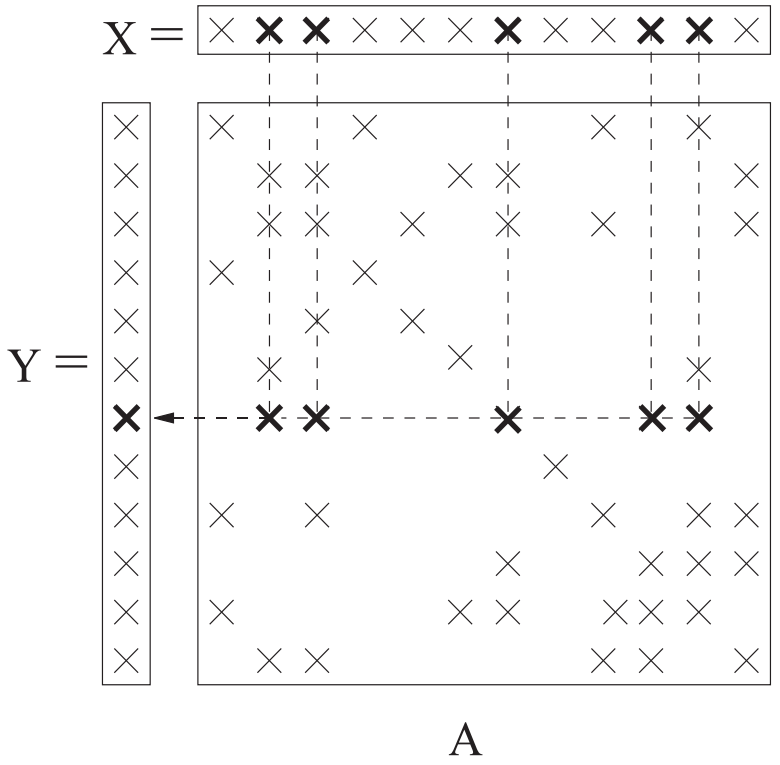
Producto matriz dispersa por M vectores, (M<<N)

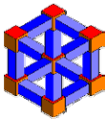
- Vectores DA, INDEX y PTR alta localidad espacial
- Matriz R alta localidad espacial
- B localidad dependiente de INDEX (matriz dispersa)
  - El acceso a cada fila de B depende del patrón de la matriz dispersa
  - Agrupando las entradas de la matriz dispersa se mejora la localidad de este lazo



Localidad en el SpMxV

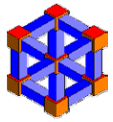
- Cada fila *j* de B es un vector X
- La localidad en los accesos a X se puede mejorar:
  - Espacial: agrupando las columnas
  - Temporal: agrupando por filas



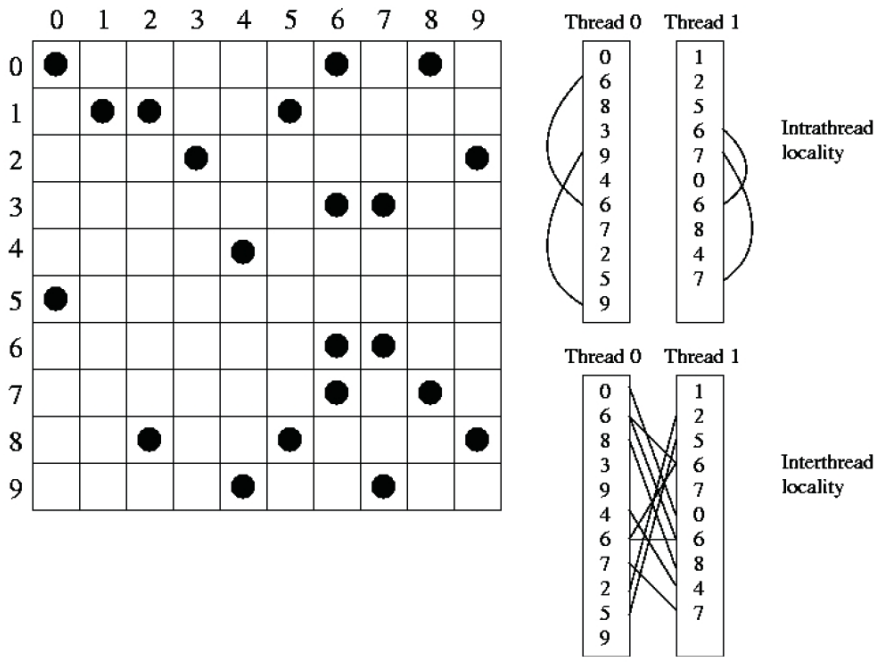


Localidad en SMT

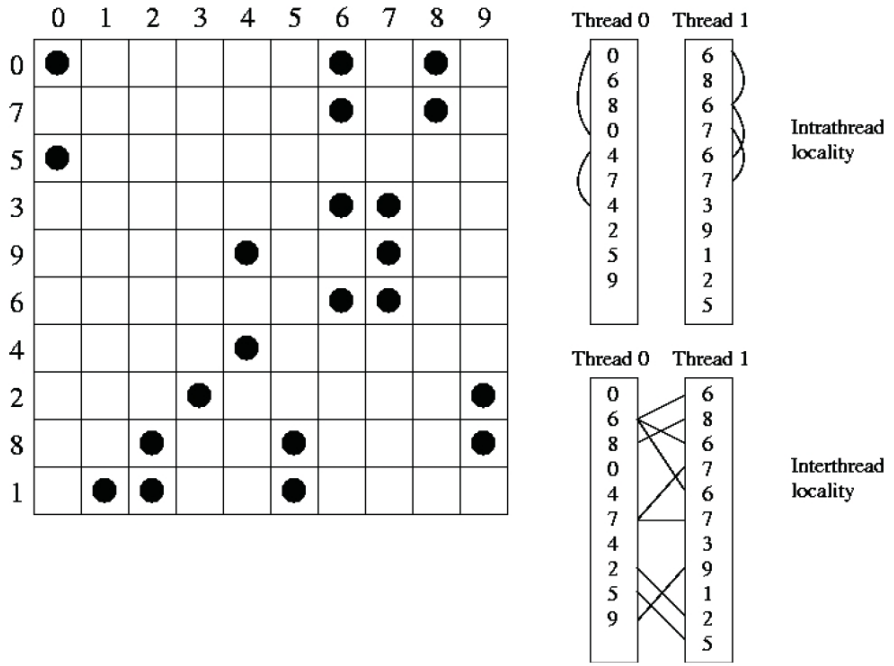
- Optimización de la localidad en arquitecturas SMT:
  - Localidad Intra-thread
  - Localidad Inter-thread
- Cada iteración *i* recorre la matriz dispersa por filas
- Distribución cíclica del lazo *i*:
  - Agrupamiento de entradas entre filas consecutivas de la matriz (localidad inter-thread)
  - Agrupamiento dentro de cada fila (localidad intra-thread)



Localidad intra-thread e inter-thread

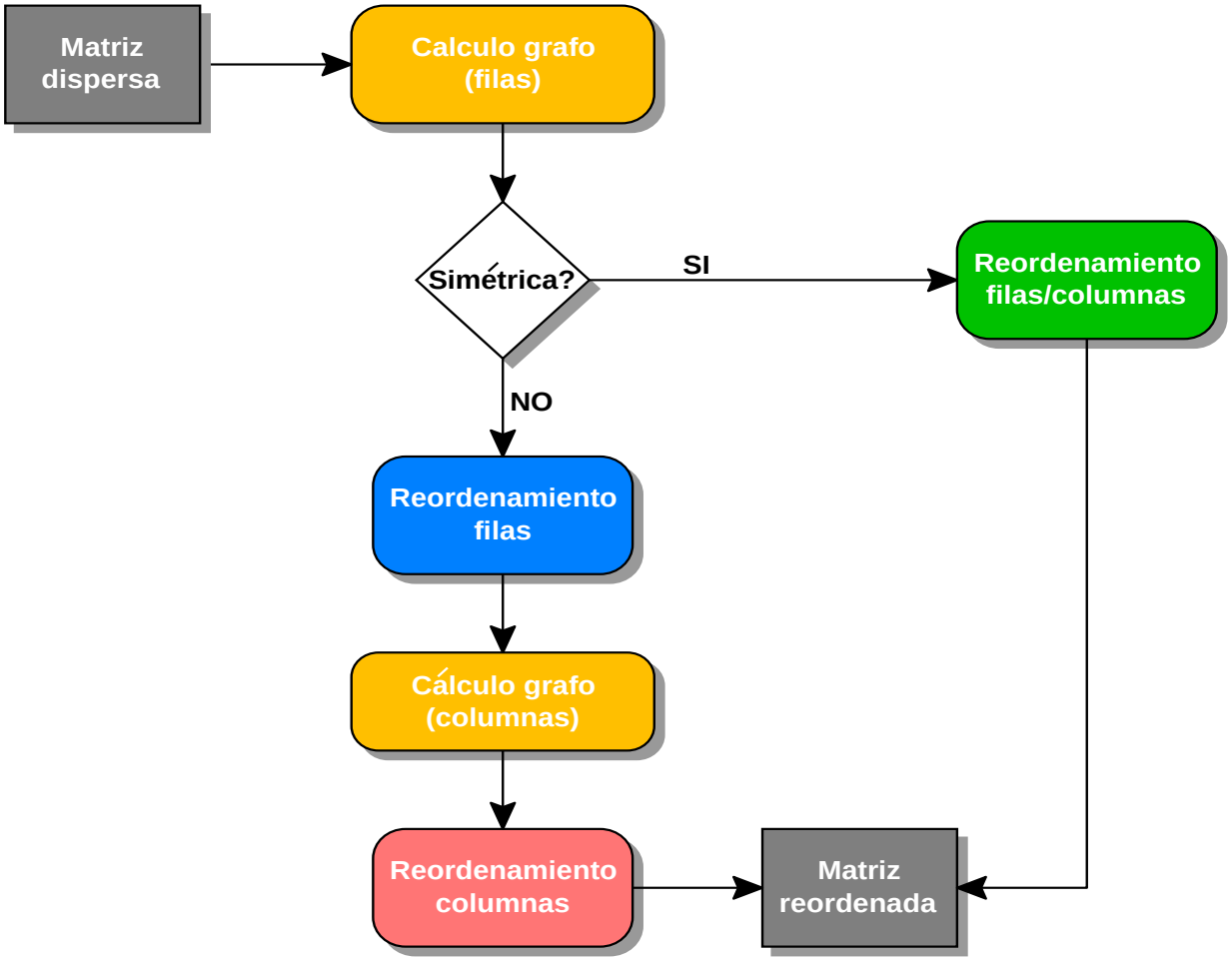


Matriz original



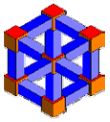
Matriz reordenada

Optimización de localidad en SMT



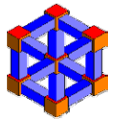
Índice

- Contexto y motivación
- Introducción
- Modelo y mejora de localidad
- Arquitecturas *multithreading*
- Mejora de localidad en SMTs
- **Resultados**
- Conclusiones y trabajo futuro



Evaluación del rendimiento

- Análisis de rendimiento:
  - Simulador de arquitectura SMT (SMTSIM):
    - Comparación con técnicas de reordenamiento estándar (RCM, AMD y METIS)
    - Comportamiento de la jerarquía caché
    - Tiempos de ejecución
    - Speedup
  - Tecnología Hyper-Threading (Pentium IV):
    - Comportamiento de la jerarquía caché
    - Tiempos de ejecución
    - Speedup



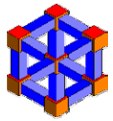
Condiciones experimentales

	L1 I	L1 D	L2	L3
Size	32 KB	32 KB	256 KB	4 MB
Associativity	2	2	2	2
Line Size (bytes)	64	64	64	64

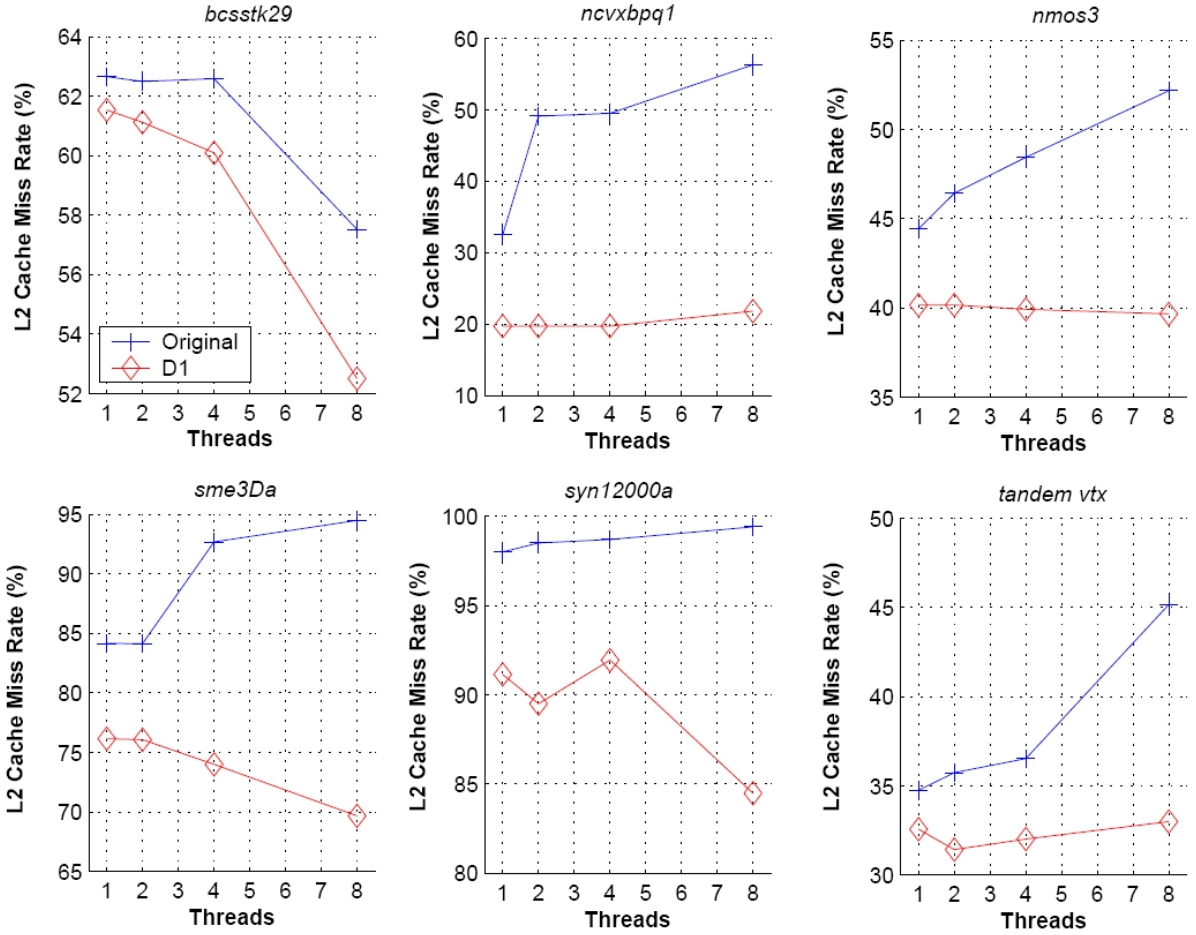
Jerarquía caché simulada

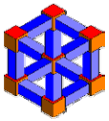
	N	N_Z		N	N_Z
bcsstk29	13992	619448	*ncvxbpq1*	50000	349968
e40r0100	17281	553562	*nmos3*	18588	386594
garon2	13535	390607	*psmigr_1*	3140	543162
memplus	17758	126150	*sme3Da*	12504	874887
msc10848	10848	1229778	*syn12000a*	12000	1436806
Na5	5832	305630	*tandem_vtx*	18454	253350

Conjunto de matrices de prueba

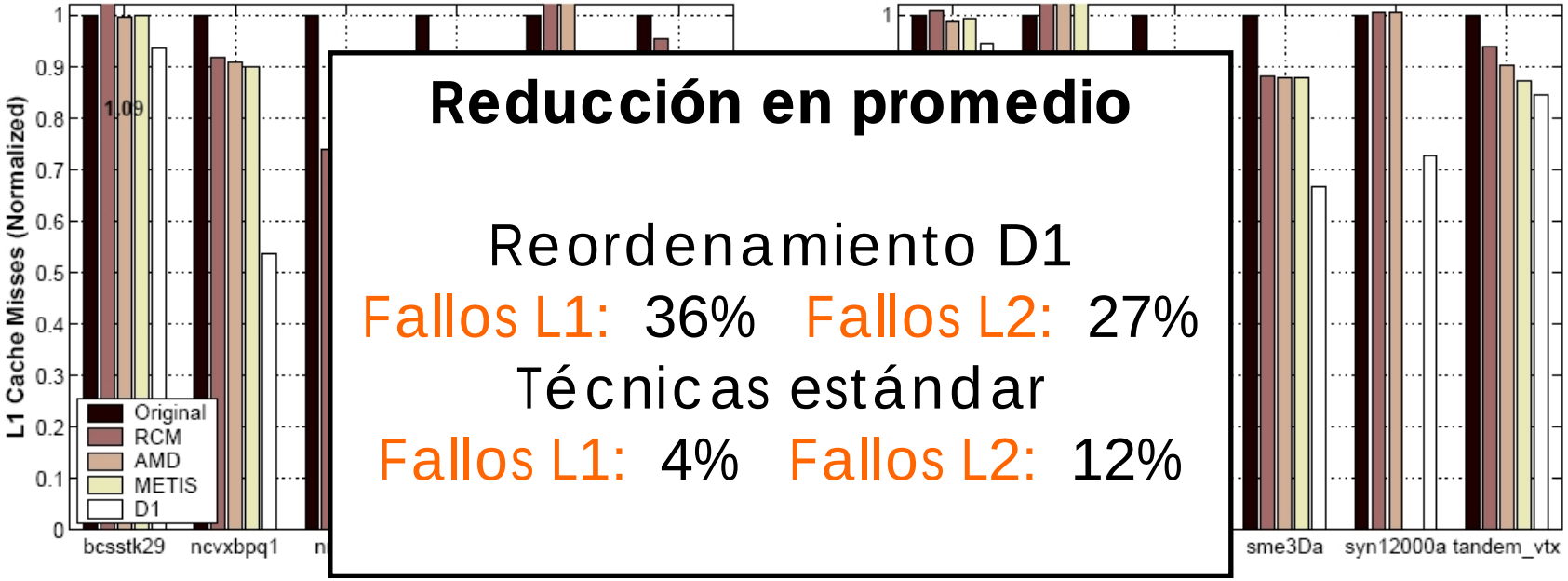


Contención en L2 (simulador)





Fallos caché (simulador)



Reducción en promedio

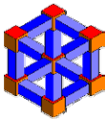
Reordenamiento D1

Fallos L1: 36% Fallos L2: 27%

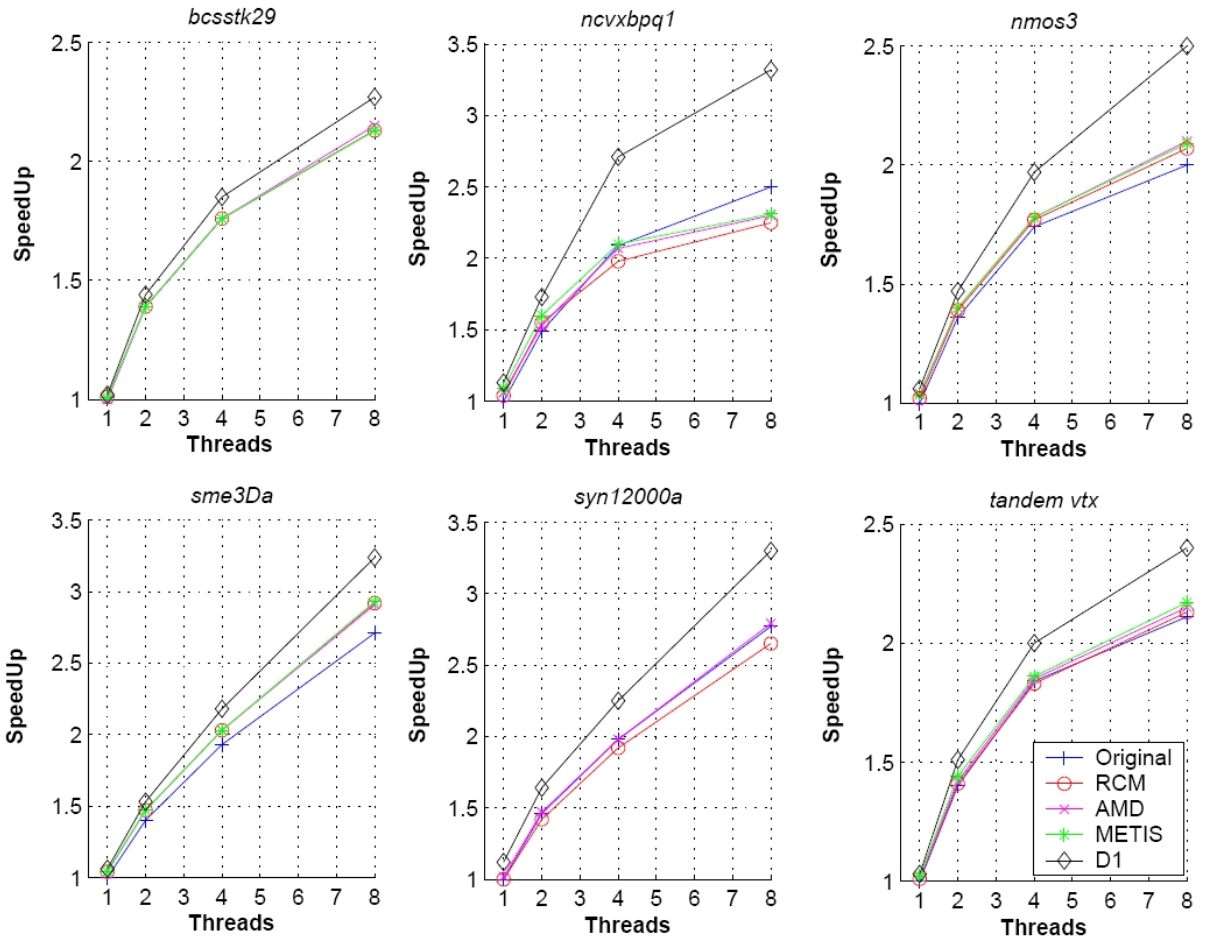
Técnicas estándar

Fallos L1: 4% Fallos L2: 12%

8 Threads

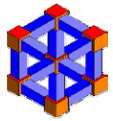


Speedup (simulador)

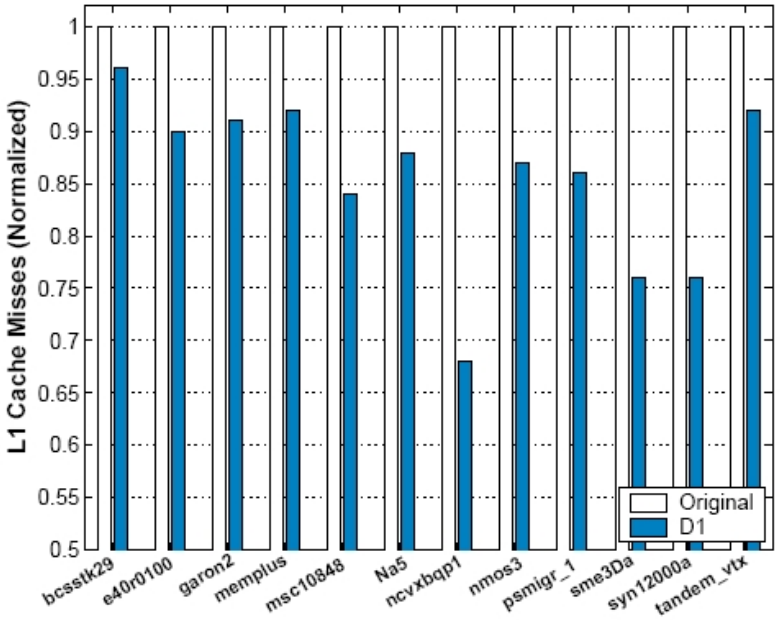


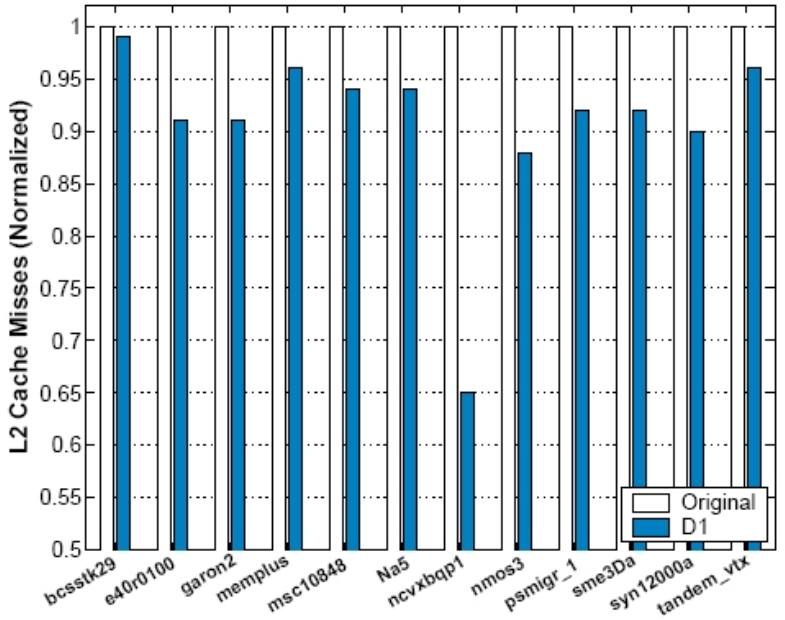
Resultados con Hyper-Threading

- Tecnología Hyper-Threading (Pentium IV):
  - Disponemos de tres configuraciones: No HT, 1-HT y 2-HT
- Condiciones experimentales
  - Pentium IV (Prescott) , 3.2 GHz y 1 GB de memoria
  - L1: caché de datos de 16 KB asociativa de 8 vías
  - L2: caché unificada de 1024 KB asociativa de 8 vías
  - Tamaño de línea de 64 bytes
  - *Trace* caché de 12K-instrucciones decodificadas asociativa de 8 vías
  - Código en C, compilador de Intel para Linux y OpenMP
  - PAPI, para acceder a los contadores hardware

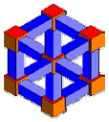


Fallos caché (HT)

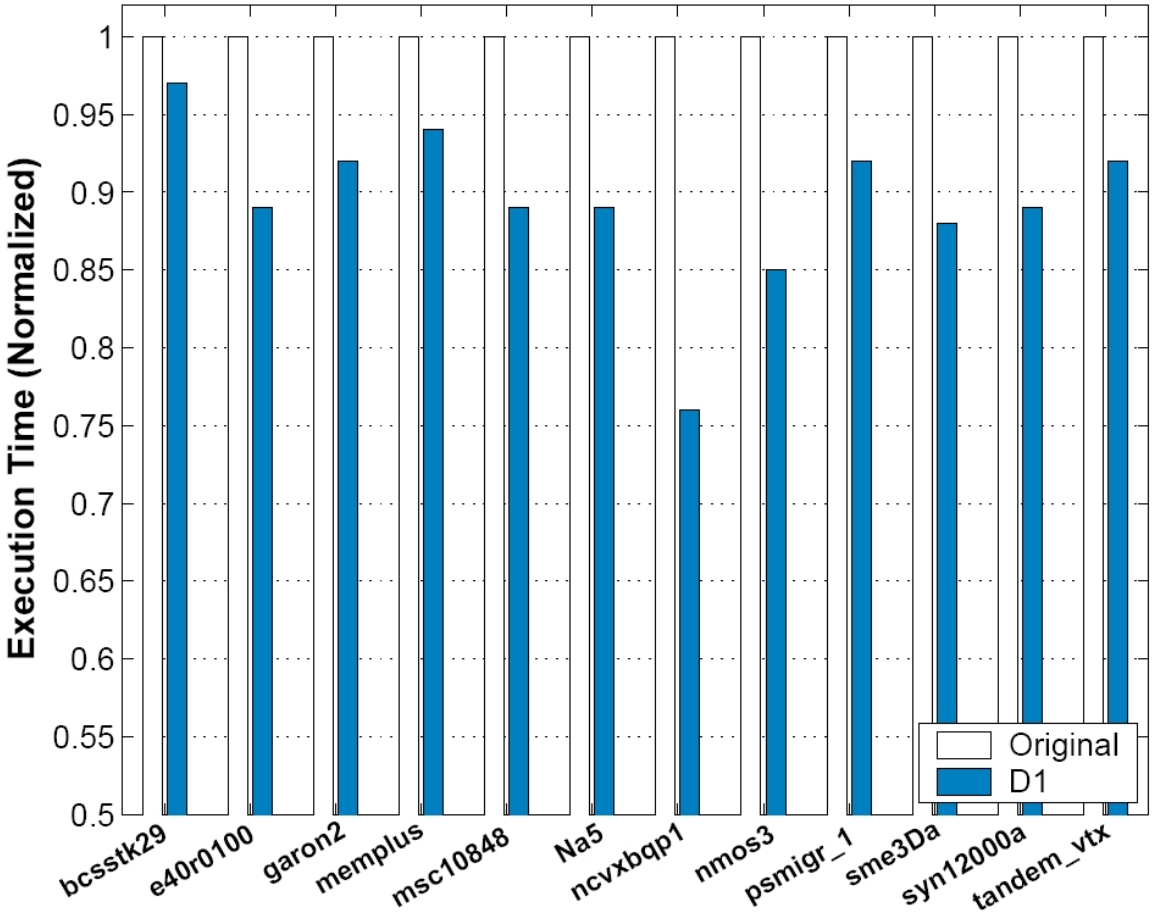




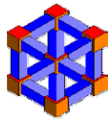
- HT sólo permite 2 threads → menos fallos debidos a localidad inter-thread
- Reducciones destacables en el número de fallos (en promedio de un 16% en L1 y 9% en L2)



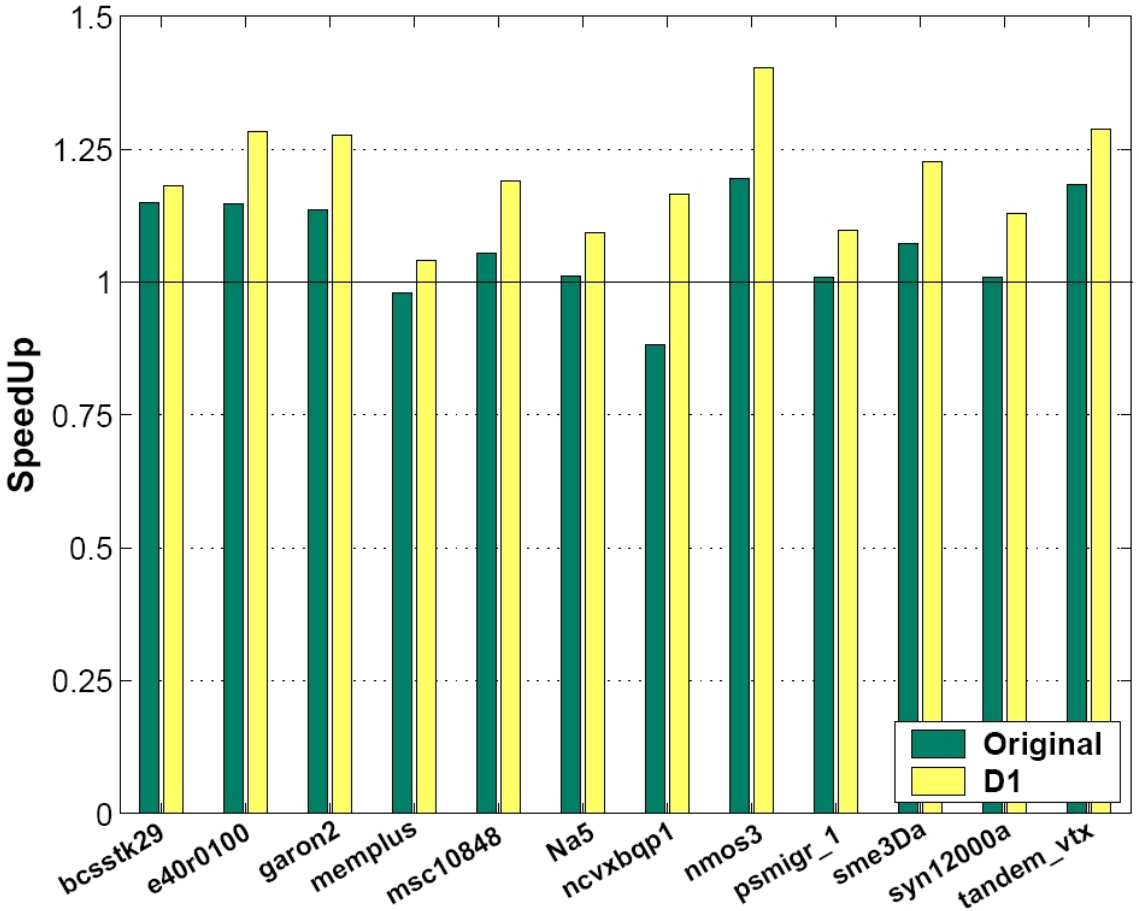
Tiempo de ejecución (HT)



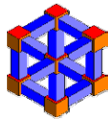
- Disminución del tiempo de ejecución en todos los casos considerados
- Reducción en promedio de un 12% (3% - 24%)



Speedup (HT)

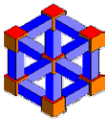


- Speedup respecto a la matriz original y en secuencial
- En todos los casos considerados, con D1 speedup > 1
- En promedio 1.2 (1.05 – 1.4)



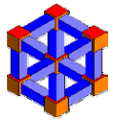
Índice

- Contexto y motivación
- Introducción
- Modelo y mejora de localidad
- Arquitecturas *multithreading*
- Mejora de localidad en SMTs
- Resultados
- **Conclusiones y trabajo futuro**



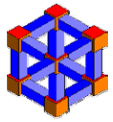
Conclusiones

- Principales aportaciones
  - Propuesta de cuantificación de la localidad en accesos irregulares a los datos: concepto de distancia
  - Mejora de la localidad a través de la optimización de la distancia
  - Se mejora la localidad en los accesos sobre arquitecturas SMT
  - La mejora de localidad se traduce en una mejora en tiempo de ejecución
  - Las mejoras son más notables cuando el número de threads es mayor



Trabajo futuro

- Validar los métodos en arquitecturas multicore y en constelaciones
- Reducir el tiempo de sobrecarga (cálculo de distancias y optimización)
  - Paralelizar el modelado y la mejora de localidad
  - Simplificar los reordenamientos
- Analizar nuevos códigos irregulares
- Nuevas definiciones de distancia
- Mejora de localidad basada en contadores hardware



Gracias por su atención

José Carlos Cabaleiro Domínguez
Grupo de Arquitectura de Computadores
Dpto. Electrónica y Computación
Universidad de Santiago de Compostela
http://www.ac.usc.es