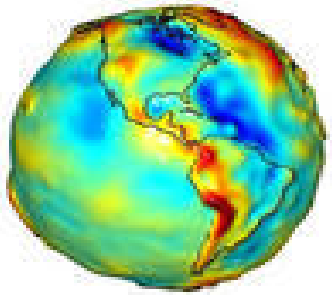
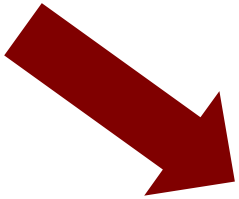
# Técnicas Superescalares en la Construcción de Bibliotecas Numéricas para Procesadores Multinúcleo y GPUs

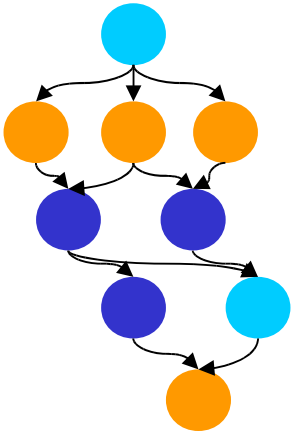

**Enrique S. Quintana Ortí**

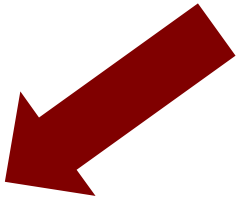
quintana@icc.uji.es http://www.hpca.uji.es

# XX JORNADAS DE PARALELISMO
## Castellón, 2008 A Coruña, 2009





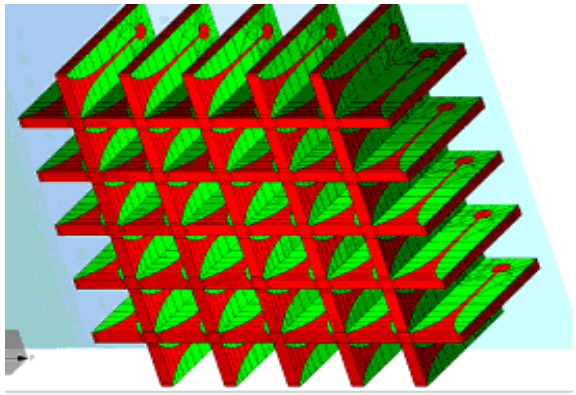





intel
Xeon

# SISTEMAS DE ECUACIONES LINEALES
## Simulación de campos electromagnéticos

- Antena Vivaldi



Resolver

$$A\ x = b$$

$A$ de dimensión

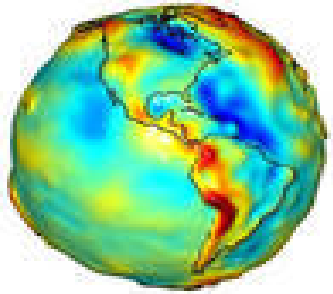
$49.820 \times 49.820$ a

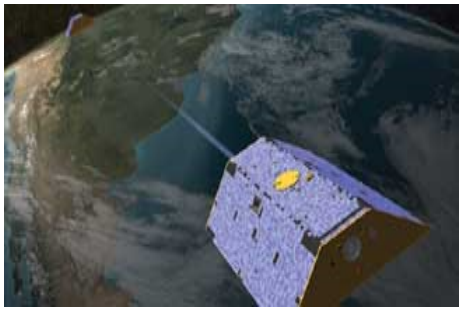
$92.729 \times 92.729$,

en función de la frecuencia de operación de la antena

# PROBLEMA DE MÍNIMOS CUADRADOS
## Estimación del campo gravitatorio terrestre

- Proyecto GRACE





Resolver

$$min_x \| A\ x - b \|$$

$A$ de dimensión
$\sim 130.000 \times 130.000$
(156 GBytes)

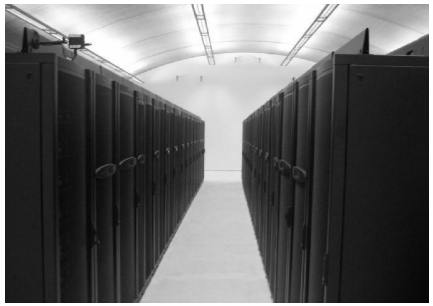
***elmundo.es*** (25.08.2009)
**Satélites espaciales constatan que la India se seca por dentro**

La India se seca por dentro, y por fuera. Más de 33 centímetros cúbicos de aguas subterráneas desaparecen cada año debido [...]

# BIBLIOTECAS DE COMPUTACIÓN
## Utilidad

Posibilitan los avances de otras ciencias (*Computational Science*) →

Simulaciones en lugar de experimentación: menores costes económicos y temporales



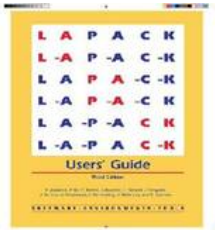
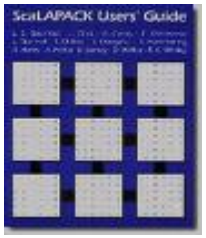
# BIBLIOTECAS DE COMPUTACIÓN
## Utilidad

- ¿Cuánto tardaríamos en resolver un problema lineal de mínimos cuadrados en condiciones ideales?
  - Algoritmo limitado únicamente por la velocidad del procesador, no por la conexión con la memoria
  - 1 procesador Intel Xeon 3,6 GHz (4 operaciones aritméticas/ciclo)

| | | |
|---|---|---|
| 200.000 x 200.000 | → | 8,6 días |
| 400.000 x 400.000 | → | 68,8 días |

# BIBLIOTECAS DE COMPUTACIÓN
## Alto rendimiento

- Necesidad de una ejecución eficiente: persiguiendo a la arquitectura HPC del momento…

| 1980<br>ISA vectorial | 1990<br>Memoria caché | 2000<br>Memoria distribuida |
|---|---|---|
|  |  |  |
|  |  |  |

# BIBLIOTECAS DE COMPUTACIÓN
## Alto rendimiento

- Necesidad de una ejecución eficiente: persiguiendo a la ¿*arquitectura HPC del momento*...?

La Ley de Moore sigue vigente, pero...

- No es posible aumentar la frecuencia por los problemas de disipación de calor y consumo

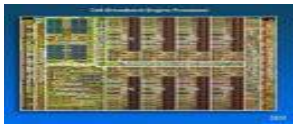
$$f \times 1{,}3 \rightarrow \text{consumo } \times 2$$

- Hay poco paralelismo más a nivel de instrucción
- La latencia de memoria es muy alta (1 acceso a memoria ≈ 240 ciclos)

# BIBLIOTECAS DE COMPUTACIÓN
## Alto rendimiento

- Necesidad de una ejecución eficiente: persiguiendo a la ¿*arquitectura HPC del momento*…?

Aceleradores *hardware*



Procesadores multinúcleo de propósito general

# TÉCNICAS SUPERESCALARES
## Índice

- Introducción
- Técnicas superescalares para la construcción de bibliotecas de computación matricial sobre procesadores multinúcleo y GPUs:
  1. Ejecución paralela dirigida por las dependencias de datos
  2. Uso de cachés *software* para ocultar las existencia de múltiples espacios de direcciones (DSM)
  3. Uso de cachés *software* para ocultar la latencia de acceso a disco

# TÉCNICAS SUPERESCALARES
## Índice

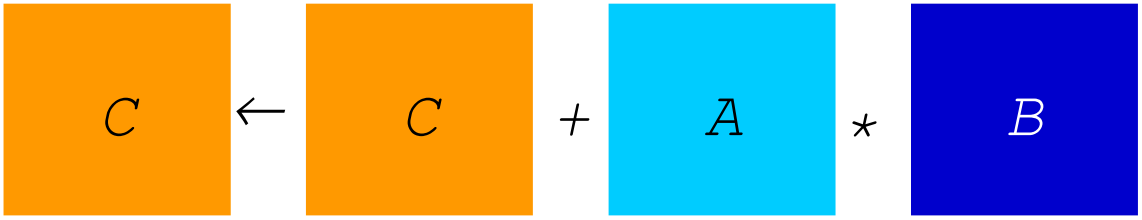
- Introducción
- Técnicas superescalares para la construcción de bibliotecas de computación matricial sobre procesadores multinúcleo y GPUs:
  - Aplicación de las técnicas a nivel de “bloque”
  - Implementación mediante *software*
  - Paralelismo a nivel de tarea/hebra
  - Objetivo: núcleos del procesador

# TÉCNICAS SUPERESCALARES
## Índice

- Introducción
- Técnicas superescalares para la construcción de bibliotecas de computación matricial sobre procesadores multinúcleo y GPUs:
  1. Ejecución paralela dirigida por las dependencias de datos
  2. Uso de cachés *software* para ocultar las existencia de múltiples espacios de direcciones (DSM)
  3. Uso de cachés *software* para ocultar la latencia de acceso a disco

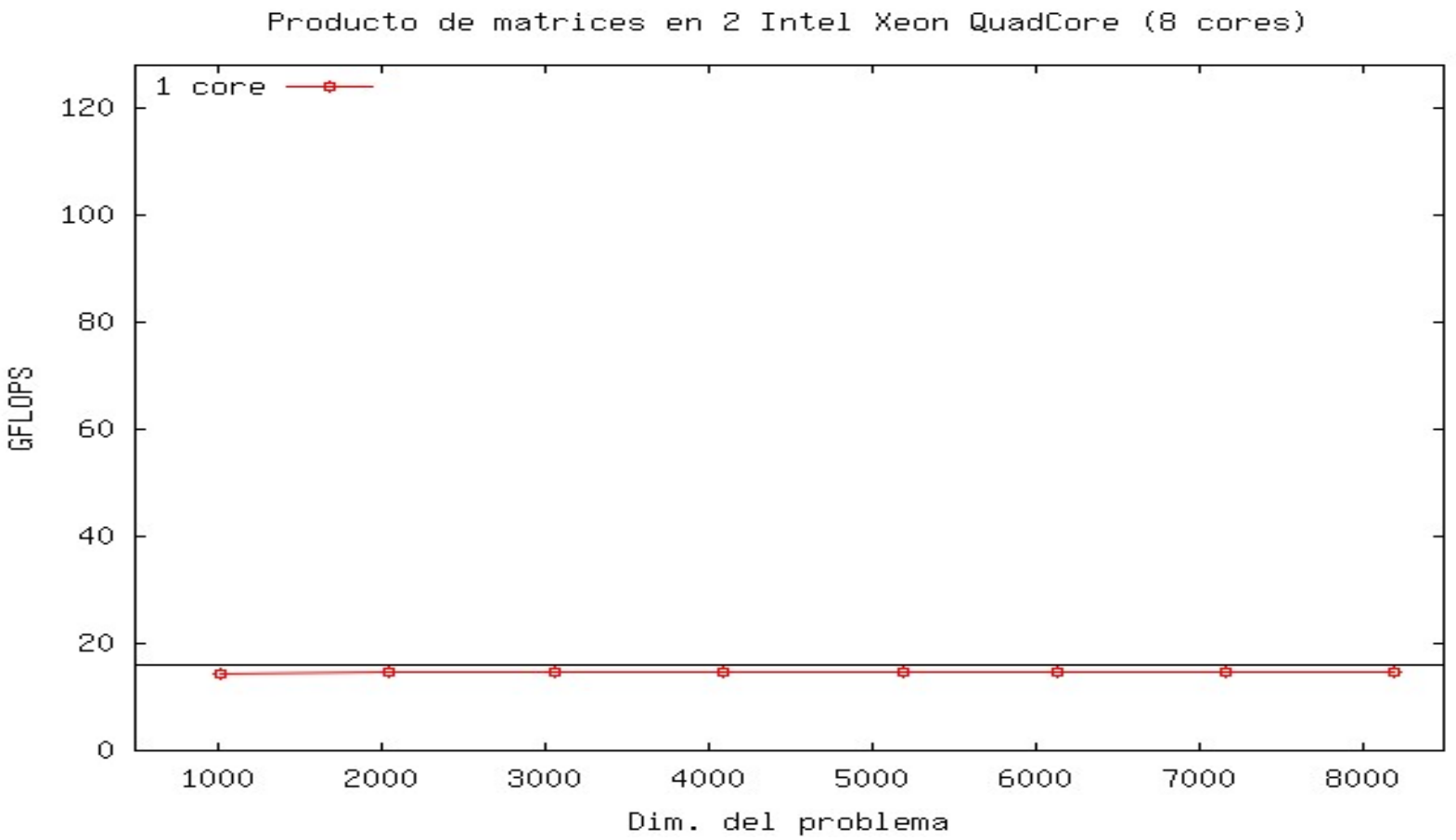
# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo

- Una operación sencilla: producto de matrices



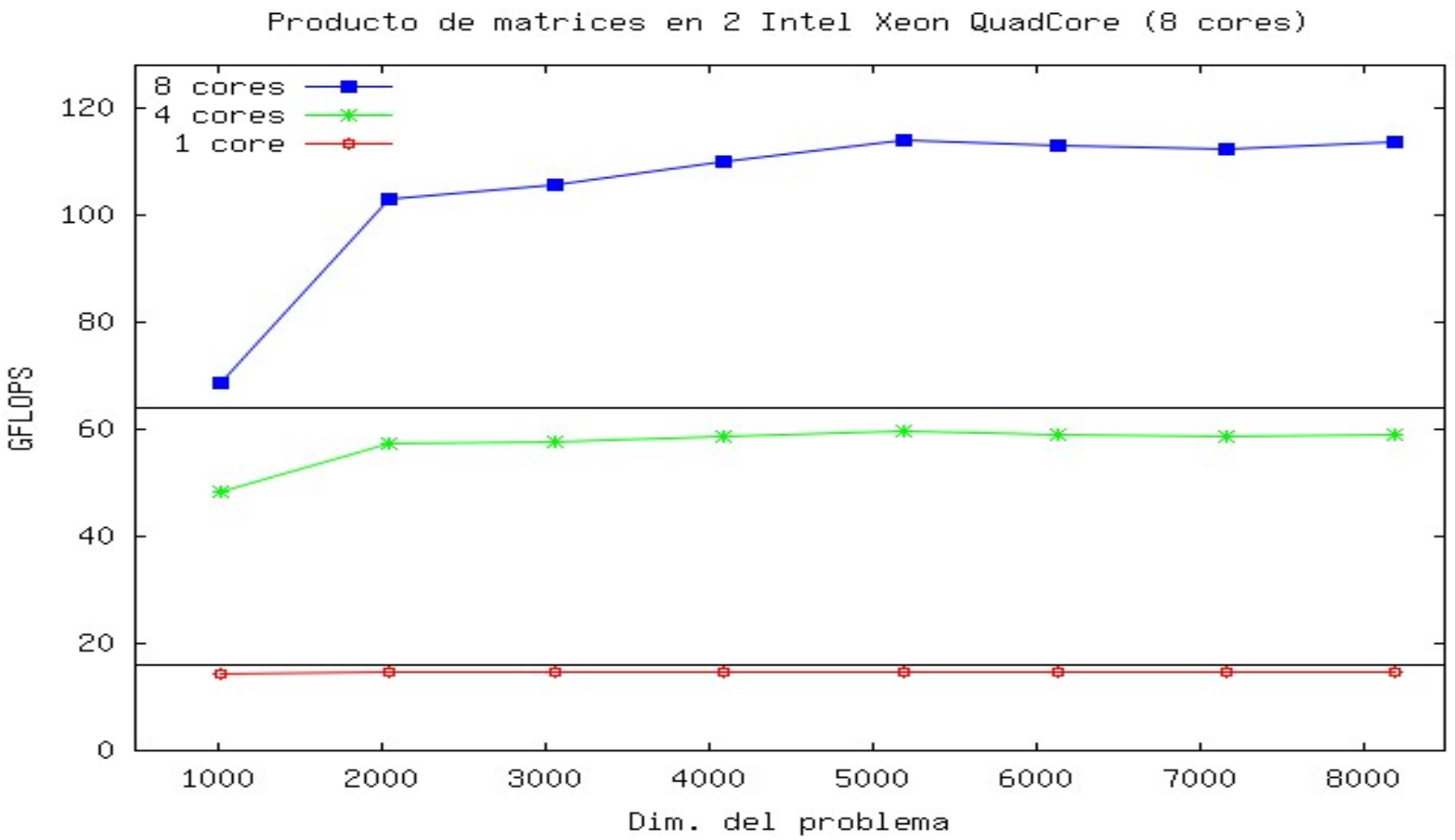

  - Muy optimizada: Intel MKL, AMD ACML, IBM ESSL, SUN SPL, NVIDIA CUBLAS,…
  - También con versiones paralelas (multihebra) para procesadores multinúcleo

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo




**92% pico**

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo



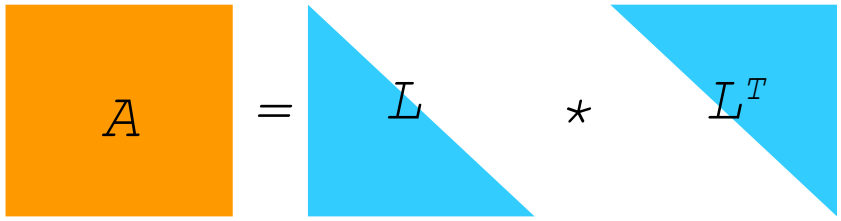

**89% pico**

**92% pico**

**92% pico**

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo

- Una operación compleja: factorización de Cholesky




Clave en la resolución de sistemas lineales (s.p.d.)

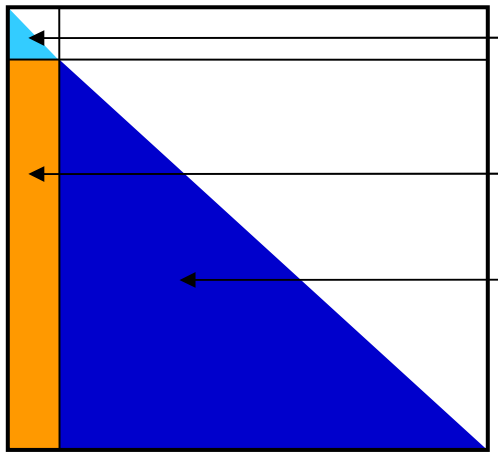
$$A\ x = b \equiv \quad (LL^T)x = b$$

$$L\ y = b \Rightarrow y$$

$$L^T x = y \Rightarrow x$$

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo

- Cálculo de la factorización de Cholesky basado en el producto de matrices



1ª iteración

F: $A_{11} = L_{11} * L_{11}^T$

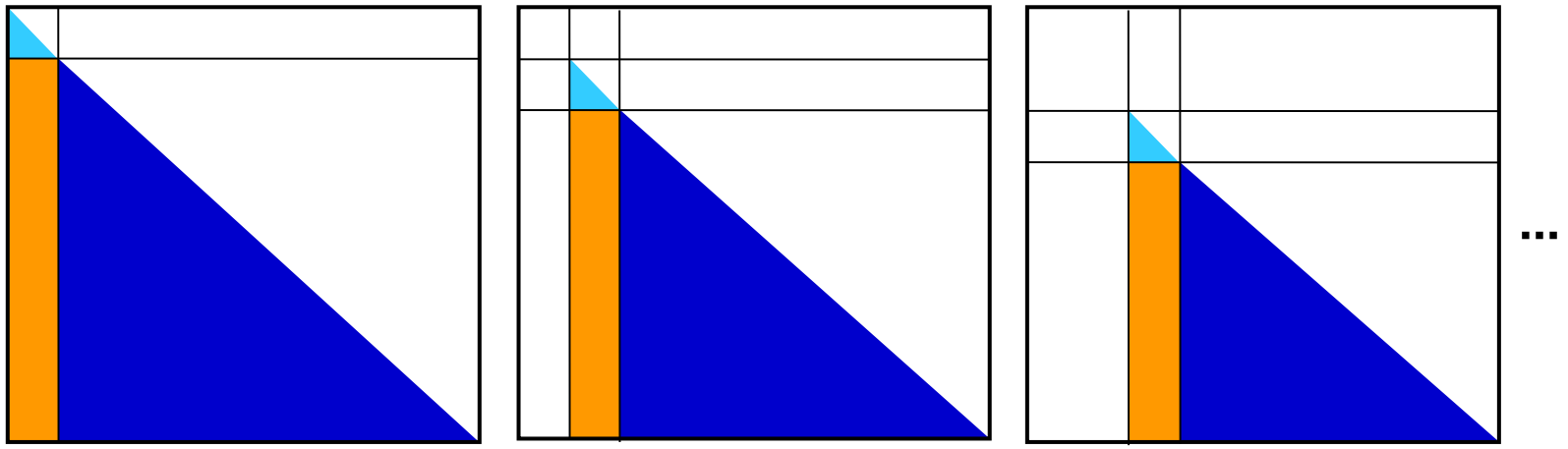
T: $L_{21} \leftarrow A_{21} * L_{11}^{-T}$

P: $A_{22} \leftarrow A_{22} - L_{21} * L_{21}^T$

Procesador multinúcleo: usar una implementación paralela (multihebra) del T y, especialmente, P

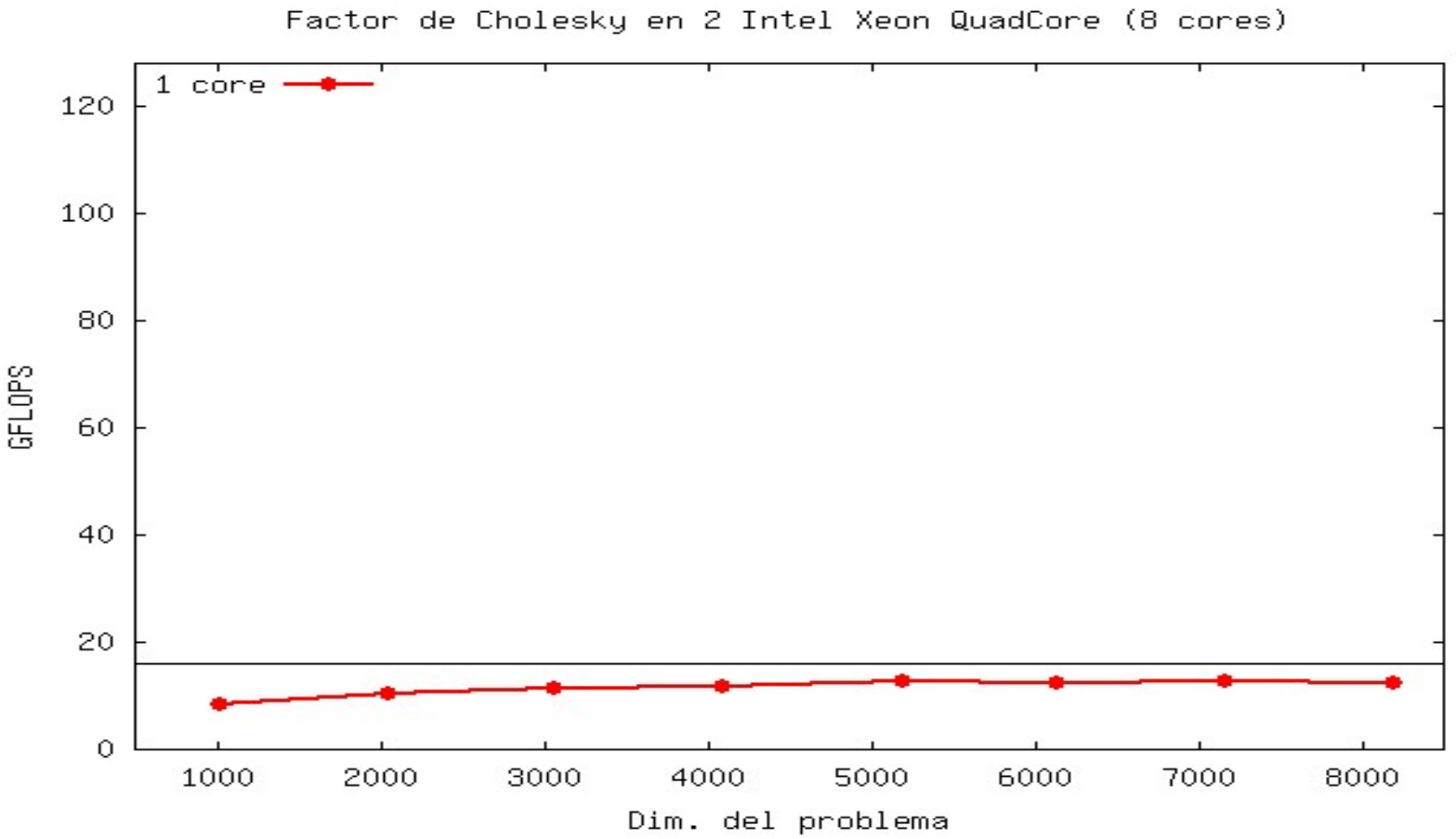
# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo

- Cálculo de la factorización de Cholesky basado en el producto de matrices



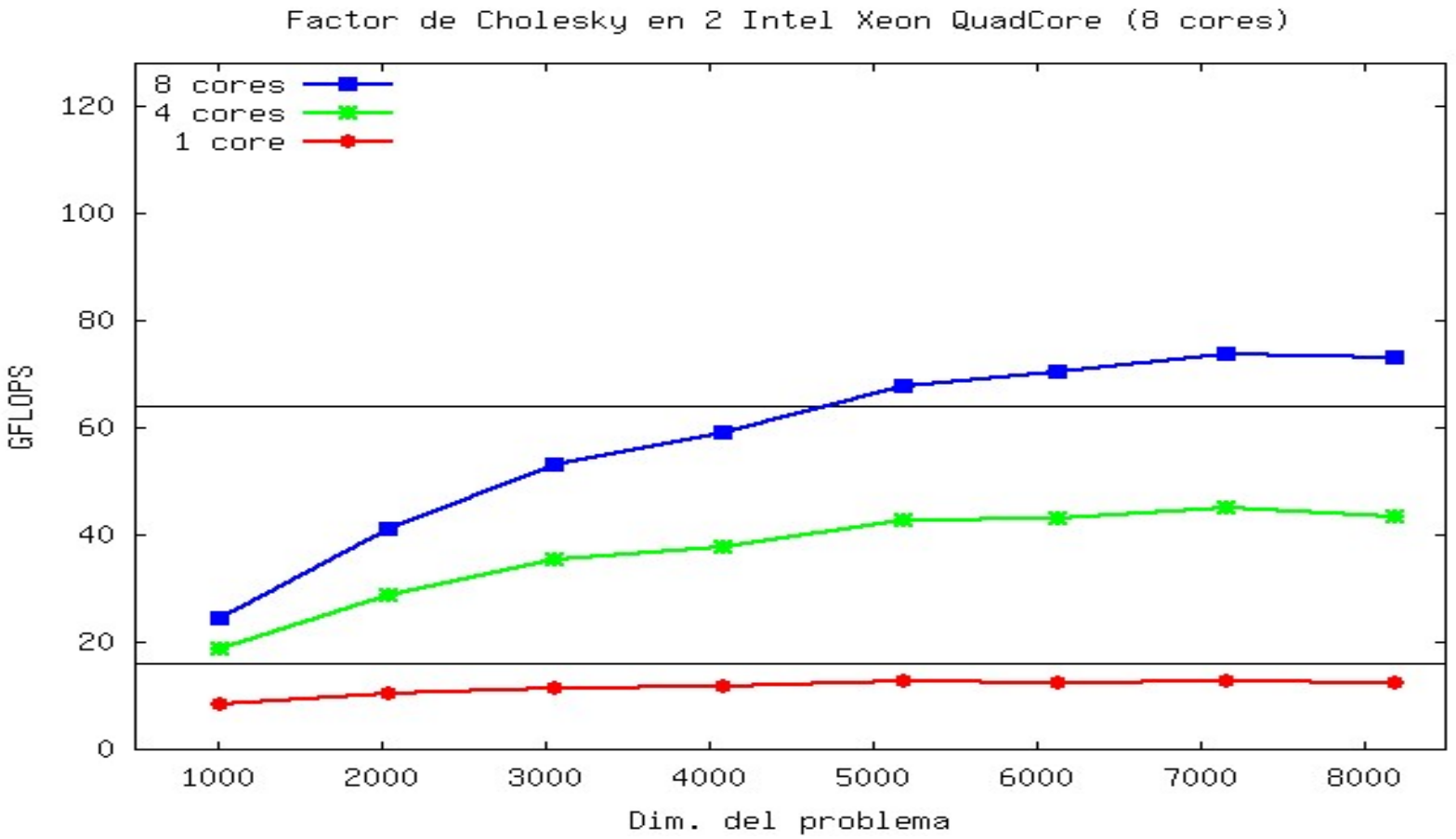
1ª iteración 2ª iteración 3ª iteración

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo




**80% pico**

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo




**57% pico**

**71% pico**

**80% pico**

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo

- ¿Por qué?

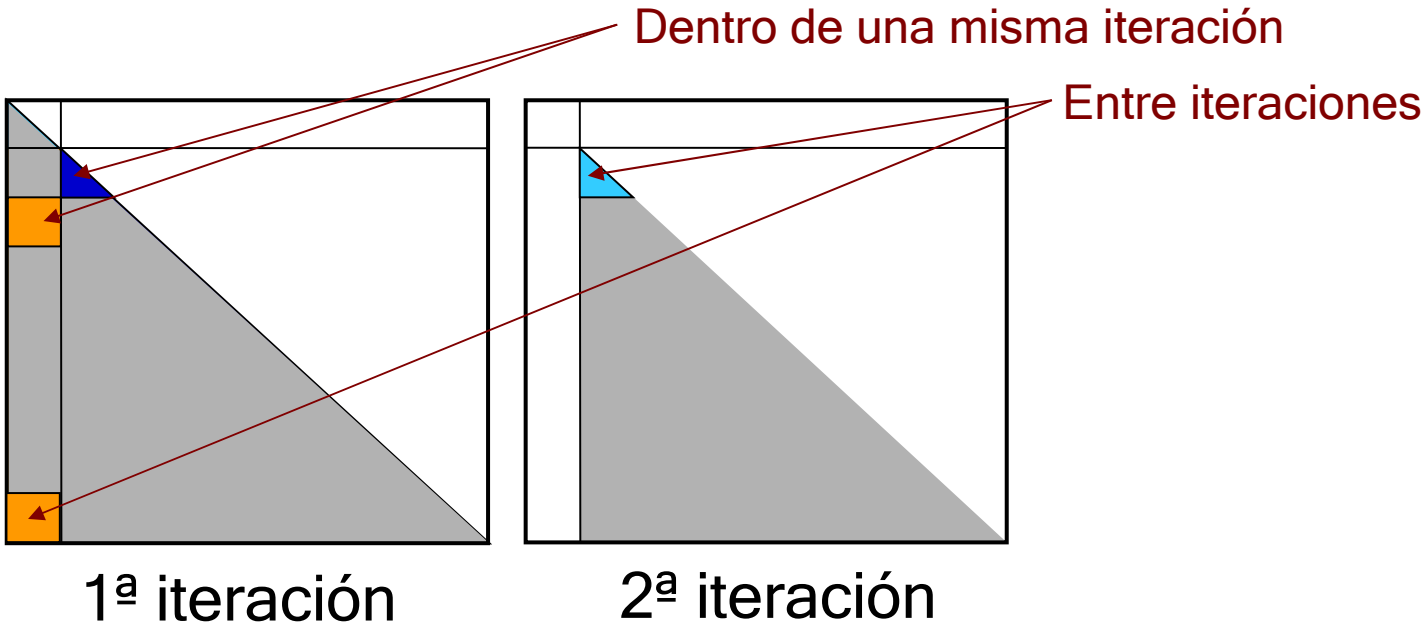
Excesiva sincronización de hebras

```
for (k=0; k<nb; k++){
  F:  Chol(A[k,k]);                          // A_kk      = L_kk * L_kk^T
    if (k<nb){
  T:  Trsm(A[k,k], A[k+1,k]);                // L_k+1,k   <-A_k+1,k * L_kk^-T
  P:  Syrk(A[k+1,k], A[k+1,k+1]);            // A_k+1,k+1 <-A_k+1,k+1
                                             //           - L_k+1,k * L_k+1,k^T
    }
}
```

# BIBLIOTECAS ACTUALES
## Rendimiento en procesadores multinúcleo

- ¿Por qué?

Hay más paralelismo del que se está aprovechando



# PARALELISMO DE FLUJO DE DATOS

- Ejecución fuera de orden dirigida por las dependencias de datos (*data-flow parallelism*)

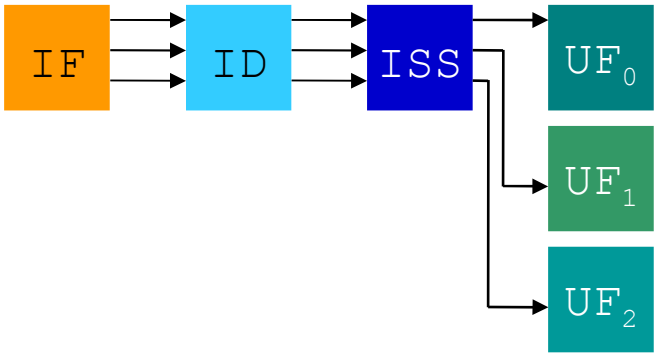
- Objetivo: Aumentar el paralelismo durante la ejecución de algoritmos de computación matricial

# PARALELISMO DE FLUJO DE DATOS
## Procesadores superescalares

Código escalar

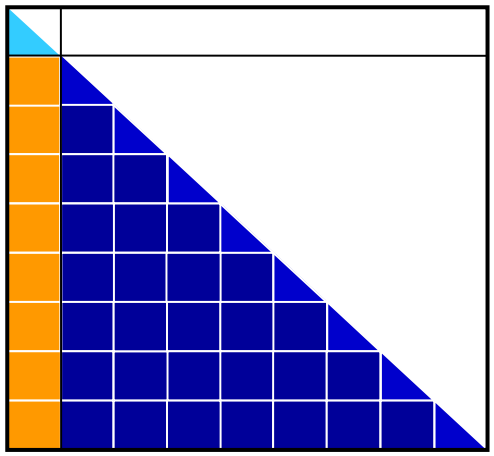
```
loop: ld   f0, 0(r1)
      addd f4, f0, f2
      sd   f4, 0(r1)
      addi r1, r1, #8
      subi r2, r2, #1
      bnez r2, loop
```

Procesador superescalar



# PARALELISMO DE FLUJO DE DATOS
## Computación matricial

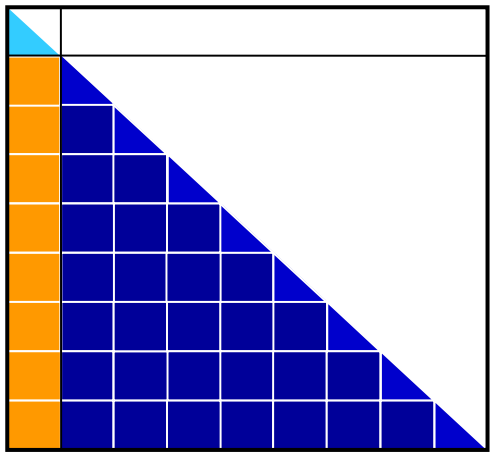
- ¿Algo parecido para computación matricial?



```
for (k=0; k<nb; k++){
F:  Chol(A[k,k]);
  for (i=k+1; i<nb; i++)
T:  Trsm(A[k,k], A[i,k]);
  for (i=k+1; i<nb; i++){
P:  Syrk(A[i,k],A[i,i]);
    for (j=k+1; j<i; j++)
P:    Gemm(A[i,k], A[j,k], A[i,j]);
  }
}
```

# PARALELISMO DE FLUJO DE DATOS
## Computación matricial

- ¿Algo parecido para computación matricial?



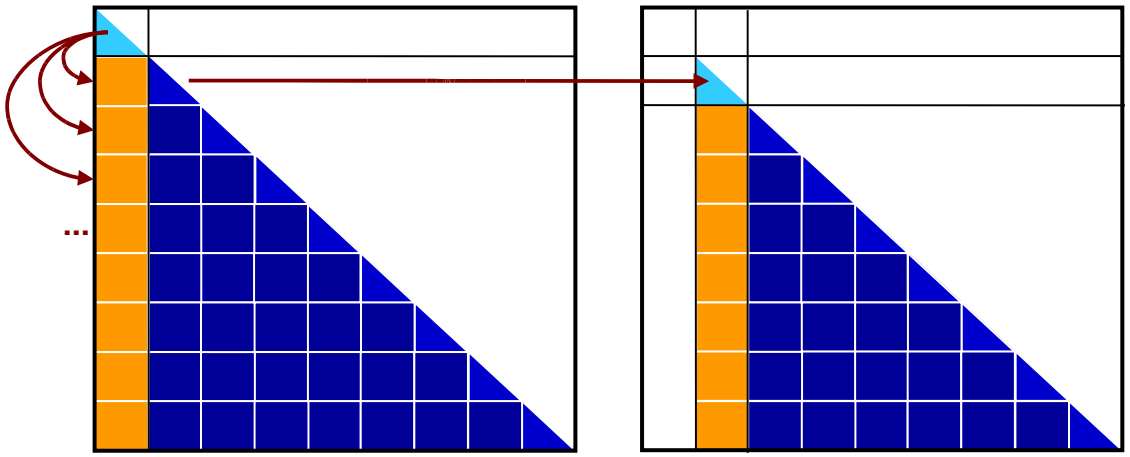
- Aplicación de las técnicas a nivel de “bloque”
- Implementación mediante *software*
- Paralelismo a nivel de tarea/hebra
- Objetivo: núcleos del procesador

# PARALELISMO DE FLUJO DE DATOS
## Computación matricial

- Los bloques leídos/escritos fijan las dependencias, igual que con las instrucciones del código escalar:

```
loop:  ld   f0, 0(r1)          for (k=0; k<nb; k++){
       addd f4, f0, f2           Chol(A[k,k]);
       sd   f4, 0(r1)            for (i=k+1; i<nb; i++)
       addi r1, r1, #8 …           Trsm(A[k,k], A[i,k]); …
```

- Las dependencias entre bloques definen un árbol de tareas:



# PARALELISMO DE FLUJO DE DATOS
## Computación matricial

- Código por bloques:

```
for (k=0; k<nb; k++){
  Chol(A[k,k]);
  for (i=k+1; i<nb; i++)
    Trsm(A[k,k], A[i,k]); …
```

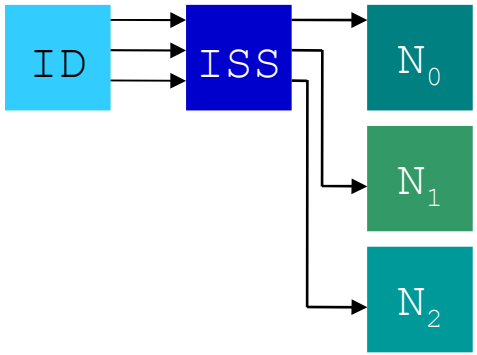
→

Procesador multinúcleo


- ¿Cómo generamos el árbol de tareas?
- ¿Qué debemos tener en cuenta para ejecutar las tareas del árbol?

# PARALELISMO DE FLUJO DE DATOS
## Computación matricial

- Uso de un “entorno de ejecución” (*runtime*):
  - Decodificación (`ID`): Generación del árbol de tareas mediante el análisis simbólico del código en tiempo de ejecución
  - Emisión (`ISS`): Ejecución del árbol de tareas consciente de las dependencias y de la arquitectura



# PARALELISMO DE FLUJO DE DATOS
## Computación matricial

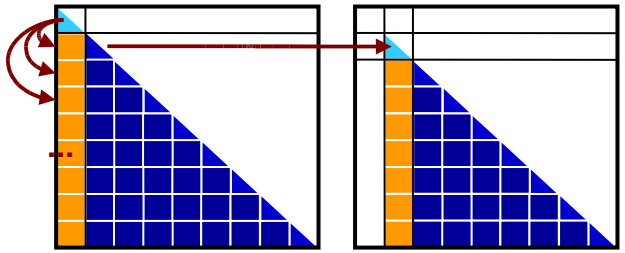
- Etapa de decodificación: análisis simbólico del código

Código por bloques:

```
for (k=0; k<nb; k++){
  Chol(A[k,k]);
  for (i=k+1; i<nb; i++)
    Trsm(A[k,k], A[i,k]); …
```

→

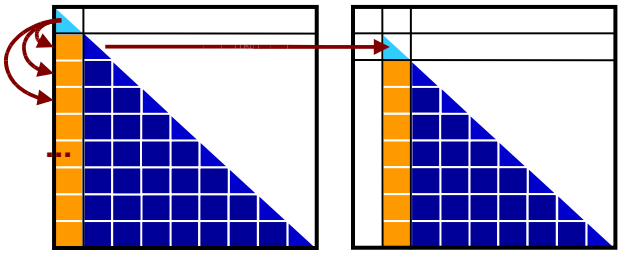
Árbol de tareas:



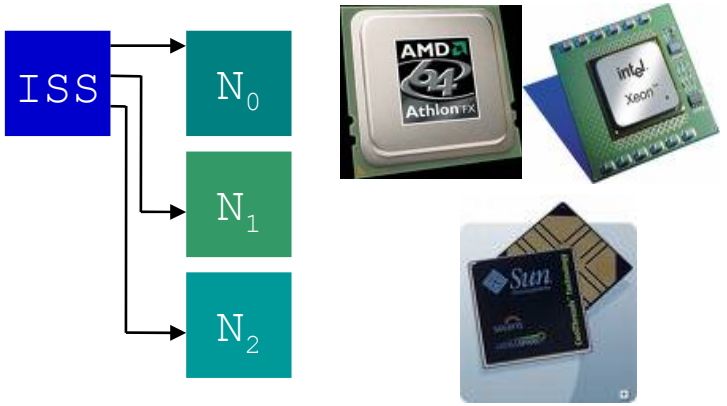
# PARALELISMO DE FLUJO DE DATOS
## Computación matricial

- Etapa de emisión:
  - Planificación temporal de tareas en función de las dependencias
  - Asignación espacial de tareas a núcleos consciente de la localidad espacial

Árbol de tareas:



Procesador multinúcleo:



# PARALELISMO DE FLUJO DE DATOS
## Implementaciones del *runtime*

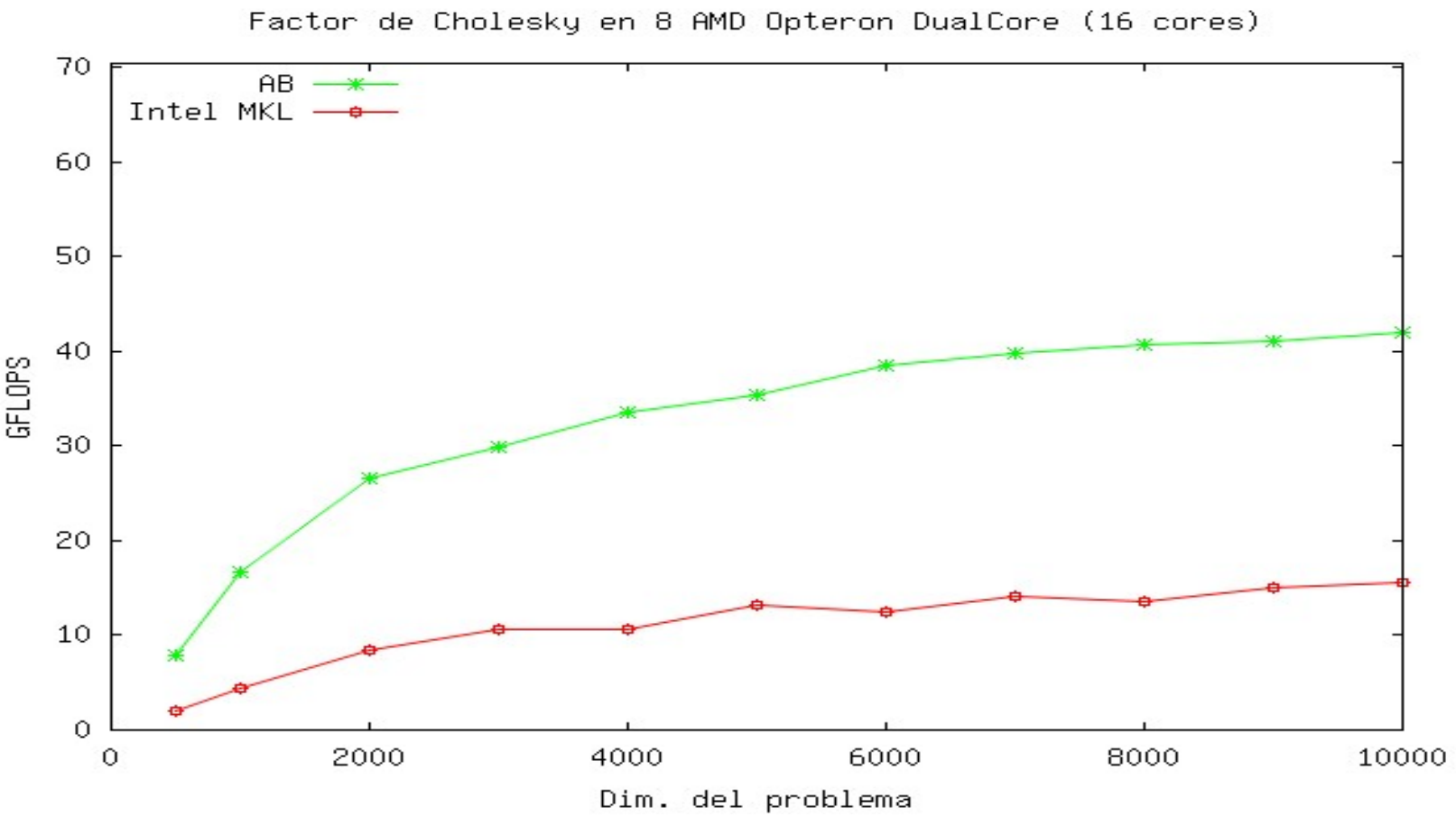
- SuperMatrix (proyecto FLAME de UT@Austin y UJI)
  - Bloques de lectura/escritura definidos intrínsecamente por las operaciones
  - Sólo válido para computación matricial

- SMPSs (proyecto StarSs del BSC)
  - Bloques de lectura/escritura definidos explícitamente

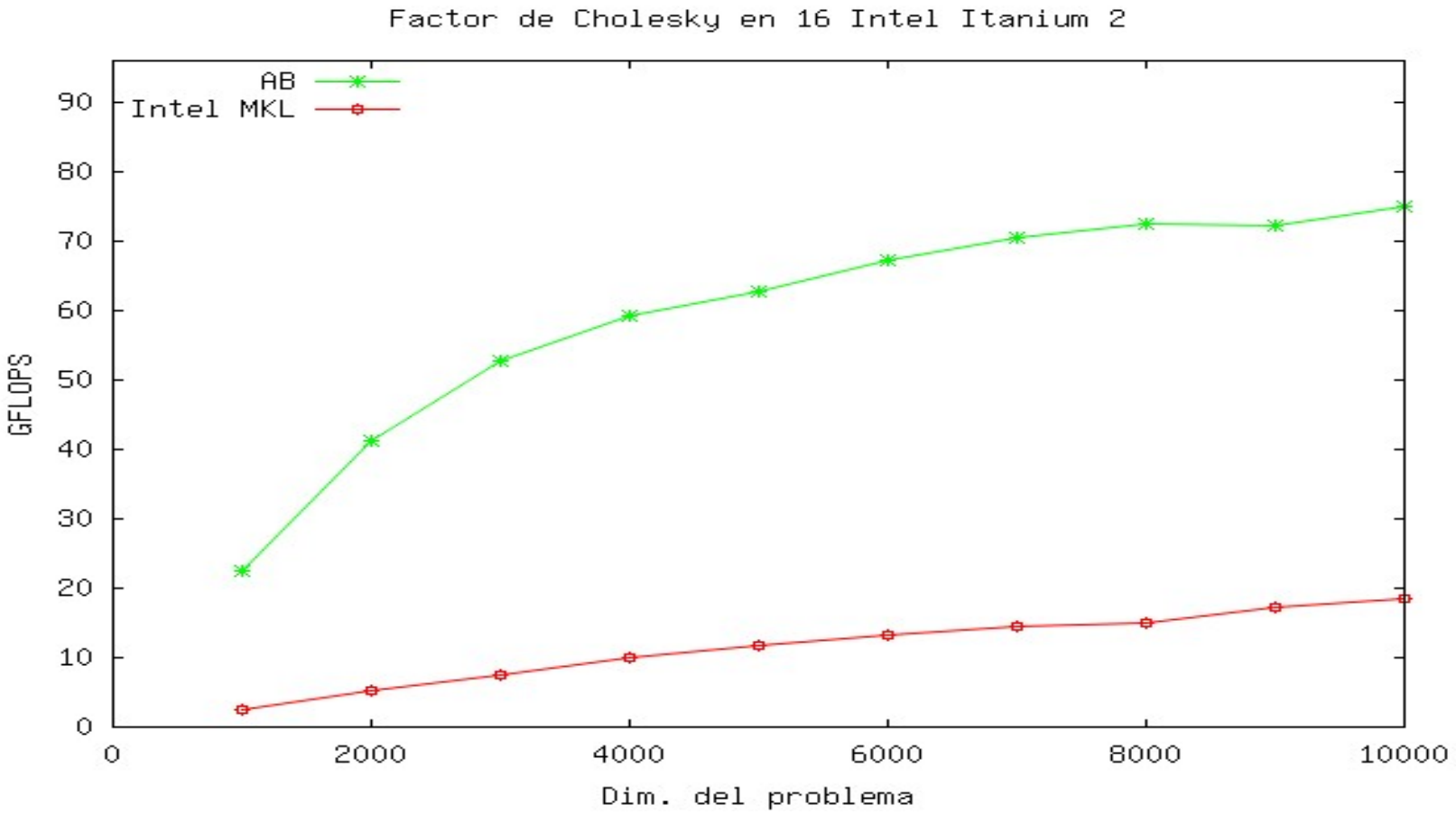
```
#pragma css task inout(A[b*b])
void Chol(double *A);
```

  - Válido para códigos con paralelismo de tareas

# PARALELISMO DE FLUJO DE DATOS
## Rendimiento en procesadores multinúcleo



# PARALELISMO DE FLUJO DE DATOS
## Rendimiento en procesadores multinúcleo

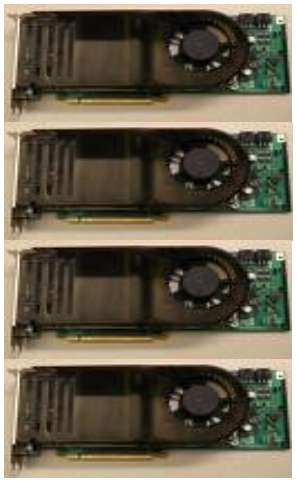


# TÉCNICAS SUPERESCALARES
## Índice

- Introducción
- Técnicas superescalares para la construcción de bibliotecas de computación matricial sobre procesadores multinúcleo y GPUs:
  1. Ejecución paralela dirigida por las dependencias de datos
  2. Uso de cachés *software* para ocultar las existencia de múltiples espacios de direcciones (DSM)
  3. Uso de cachés *software* para ocultar la latencia de acceso a disco

# ARQUITECTURAS ACTUALES
## Sistemas heterogéneos

- CPU-Acelerador(es):
  - Mejores ratios precio/consumo-rendimiento
    - Comunicaciones lentas entre *host* y dispositivos
    - El *host* y cada dispositivo tienen su propio espacio de direccionamiento
    - Sin *hardware* de coherencia

# DSM EN SISTEMAS HETEROGÉNEOS

- DSM (*distributed-shared memory*) para computación matricial

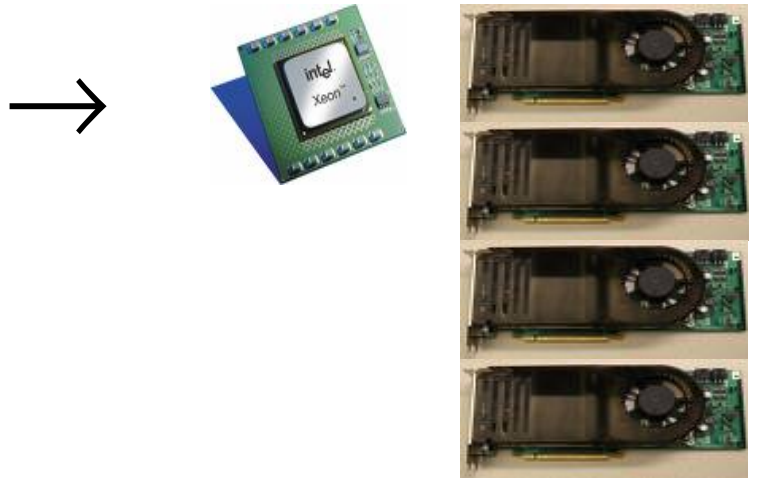
- Objetivos:
  - Ocultar la existencia de múltiples espacios de direccionamiento (facilitar la programación)
  - Hacerlo de forma eficiente (mejorar el rendimiento)

# DSM EN SISTEMAS HETEROGÉNEOS

- Asignación de tareas a “núcleos” heterogéneos

Código por bloques:

```
for (k=0; k<nb; k++){
  Chol(A[k,k]);
  for (i=k+1; i<nb; i++)
    Trsm(A[k,k], A[i,k]); …
```



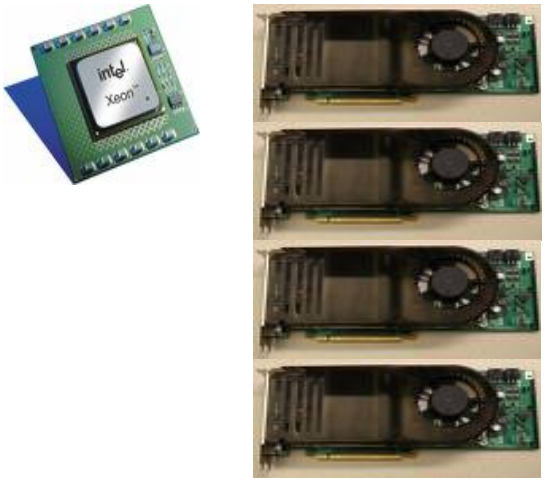
Sistema heterogéneo

# DSM EN SISTEMAS HETEROGÉNEOS

- Transferencia de datos
  - Antes de iniciar los cálculos en un dispositivo libre, se transfieren los datos
  - Cuando termina la ejecución, se recuperan los resultados

$\rightarrow$

Sistema heterogéneo

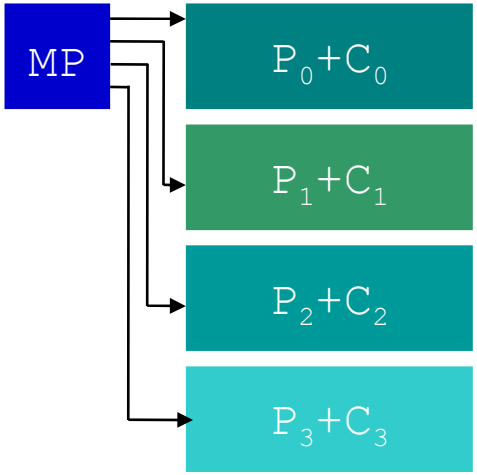



$\rightarrow$ pobre localidad de referencia

# DSM EN SISTEMAS HETEROGÉNEOS

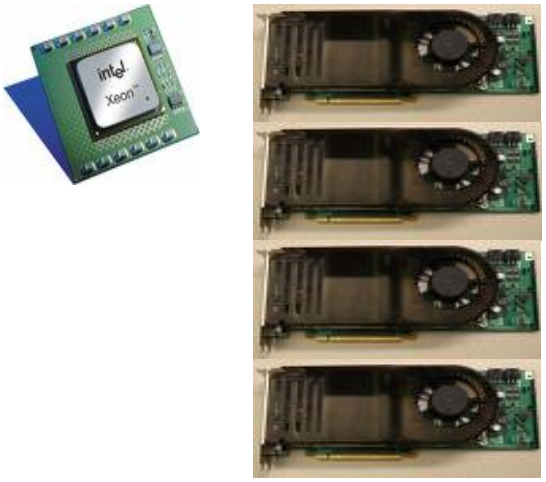
- Analogía con sistemas actuales

Sistema homogéneo con caché:




≡

Sistema heterogéneo

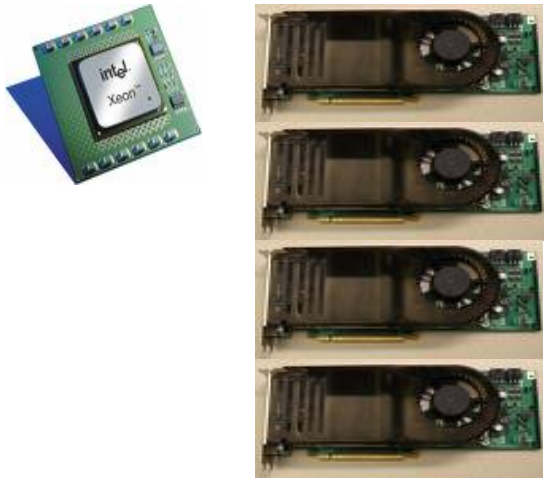


# DSM EN SISTEMAS HETEROGÉNEOS

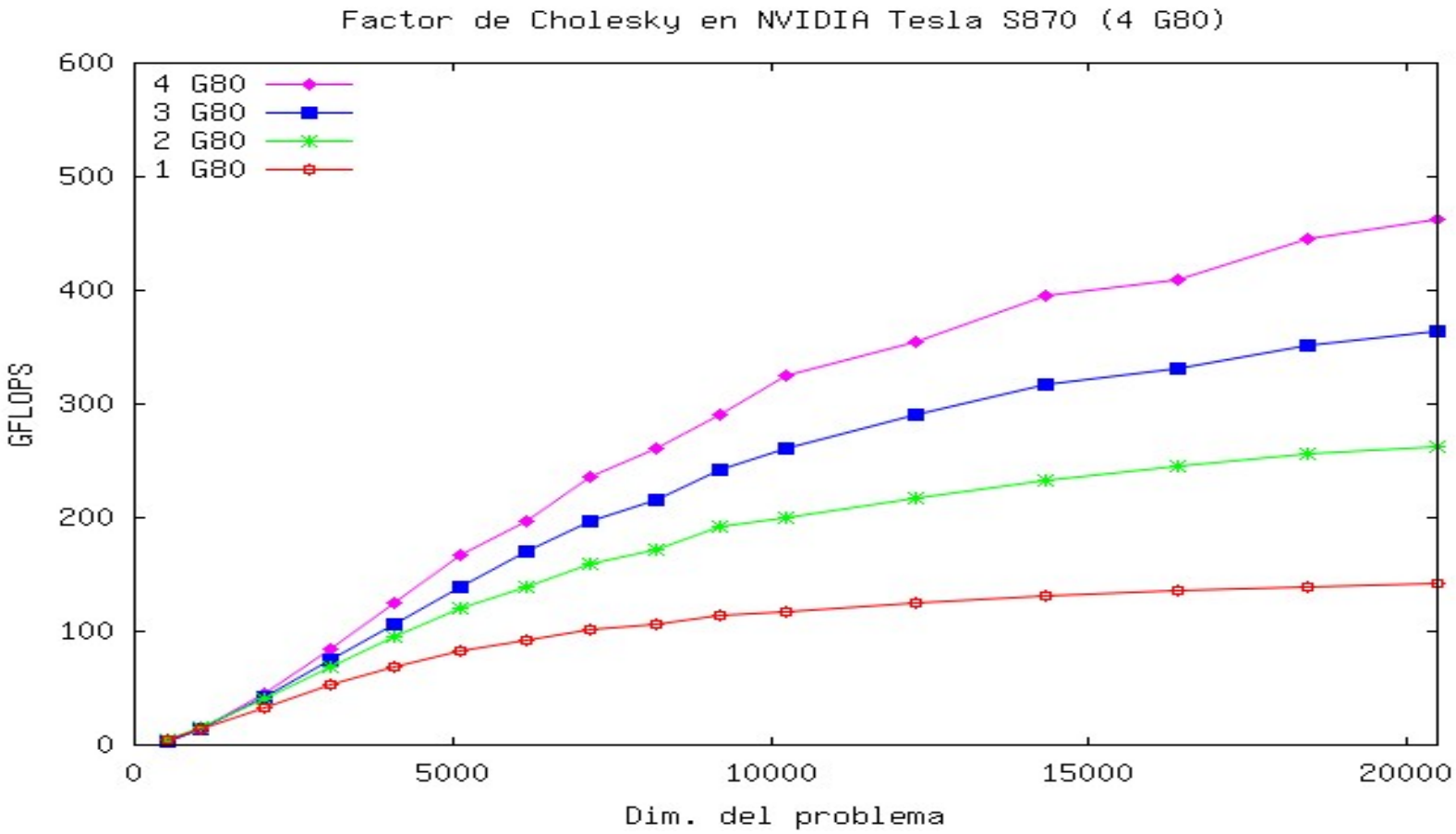
- Ahorro en las transferencias de datos
  - Caché gestionada por software en la memoria de las GPUs:
    - Trabajo sobre bloques para amortizar el coste de gestión
    - Software → más flexibilidad
    - *Write-back* para mantener la coherencia con bloques en el host
    - *Write-invalidate* para mantener la coherencia de bloques en dispositivos

→

Sistema heterogéneo



# DSM EN SISTEMAS HETEROGÉNEOS
## Rendimiento en sistemas CPU-GPUs



# TÉCNICAS SUPERESCALARES
## Índice

- Introducción
- Técnicas superescalares para la construcción de bibliotecas de computación matricial sobre procesadores multinúcleo y GPUs:
  1. Ejecución paralela dirigida por las dependencias de datos
  2. Uso de cachés *software* para ocultar las existencia de múltiples espacios de direcciones (DSM)
  3. Uso de cachés *software* para ocultar la latencia de acceso a disco

# PROBLEMAS DE GRAN DIMENSIÓN

- Algunos problemas de computación matricial densa son realmente grandes (`795k x 795k`)
- A veces el tiempo de respuesta no es absolutamente crítico
- El coste de la RAM no es despreciable, mientras que el coste del disco es menor y sus prestaciones siguen aumentando (memoria sólida)
- Los aceleradores *hardware* están aumentando considerablemente la potencia de cálculo

# EXTENSIÓN DE LA MEMORIA AL DISCO

- Uso del disco en la resolución de problemas de computación matricial
  - El S.O. no siempre gestiona bien la memoria virtual

- Objetivos:
  - Ocultar el uso del disco y de la E/S asíncrona (facilitar la programación)
  - Hacerlo de forma eficiente (mejorar el rendimiento)

# EXTENSIÓN DE LA MEMORIA AL DISCO

- Gestión de la E/S

Código por bloques:

```
for (k=0; k<nb; k++){
  Chol(A[k,k]);
  for (i=k+1; i<nb; i++)
    Trsm(A[k,k], A[i,k]); …
```

# EXTENSIÓN DE LA MEMORIA AL DISCO

- Transferencia de datos
  - Antes de iniciar los cálculos, se transfieren los datos a RAM
  - Cuando termina la ejecución, se escriben los resultados en disco

$\rightarrow$


$\rightarrow$ pobre localidad de referencia

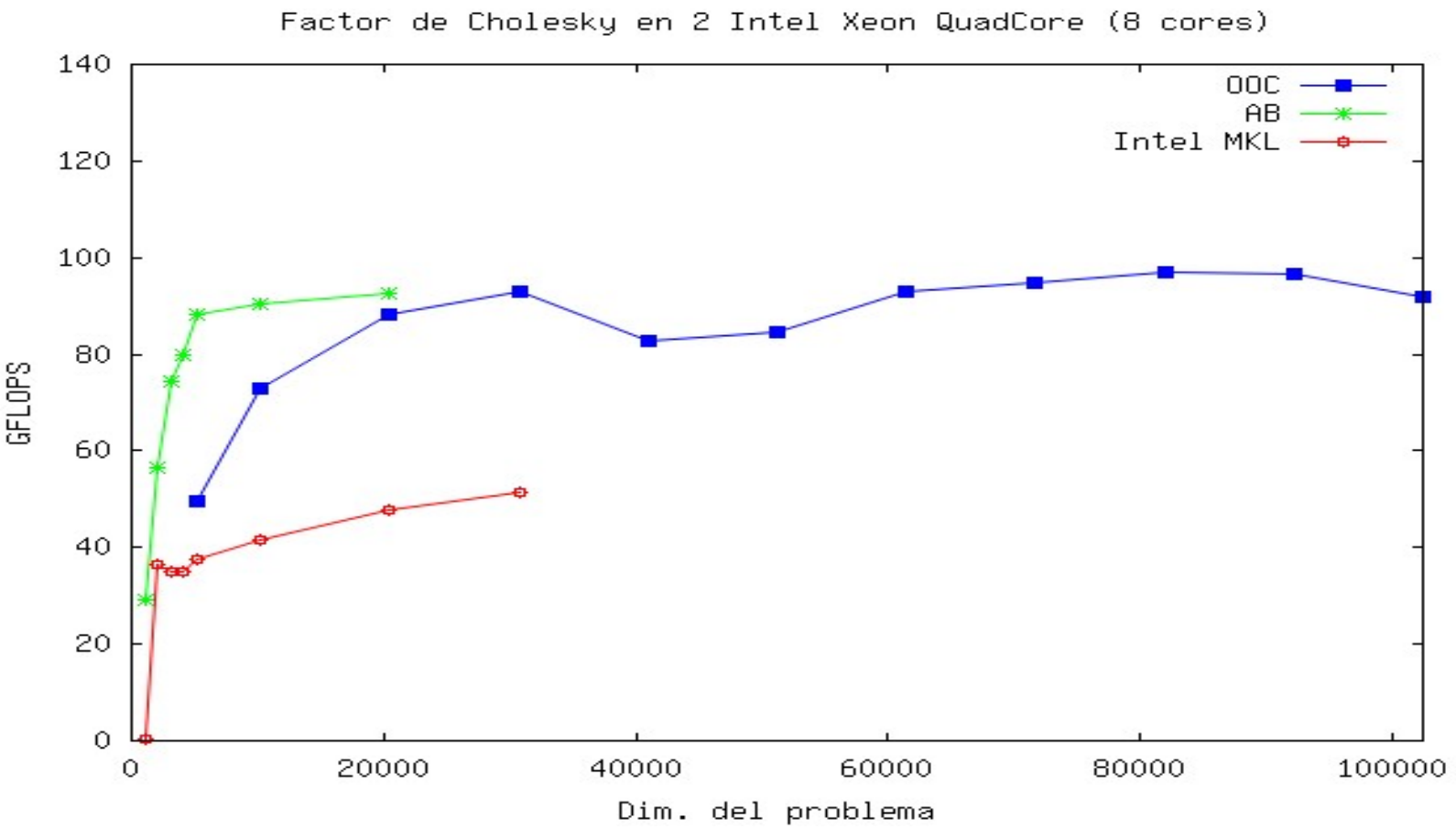
# EXTENSIÓN DE LA MEMORIA AL DISCO

- Ahorro en las transferencias de datos
  - Caché gestionada por *software* en la RAM:
    - Trabajo sobre bloques para amortizar el coste de gestión
    - Software → más flexibilidad
    - Análisis simbólico del código para determinar la lista de tareas
    - *Prefetch* perfecto

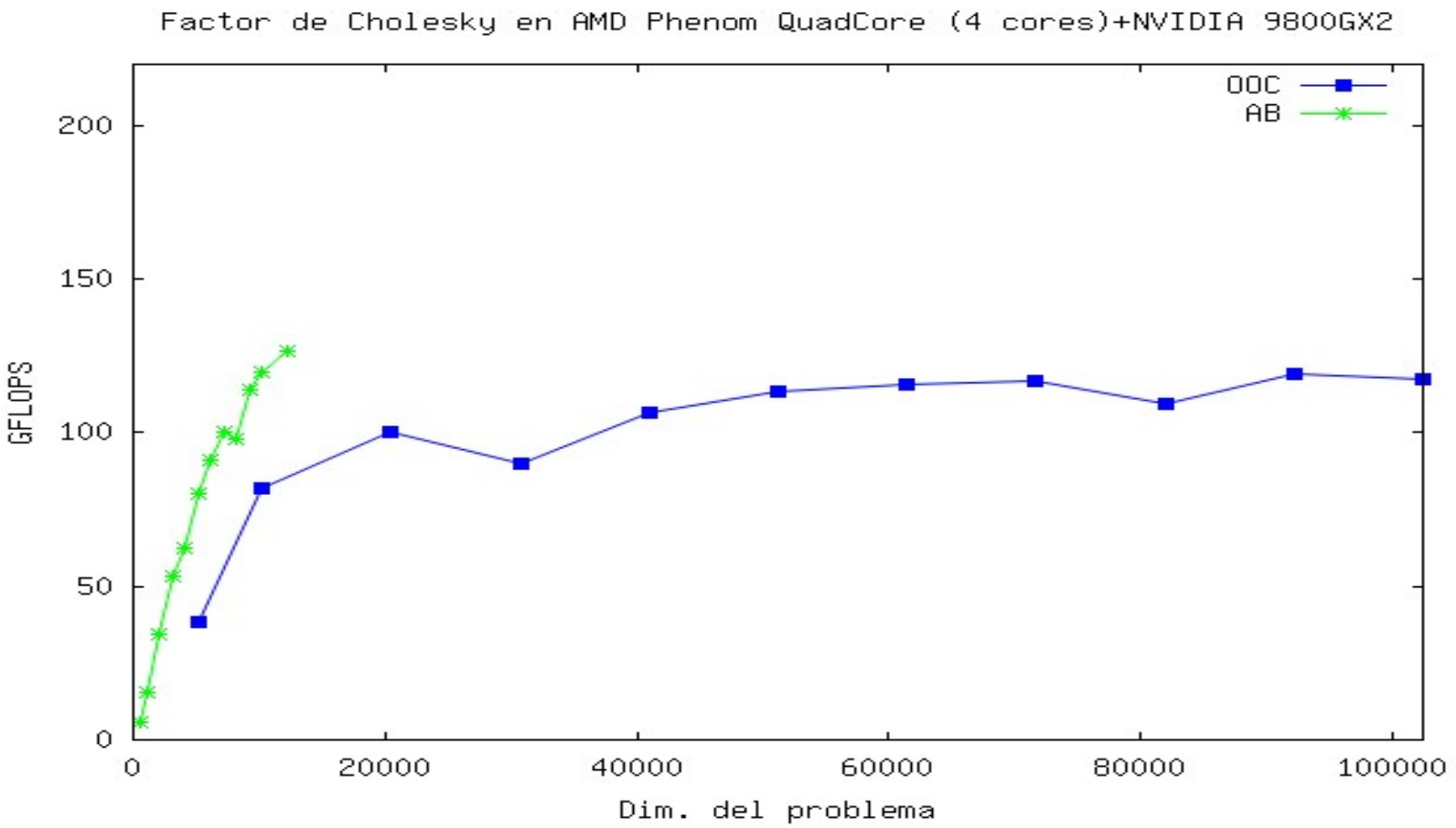
# EXTENSIÓN DE LA MEMORIA AL DISCO

- E/S asíncrona
  - Una hebra al cargo de las transferencias entre RAM (caché) y disco
    - Reemplazo LRU
  - Restantes hebras encargadas de la ejecución paralela con datos en RAM
  - Almacenamiento de matrices por bloques en disco

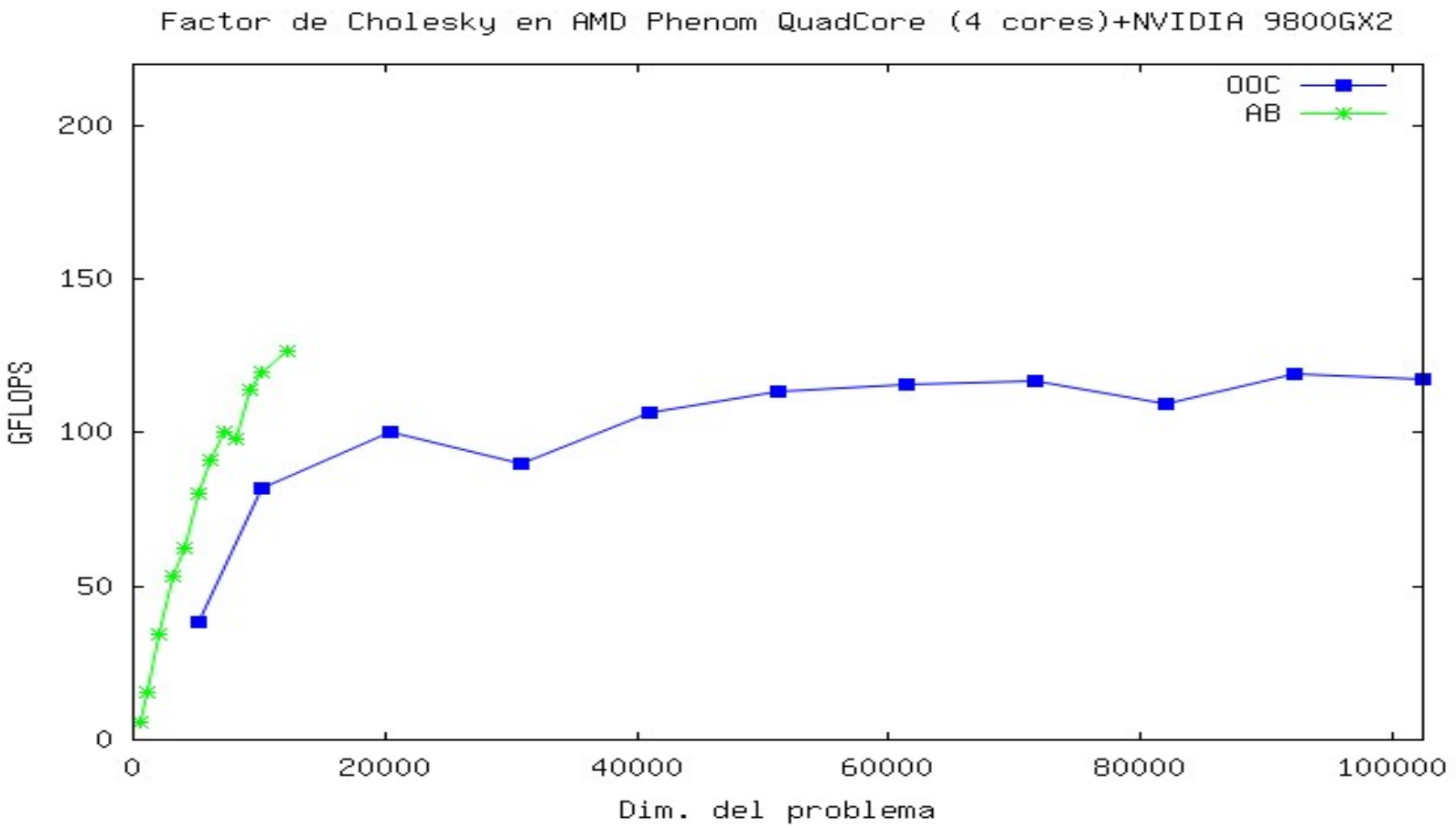
# EXTENSIÓN DE LA MEMORIA AL DISCO
## Rendimiento en procesadores multinúcleo



# EXTENSIÓN DE LA MEMORIA AL DISCO
## Rendimiento en sistemas CPU-GPU



# EXTENSIÓN DE LA MEMORIA AL DISCO
## Rendimiento en sistemas CPU-GPU



# FUTURO
## Carta a los Reyes Magos…

- Más núcleos, sin olvidar la E/S
- Conexiones CPU-GPU más rápidas que PCI
- Comunicaciones directas entre GPUs
- Soporte hardware para coherencia
- En OOC: discos sólidos

# AGRADECIMIENTOS


- Trabajo conjunto:

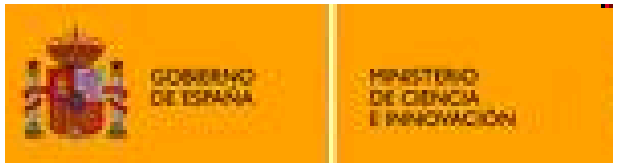
| UJI | UT@Austin |
| --- | --- |
| Francisco D. Igual | Ernie Chan |
| Mercedes Marqués | Robert van de Geijn |
| Alberto Martín | Field G. Van Zee |
| Gregorio Quintana | |

- Colaboraciones con BSC (uso de SMPSs y desarrollo de GPUSs)

# AGRADECIMIENTOS

- Financiación:

UJI






UT@Austin


¡Gracias por la atención y por las preguntas*!

* Preguntas difíciles: quintana@icc.uji.es ;-)