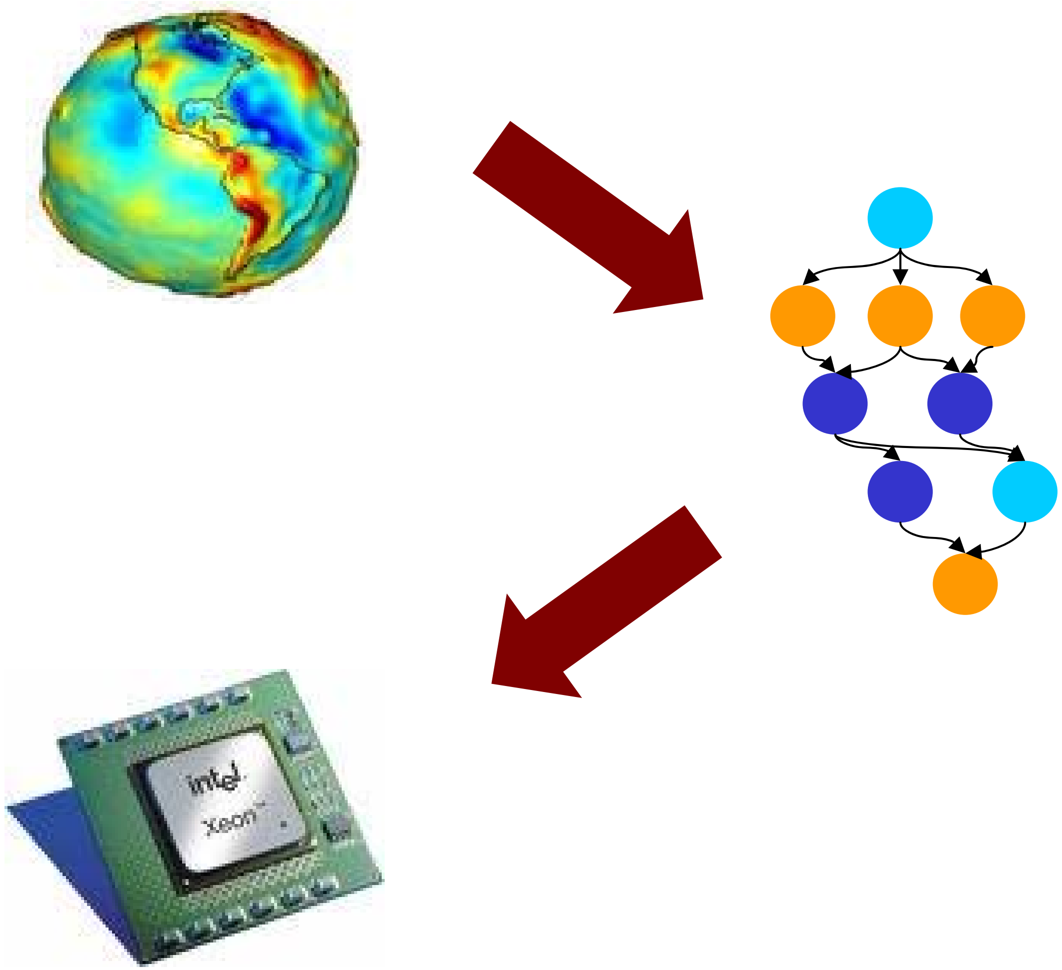
Use of Superscalar Techniques in the Construction of Linear Algebra Libraries for Multi-core Processors and GPUs

HPC&A

UNIVERSITAT JAUME·I High Performance Computing and Architectures

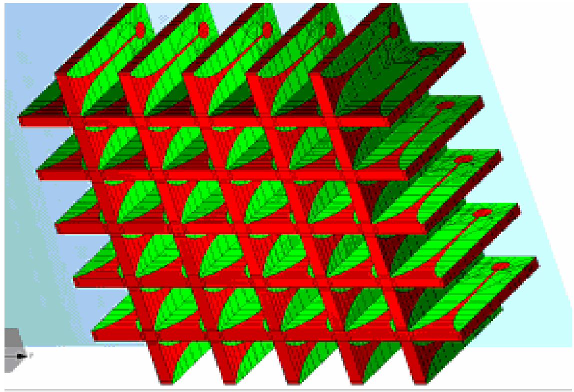
Enrique S. Quintana Ortí

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


intel
Xeon

LINEAR SYSTEMS
Simulation of electromagnetic fields

- Antenna Vivaldi



Solve

$$A\ x = b$$

A of size

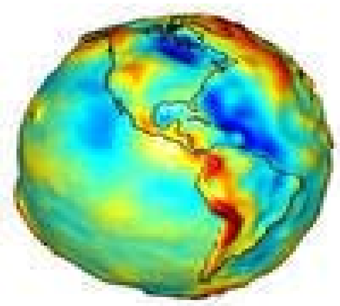
49.820 x 49.820 to

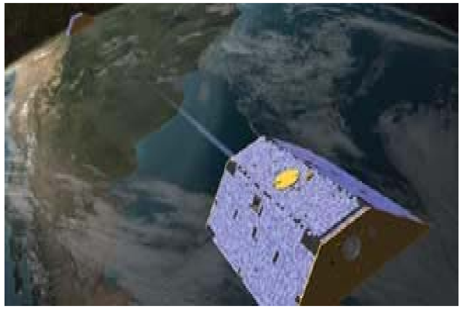
92.729 x 92.729,

depending on the operation frequency of the antenna

LINEAR LEAST SQUARES PROBLEMS
Estimation of the Earth's gravitational filed

- GRACE project





Solve

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

A of size

~130.000 x 130.000

(156 GBytes)

NUMERICAL LIBRARIES
Use

Allow the advances in other sciences

(*Computational Science*) →

Simulation replaces experimentation: lower economic and temporal costs

NUMERICAL LIBRARIES
Use

- How long would it take to solve a large-scale linear least problem under ideal conditions?
  - Algorithm limited only by the processor speed, not by the memory
  - 1 Intel Xeon processor @ 3,6 GHz (4 floating-point arithmetic operations/cycle)

`200.000 x 200.000` → 8,6 days
`400.000 x 400.000` → 68,8 days

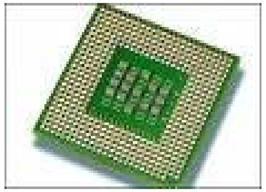
NUMERICAL LIBRARIES
High performance

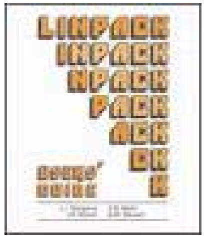
- Need an efficient execution: pursuing the “current” HPC architecture…

1980
Vector ISA

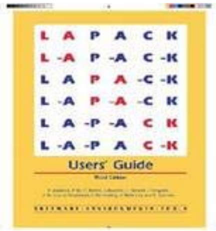
1990
Cache memory

2000
Distributed memory





LINPACK
INPACK
NPACK
PACK
ACK
CK
K
Users' Guide



L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K
Users' Guide

ScaLAPACK Users' Guide

NUMERICAL LIBRARIES
High performance

- Need an efficient execution: pursuing the *"current" HPC architecture?*

Moore's Law still valid, but…

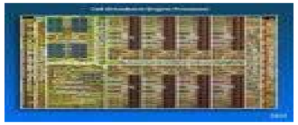
- It is not possible to increase the frecuency due to power consumption and heat

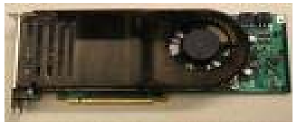
  f x 1,3 → power x 2
- Little instruction-level parallelism left
- Memory latency is high (1 memory access ≈ 240 cycles)

BIBLIOTECAS DE COMPUTACIÓN
Alto rendimiento

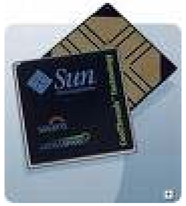
- Need an efficient execution: pursuing the *"current" HPC architecture?*

Hardware accelerators





General-purpose multi-core processors



SUPERSCALAR TECHNIQUES
Outline

- Introduction
- Superscalar techniques in the construction of linear Algebra libraries for multi-core processors and GPUs:
  1. Parallel execution dictated by data dependencies
  2. Use of *software* caches to hide the existence of multiple address spaces (DSM)
  3. Use of *software* caches to hide latency of disk access

SUPERSCALAR TECHNIQUES

Outline

- Introduction
- Superscalar techniques in the construction of linear Algebra libraries for multi-core processors and GPUs:
    - Techniques applied on “blocks”
    - S*oftware* implementation
    - Task/thread-level parallelism
    - Target: processor cores

SUPERSCALAR TECHNIQUES
Outline

- Introduction
- Superscalar techniques in the construction of linear Algebra libraries for multi-core processors and GPUs:

1. Parallel execution dictated by data dependencies
2. Use of *software* caches to hide the existence of multiple address spaces (DSM)
3. Use of *software* caches to hide latency of disk access

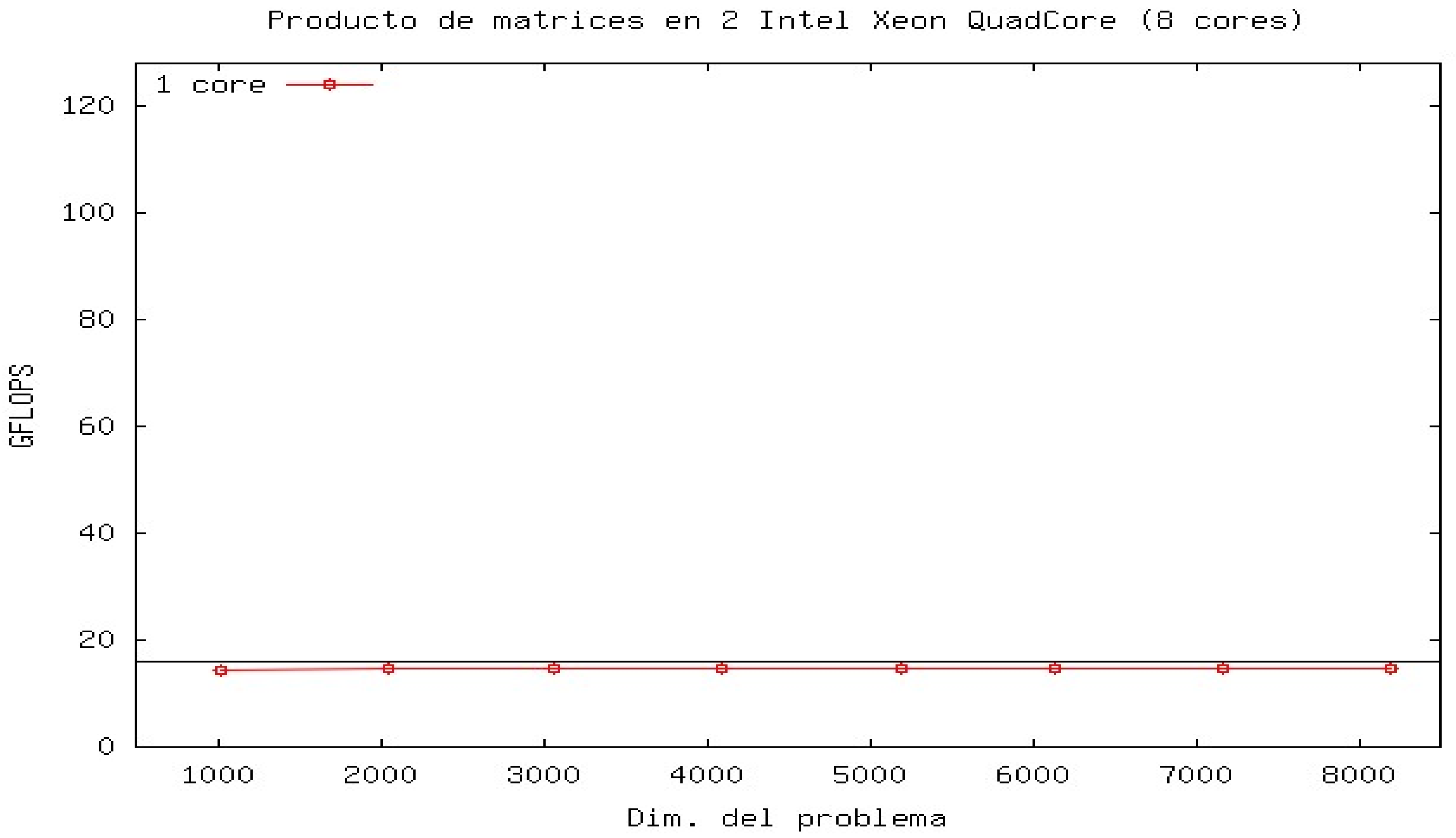
CURRENT LIBRARIES
Performance on a multi-core processor

- A simple operation: matrix-matrix product

$$C \leftarrow C + A * B$$

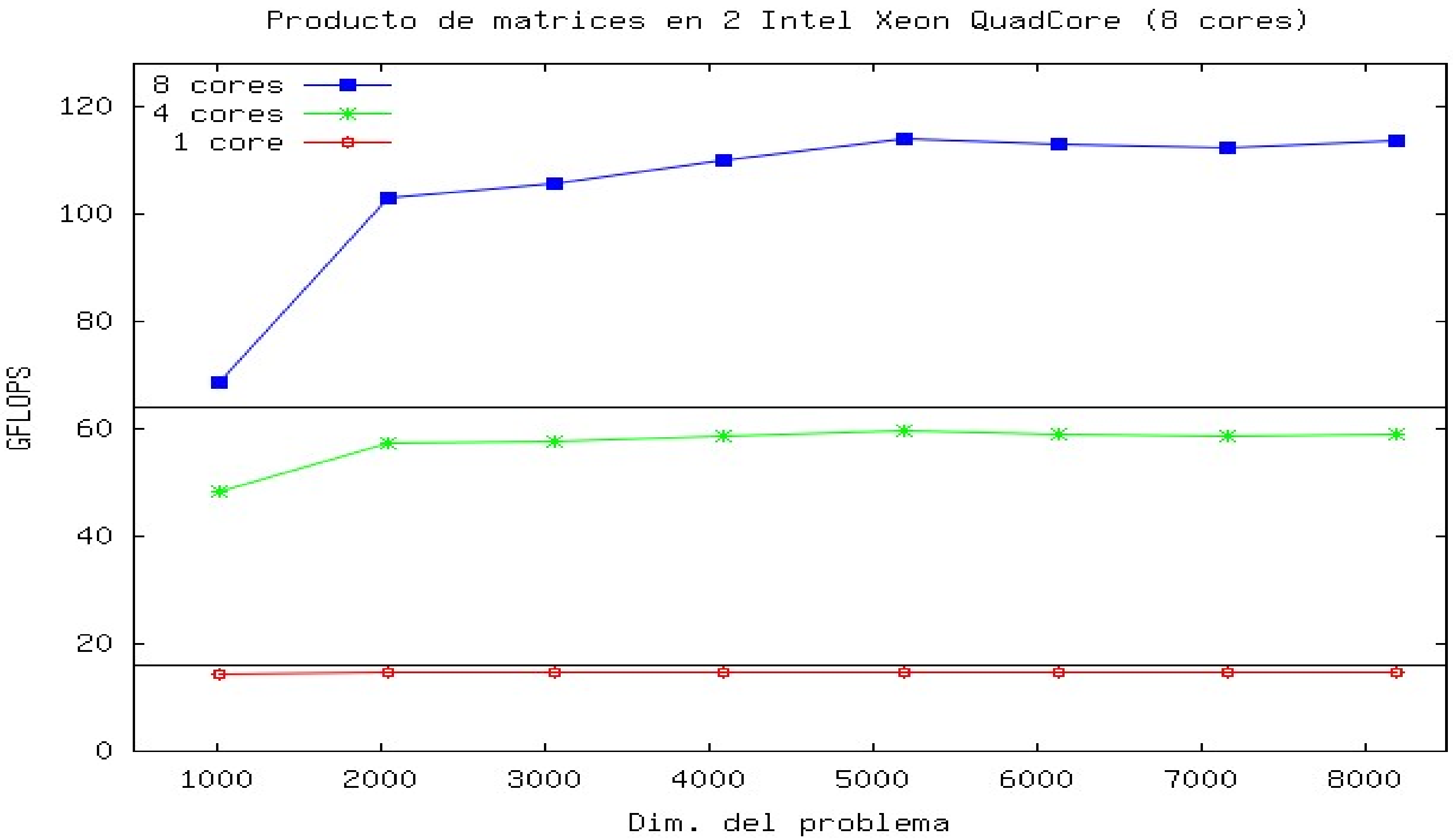
  - Highly tuned: Intel MKL, AMD ACML, IBM ESSL, SUN SPL, NVIDIA CUBLAS,…
  - Also parallel (multi-threaded) versions for multi-core processors

CURRENT LIBRARIES
Performance on a multi-core processor



92% peak

CURRENT LIBRARIES
Performance on a multi-core processor



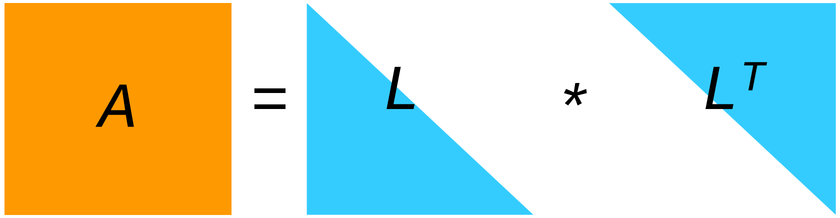
89% peak

92% peak

92% peak

CURRENT LIBRARIES
Performance on a multi-core processor

- A more complex operation: Cholesky factorization



Key in the solution of (s.p.d.) linear systems

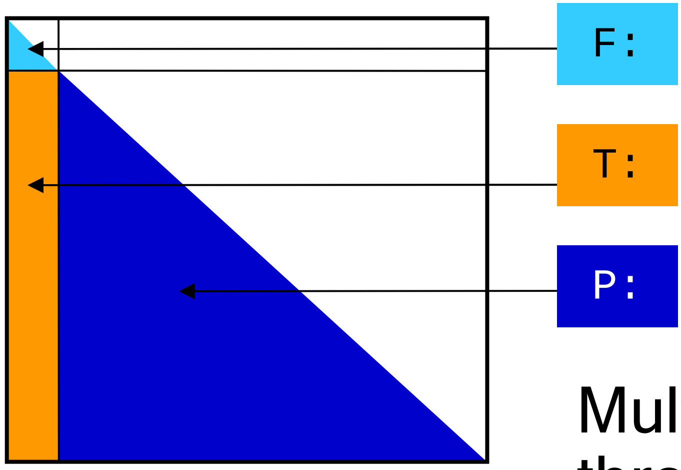
$$A\ x = b \equiv \qquad (LL^T)x = b$$

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

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

CURRENT LIBRARIES
Performance on a multi-core processor

- Blocked algorithm for the Cholesky factorization based on matrix-matrix product



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

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

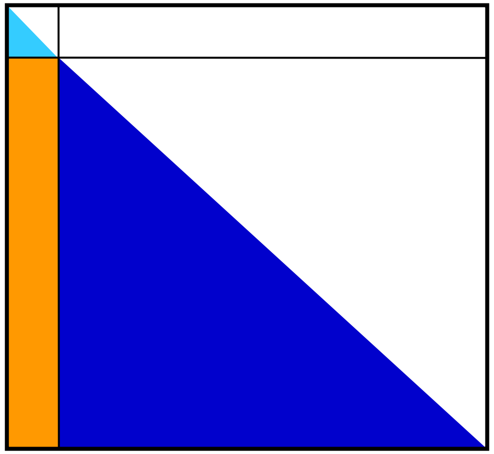
P: $A_{22} \leftarrow A_{22} - L_{21} * L_{21}^{T}$

1st iteration

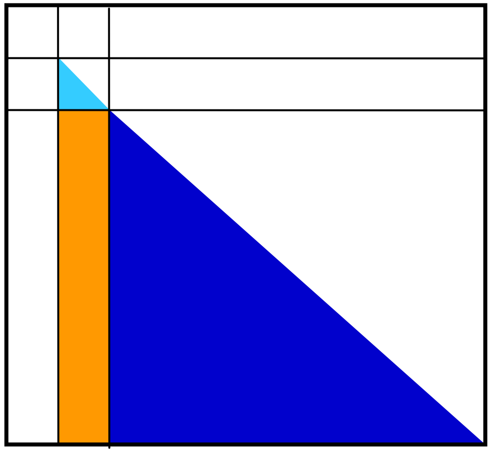
Multi-core processor: use a (multi-threaded) implementation for T and, specially, P

CURRENT LIBRARIES
Performance on a multi-core processor

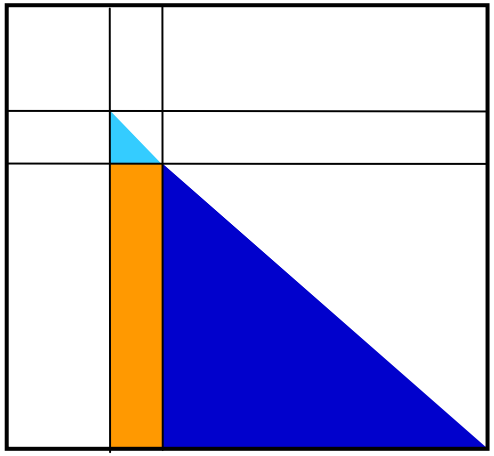
- Blocked algorithm for the Cholesky factorization based on matrix-matrix product



1st iteration



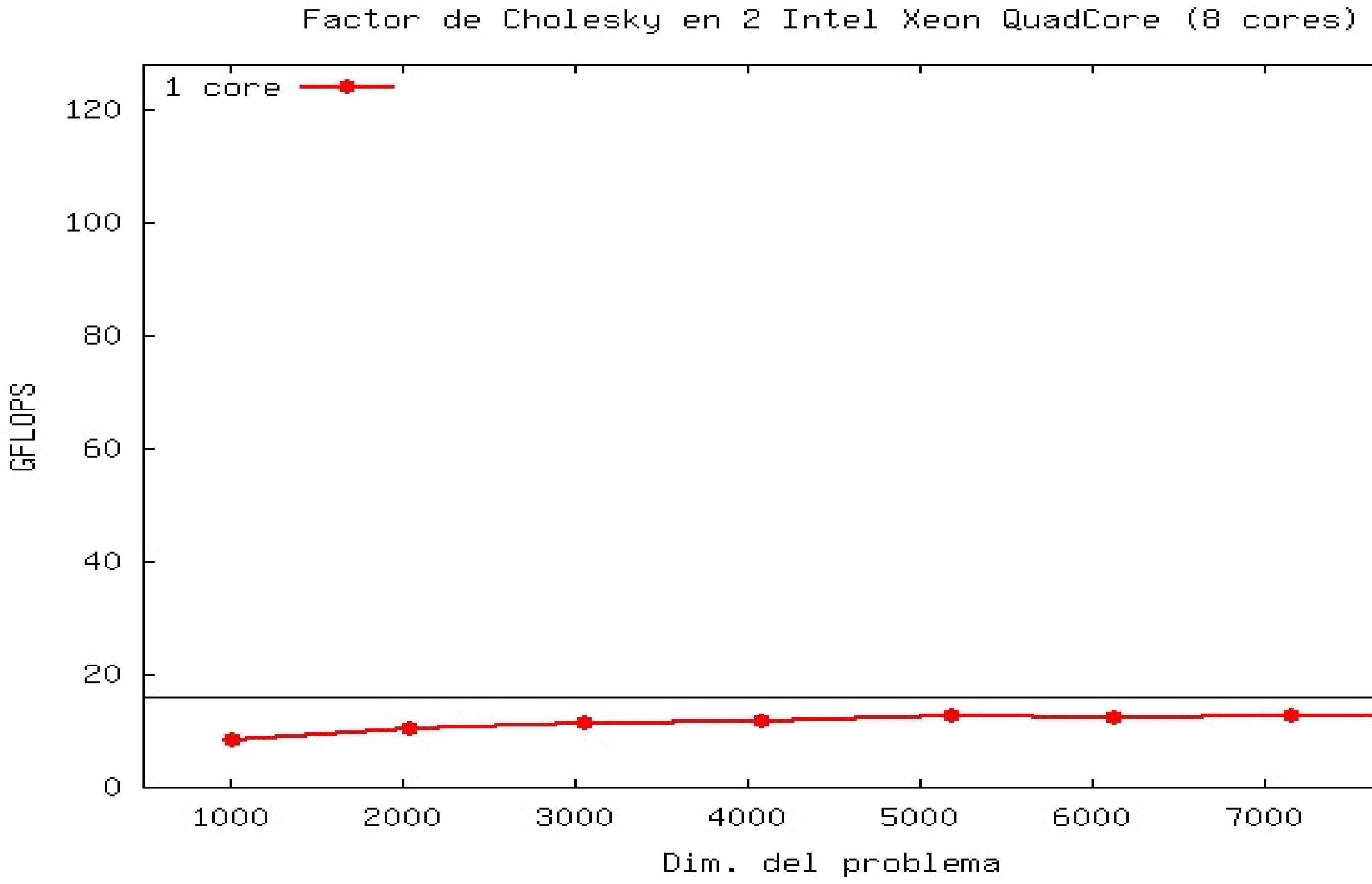
2nd iteration



3rd iteration

...

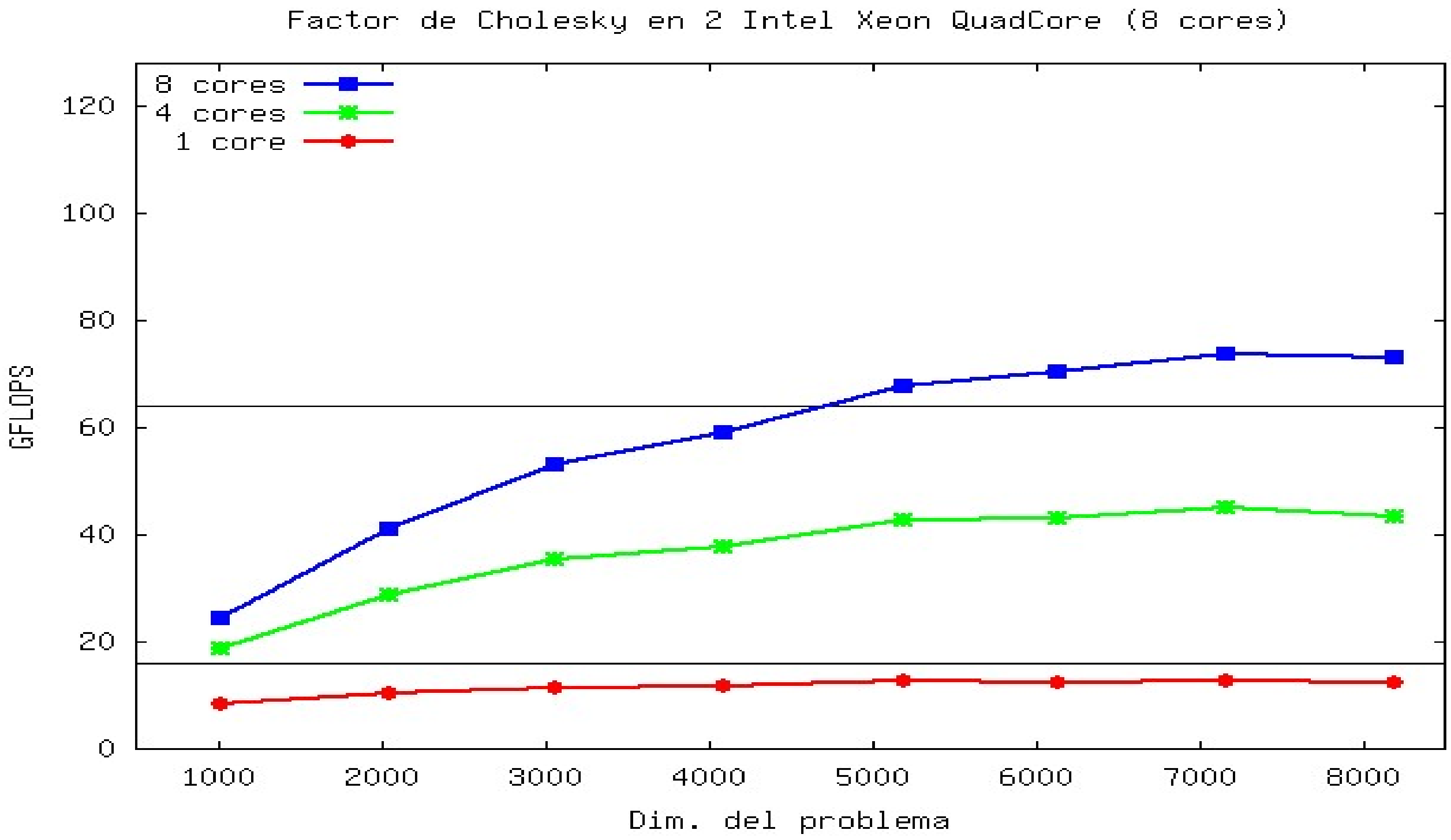
CURRENT LIBRARIES
Performance on a multi-core processor



80% peak

CURRENT LIBRARIES

Performance on a multi-core processor



57% peak

71% peak

80% peak

CURRENT LIBRARIES

Performance on a multi-core processor

- Why?

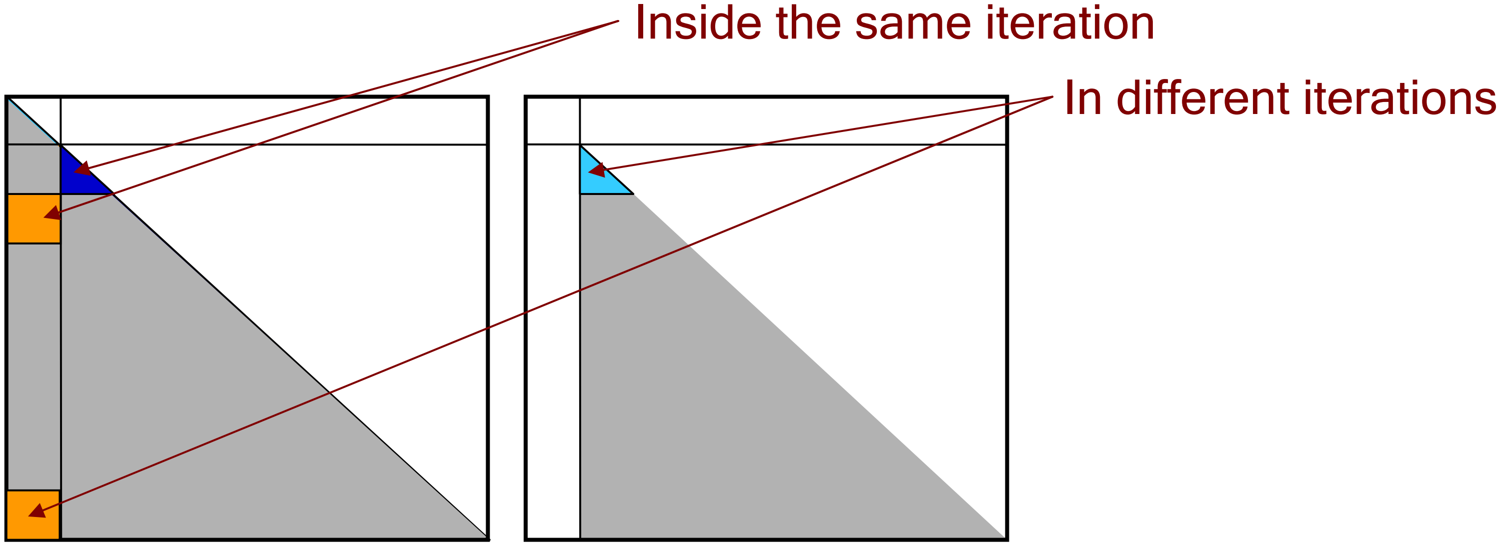
Too many thread synchronizations

```
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]);            // $L_{k+1,k} \leftarrow A_{k+1,k} * L_{kk}^{-T}$
P:  Syrk(A[k+1,k], A[k+1,k+1]);        // $A_{k+1,k+1} \leftarrow A_{k+1,k+1}$
                                       // $- L_{k+1,k} * L_{k+1,k}^T$
  }
}
```

CURRENT LIBRARIES
Performance on a multi-core processor

- Why?

There is more parallelism in the factorization



1st iteration 2nd iteration

DATA-FLOW PARALLELISM

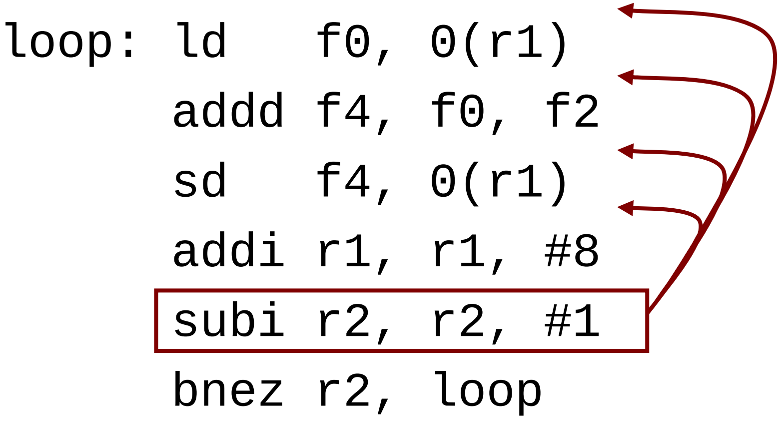
- Out-of-order execution dictated by data dependencies (*data-flow parallelism*)

- Goal: Extract/exploit more parallelism during the execution of linear algebra codes

DATA-FLOW PARALLELISM

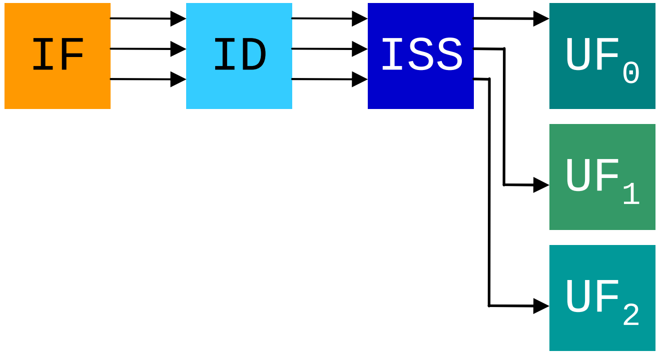
Superscalar processors

Scalar code



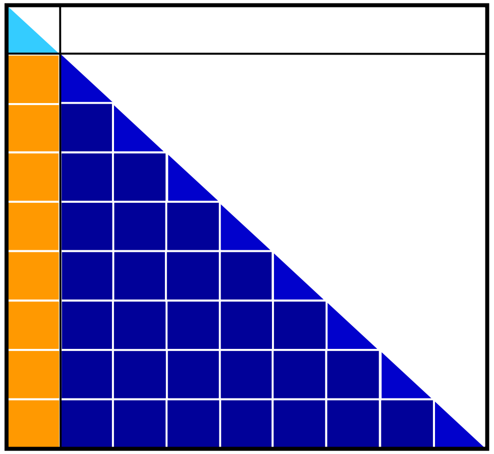
```
loop: ld   f0, 0(r1)
      addd f4, f0, f2
      sd   f4, 0(r1)
      addi r1, r1, #8
      subi r2, r2, #1
      bnez r2, loop
```

Superscalar processor



DATA-FLOW PARALLELISM
Linear algebra

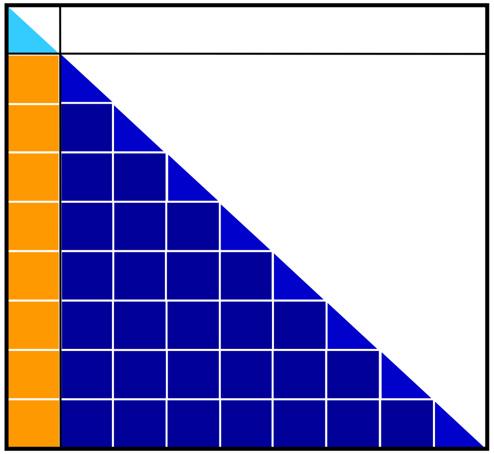
- Possible for linear algebra operations?



```
   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]);
    }
   }
```

DATA-FLOW PARALLELISM
Linear algebra

- Possible for linear algebra operations?



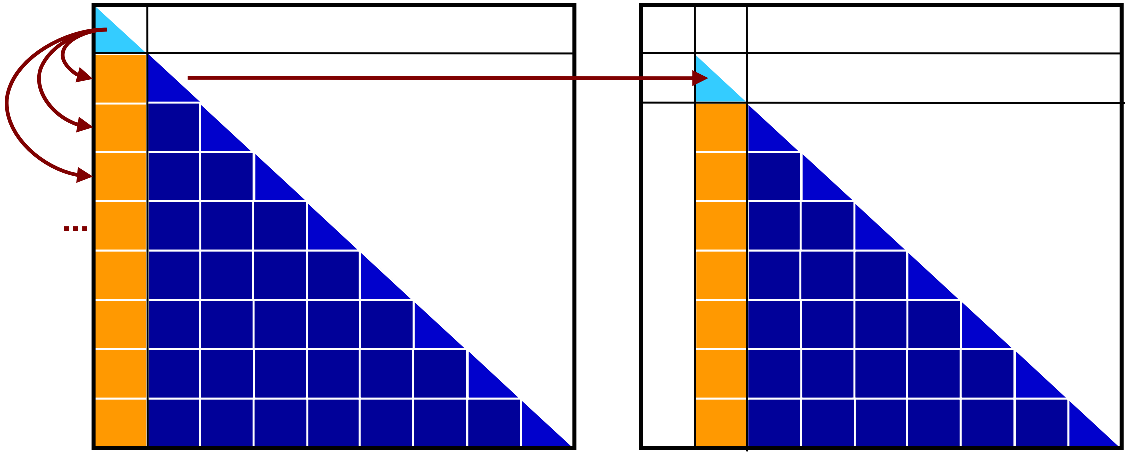
- Techniques applied on “blocks”
- S*oftware* implementation
- Task/thread-level parallelism
- Target: processor cores

DATA-FLOW PARALLELISM
Linear algebra

- Blocks read/written define the data dependencies, as in a scalar code:

```
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]); …
```

- Dependencies between blocks define a task tree:



DATA-FLOW PARALLELISM
Linear algebra

- Blocked code:

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

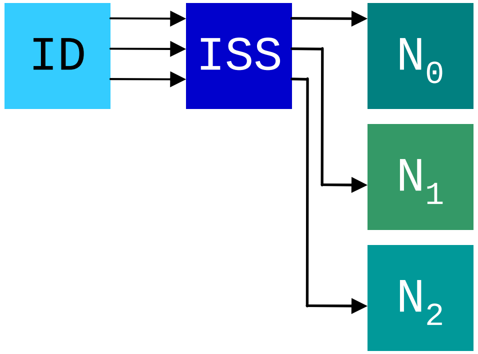
→

Multi-core processor

- How do we generate the task tree?
- How do we execute the tasks in the tree?

PARALELISMO DE FLUJO DE DATOS
Computación matricial

- Use of a *runtime*:
  - Decode (`ID`): Generate the task tree from a symbolic analysis of the code at execution time
  - Issue (`ISS`): Execute the tasks in the tree taking into account the dependencies and the target architecture



PARALELISMO DE FLUJO DE DATOS
Computación matricial

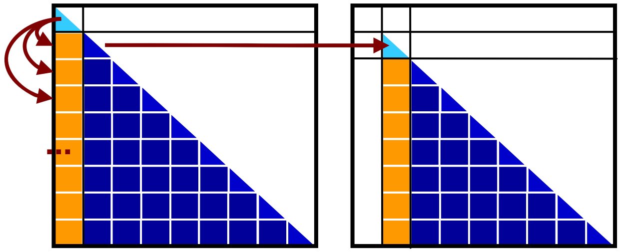
- Decode stage: symbolic analysis of code

Blocked code:

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

→

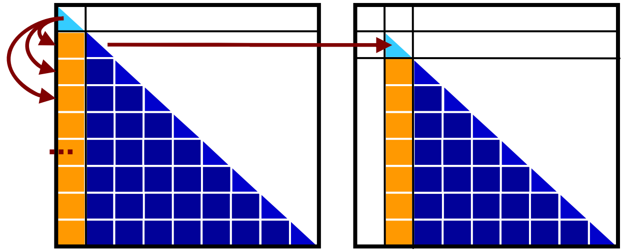
Task tree:



PARALELISMO DE FLUJO DE DATOS
Computación matricial

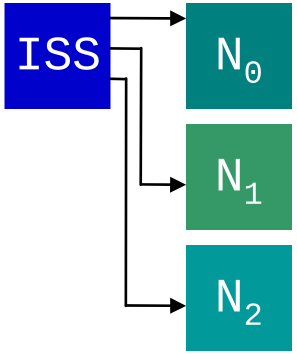
- Issue stage:
  - Scheduling of tasks (when?) depending on the dependencies
  - Mapping of tasks to cores (where?) aware of data locality

Task tree:



→

Multi-core processor:



DATA-FLOW PARALLELISM
Runtime implementations

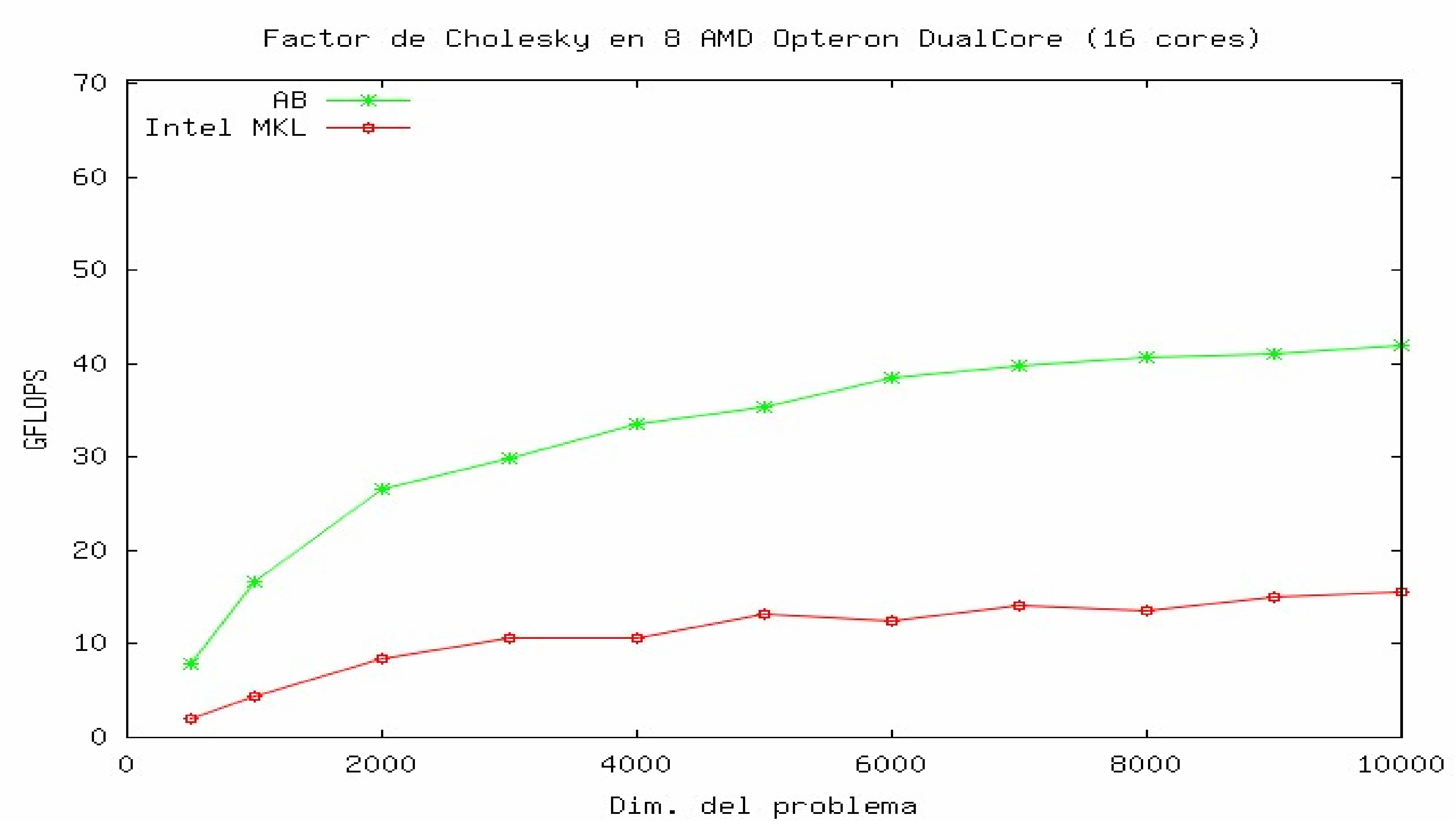
- SuperMatrix (FLAME project between UT@Austin and UJI)
  - Read/write blocks intrinsically defined by the operations
  - Only for linear algebra

- SMPSs (StarSs project from BSC)
  - Read/write blocks explicitely defined by the user

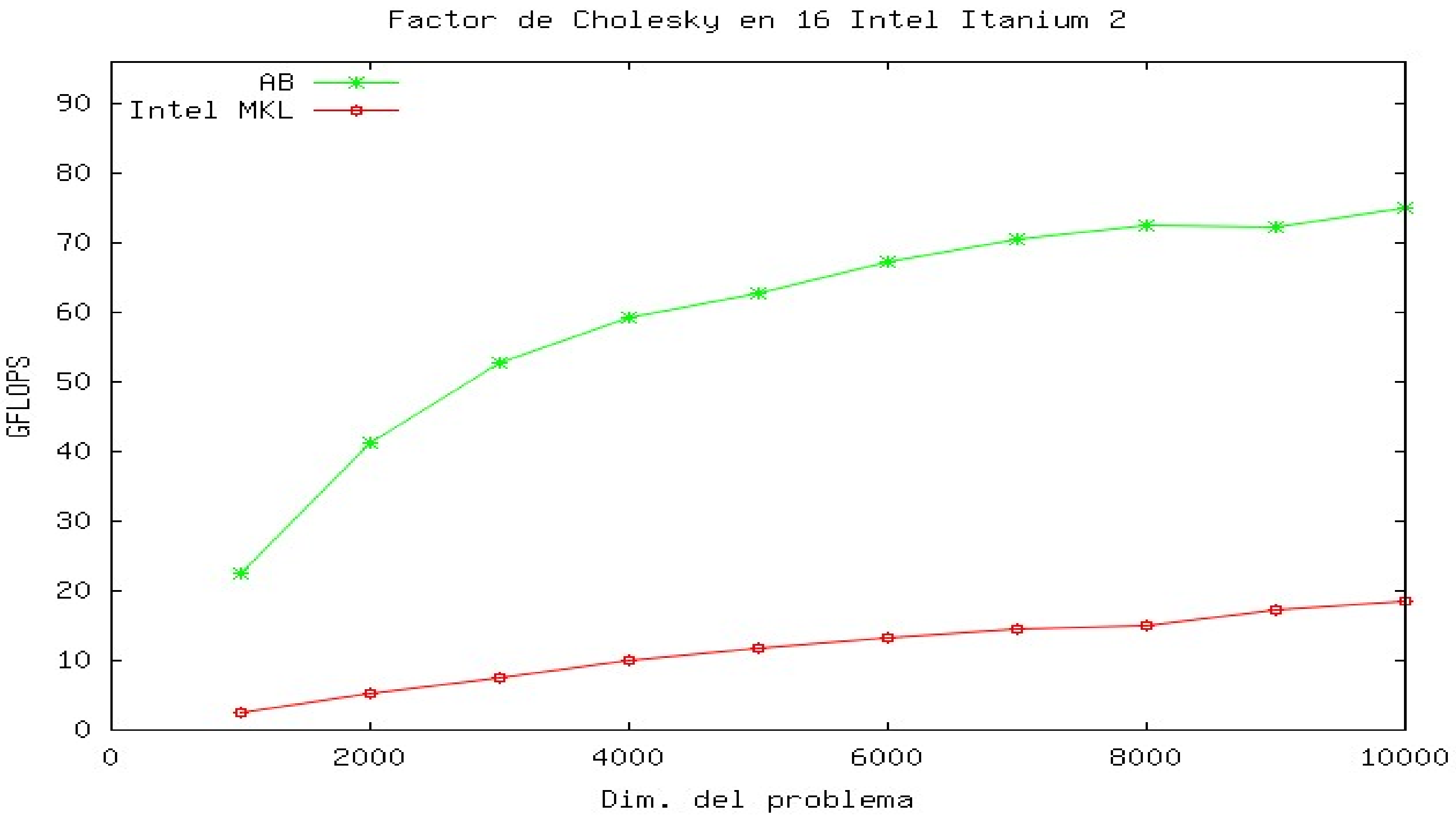
```
#pragma css task inout(A[b*b])
void Chol(double *A);
```

  - Valid for all codes with task-level parallelism

DATA-FLOW PARALLELISM
Performance on a multi-core processor



DATA-FLOW PARALLELISM
Performance on a multi-core processor



SUPERSCALAR TECHNIQUES
Outline

- Introduction
- Superscalar techniques in the construction of linear Algebra libraries for multi-core processors and GPUs:
    1. Parallel execution dictated by data dependencies
    2. Use of *software* caches to hide the existence of multiple address spaces (DSM)
    3. Use of *software* caches to hide latency of disk access

CURRENT ARCHITECTURES
Heterogeneous systems

- CPU-Accelerator(s):
  - Better price/power-performance ratios
    - Slow communication between *host* and devices
    - *Host* and device(s), each with its own address space
    - No *hardware* to maintain coherence

DSM IN HETEROGENEOUS SYSTEMS

- DSM (*distributed-shared memory*) in linear algebra

- Goals:
  - Hide the existence of multiple address spaces (ease programming)
  - Efficience (improve performance)

DSM IN HETEROGENEOUS SYSTEMS

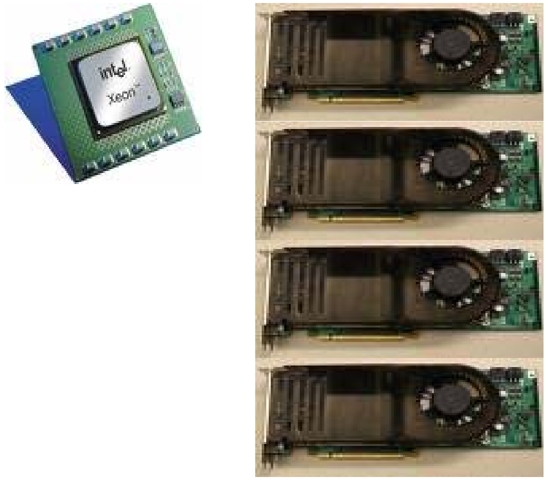
- Mapping tasks to heterogeneous cores

Blocked code:

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

→

Heterogeneous system

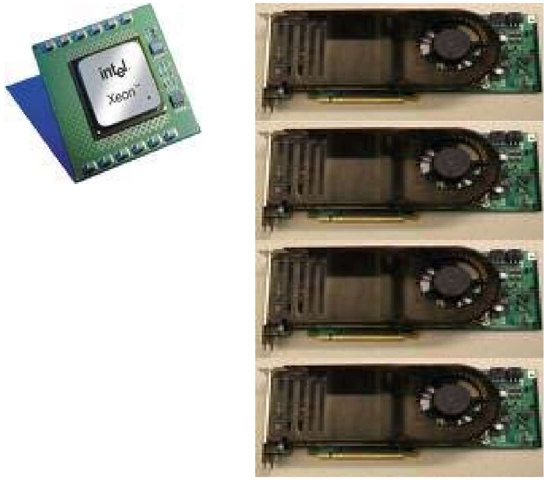


DSM IN HETEROGENEOUS SYSTEMS

- Data transfers
  - Before the computation, transfer the data to the device
  - After computation is completed retrieve back the results →

→ poor data locality

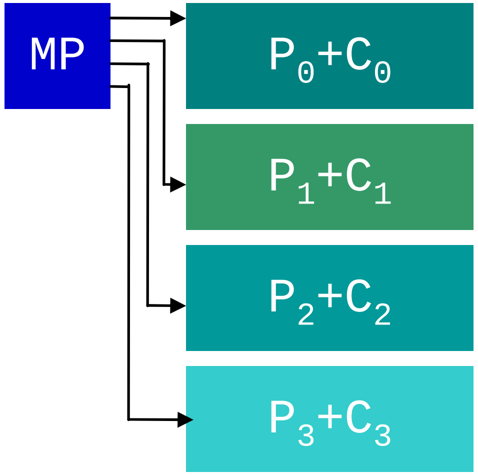
Heterogeneous system



DSM IN HETEROGENEOUS SYSTEMS

- Analogy with current systems

SMP:



MP

P_0+C_0

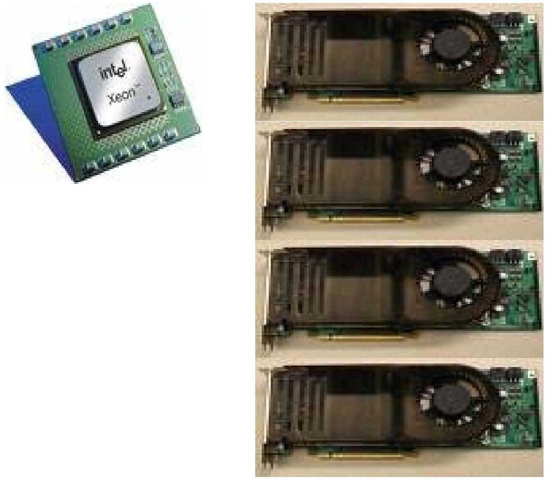
P_1+C_1

P_2+C_2

P_3+C_3

≡

Heterogeneous system

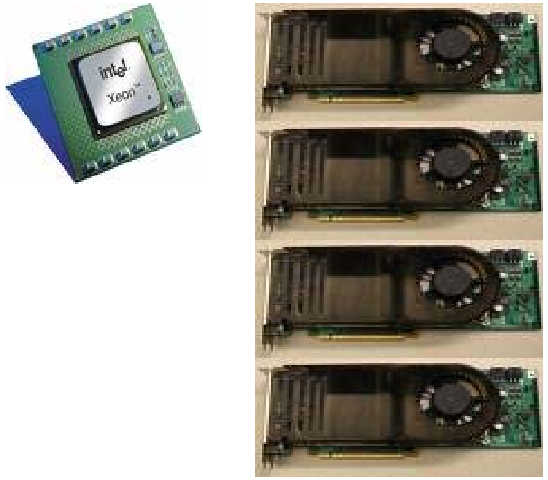


DSM IN HETEROGENEOUS SYSTEMS

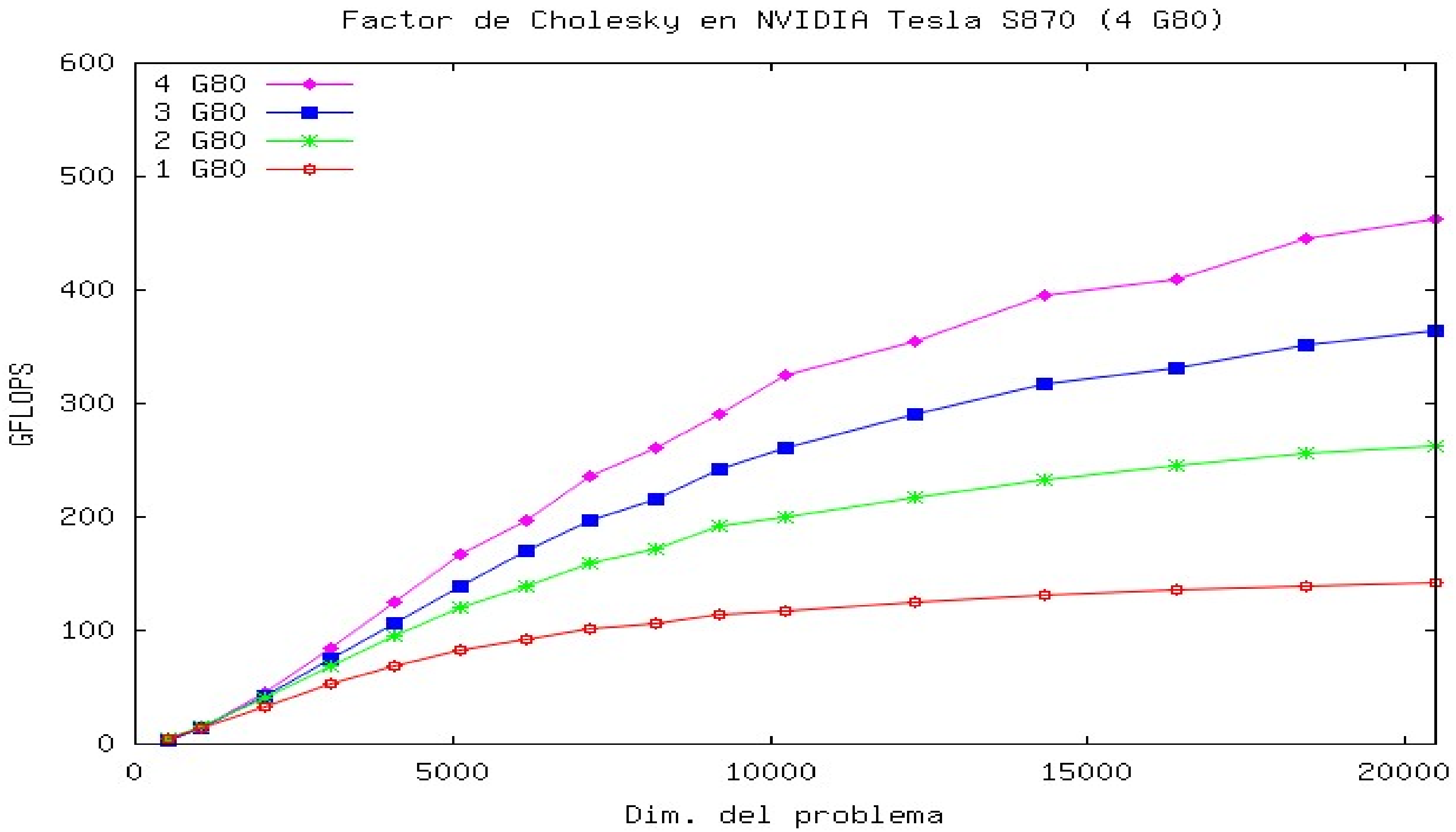
- Reducing data transfers
  - Software cache in the memory of the GPUs:
    - Operate on block to ammortize the cost of cache operation
    - Software → more flexibility
    - *Write-back* to maintain coherence with blocks in host
    - *Write-invalidate* to maintain coherence with blocks in other devices

→

Heterogeneous system



DSM IN HETEROGENEOUS SYSTEMS
Performance in CPU-GPUs systems



SUPERSCALAR TECHNIQUES
Outline

- Introduction
- Superscalar techniques in the construction of linear Algebra libraries for multi-core processors and GPUs:
  1. Parallel execution dictated by data dependencies
  2. Use of *software* caches to hide the existence of multiple address spaces (DSM)
  3. Use of *software* caches to hide latency of disk access

LARGE-SCALE PROBLEMS

- Some dense linear algebra problems are really big (`795k x 795k`)
- Time-to-response is not always critical
- Cost of RAM is not negligible; cost of disk is lower and speed is increasing (solid disks)
- Hardware accelerators are rapidly increasing the GFLOPS rate

INTEGRATE DISK WITH CACHE/RAM

- Use of disk in linear algebra problems
  - O.S. does not always handle virtual memory efficiently

- Goals:
  - Hide use of disk and asynchronous I/O (ease programming)
  - Efficience (improve performance)

INTEGRATE DISK WITH CACHE/RAM

- Handling I/O

Blocked code:

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

→

INTEGRATE DISK WITH CACHE/RAM

- Data transfer
  - Before computation, transfer data from disk to RAM
  - After computation is completed, write back results on disk →

→ poor data locality

INTEGRATE DISK WITH CACHE/RAM

- Reduce data transfers
  - *Software* cache in RAM:
    - Operate on block to ammortize the cost of cache operation
    - Software → more flexibility
    - Symbolic analysis of code to identify the list of tasks
    - Perfect *prefetch*

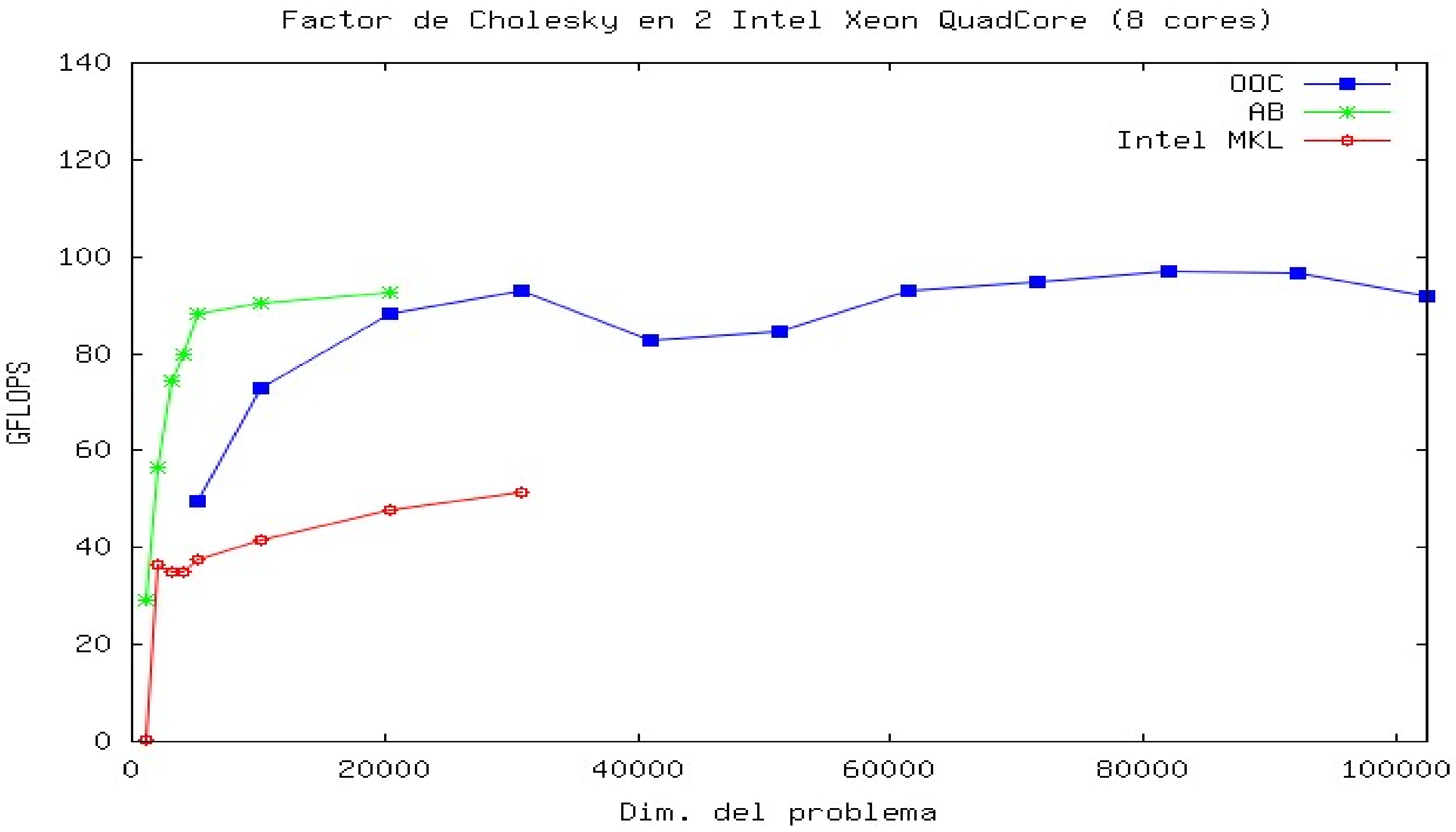
→

INTEGRATE DISK WITH CACHE/RAM

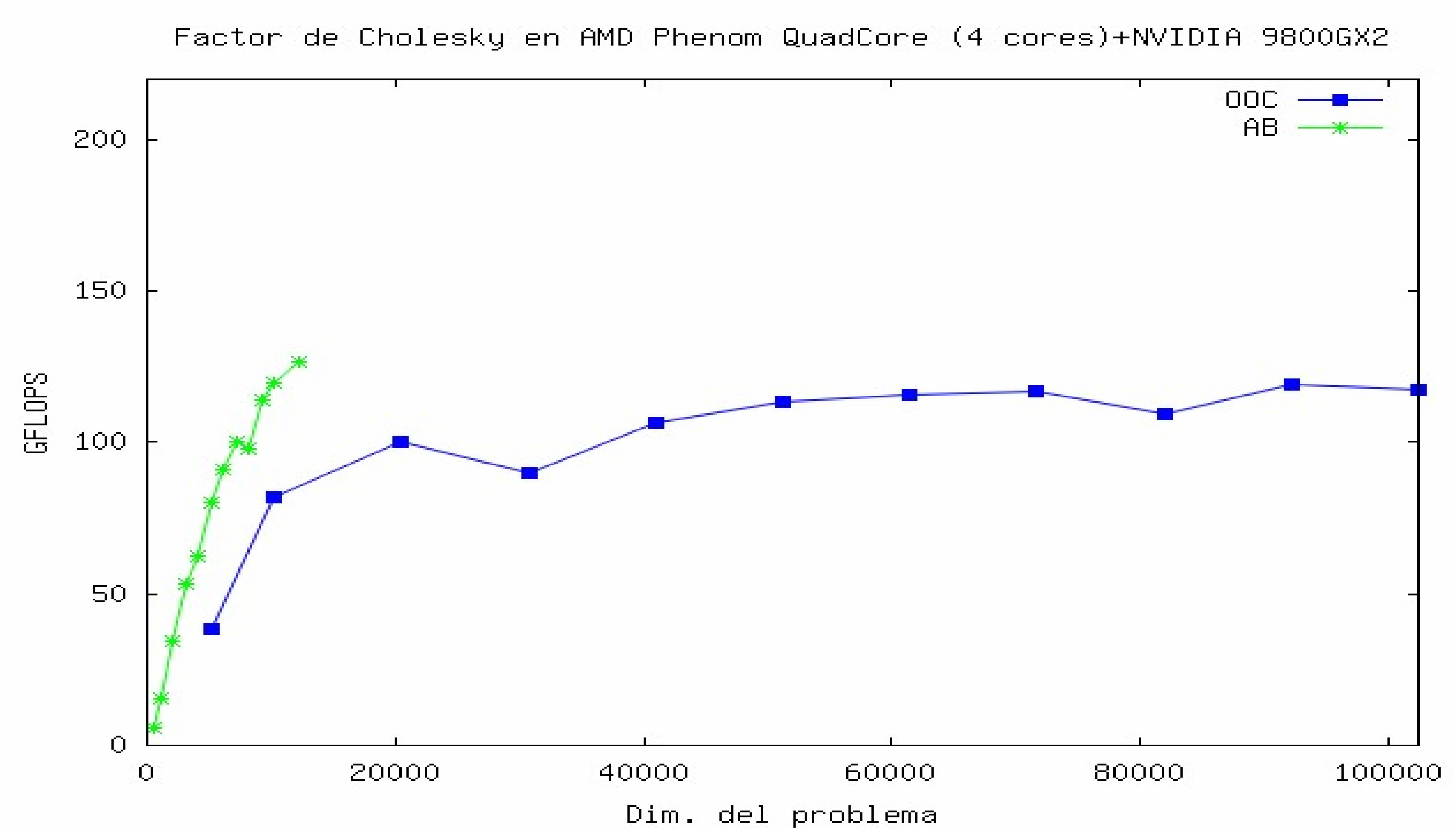
- Asynchronous I/O
  - A thread is in charge of data transfers between RAM (cache) and disk
  - LRU replacement policy →
  - All remaining threads in charge of parallel execution with data in RAM
  - Matrices stored by blocks in disk

INTEGRATE DISK WITH CACHE/RAM

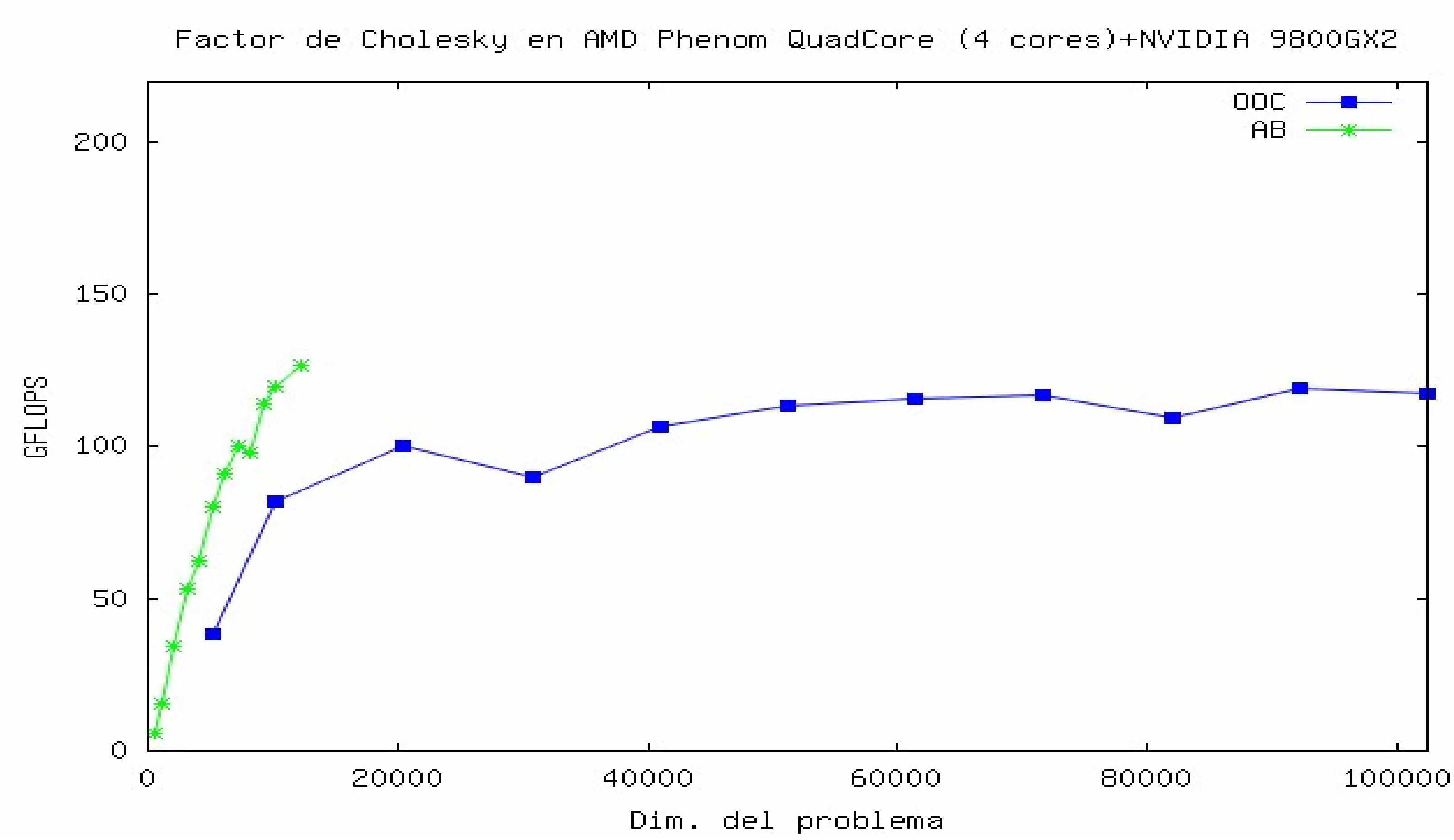
Performance in multi-core processors



INTEGRATE DISK WITH CACHE/RAM
Performance in CPU-GPU systems



INTEGRATE DISK WITH CACHE/RAM
Performance in CPU-GPU systems



FUTURE

- More cores, but watch for processor I/O
- Faster CPU-GPU interconnect (PCI is too slow)
- Direct communication between GPUs
- Hardware support for maintaining coherence
- OOC: solid disks

THANKS

- Joint work:

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

- Colaboration with BSC (use of SMPSs and development of GPUSs)

THANKS

- Support:

UJI

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN

NVIDIA

ClearSpeed™

UT@Austin

NSF

Microsoft®

Thanks for your attention and for your questions*!

* For difficult questions: quintana@icc.uji.es ;-)