Unleashing CPU-GPU Acceleration for Control Theory Applications

Enrique S. Quintana-Ortí

quintana@icc.uji.es

10th International Workshop HeteroPar'2012:

Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms

August 27, 2012, Rhodes Island, Greece

Motivation
Model order reduction

or

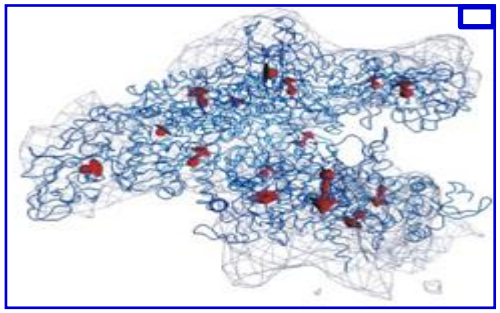
10th International Workshop HeteroPar'2012:

Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms

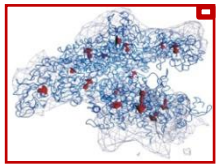
August 27, 2012, Rhodes Island, Greece

Motivation
Model order reduction

- Replace a model of a physical process by a simpler one



$$G(s) = C(sI_n\text{-}A)^{-1}B \quad \rightarrow \quad G_r(s) = C_r(sI_r\text{-}A_r)^{-1}B_r$$



- Why?
  - Control design:
    - Real-time only possible with controllers of low complexity
    - Simple controllers are more robust
  - Simulation
    - Repeated simulation for different force terms (inputs)
    - Reduce once, simulate many!

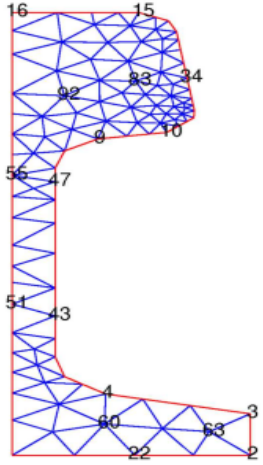
Motivation
Model order reduction

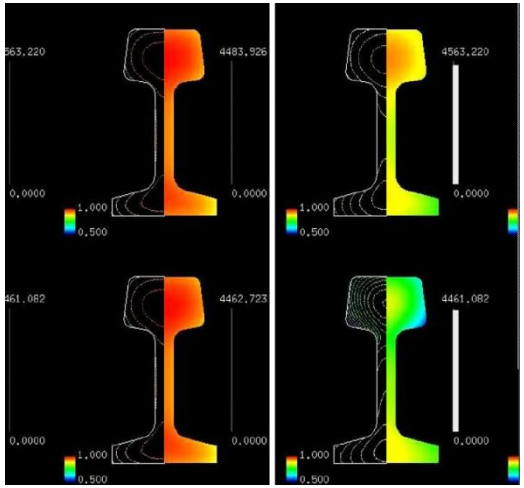
- Optimal cooling of steel profiles:
  - Productivity: reduce the temperature rapidly
  - Quality standards on durability: avoid large gradients in the temperature distributions

- Solve Riccati matrix equation

$$X := F(A,S,Q) = 0,$$

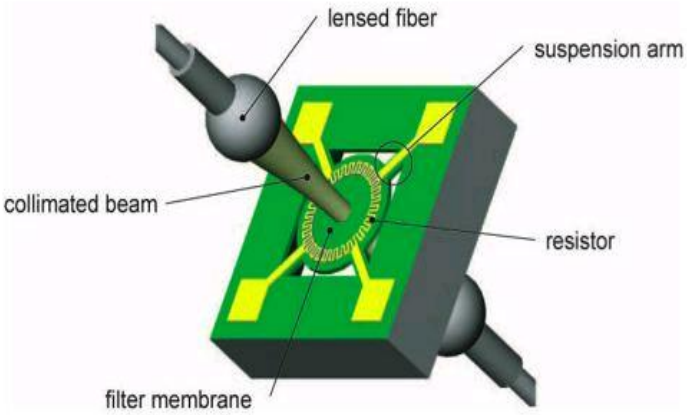
$A \rightarrow n \, x \, n$ with $n = 5{,}177$ - $79{,}841$, depending on mesh width





Motivation
Model order reduction

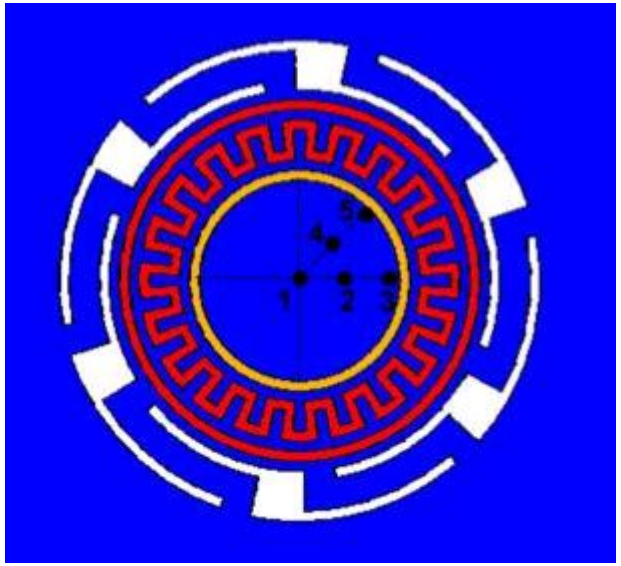
- Tunable optical filter:
  - Optical filter tunable by thermal means
  - Determine electrical power to be applied in order to reach the critical temperature and homogeneous temperature distribution



- Solve Riccati matrix equation

$$X := F(A,S,Q) = 0,$$

$A \rightarrow n \times n$ with $n = 108{,}373$



Motivation
Model order reduction

- Newton's method for the Riccati equation

$$X := F(A,S,Q) = A^T X + X A - X S X + Q = 0,$$

  → one Lyapunov equation per iteration

- Matrix sign function for the Lyapunov equation

$$X := F(A,Z) = A^T X + X A + Z = 0,$$

  → one matrix inversion per iteration

Motivation
Model order reduction

- Invert $A \rightarrow n \, x \, n$

  $\approx 2n^3$ flops

- Intel Xeon: 4 DP flops/cycle,
  e.g., at f = 2.0 GHz
  $\rightarrow$ 8 billion flops/sec.

n	Time 1 core
100	0. 25 ms
1.000	0.25 s

Motivation
Model order reduction

- Invert $A \rightarrow n \, x \, n$

  $\approx 2n^3$ flops

- Intel Xeon: 4 DP flops/cycle,
  e.g., at f = 2.0 GHz
  $\rightarrow$ 8 billion flops/sec.

Steel profiles →

n	Time 1 core	Time 8 cores
100	0. 25 ms	--
1.000	0.25 s	--
10^4	> 4 m	31.2 s

Motivation
Model order reduction

- Invert $A \longrightarrow n \, x \, n$

  $\approx 2n^3$ flops

- Intel Xeon: 4 DP flops/cycle,
  e.g., at f = 2.0 GHz
  $\longrightarrow$ 8 billion flops/sec.

n	Time 1 core	Time 8 cores	Time 16-node cluster, 192 cores
100	0.25 ms	--	--
1.000	0.25 s	--	--
10^4	> 4 m	31.2 s	--
10^5	> 69 h	> 8 h	> 21 m

Steel profiles →

Optical filter →

x #sign function iterations
x #Newton iterations

Numerical methods for model order reduction are costly from the computational point of view!

Motivation
Model order reduction

- 32-node Intel Pentium II cluster (300MHz. DGEMM: 180 DP MFLOPS/core) and myrinet interconnect:
  - Lyapunov eqn. $n = 5{,}177$ (BT method) in 38.5 m.

> *"State-space truncation methods for parallel model reduction of large-scale systems".*
> *P. Benner, E. S. Quintana, G. Quintana.*
> *Parallel Computing, 2003*

Motivation
Model order reduction

- 32-node Intel Pentium II cluster (300MHz. DGEMM: 180 DP MFLOPS/core) and myrinet interconnect:
  - Lyapunov eqn. $n = 5{,}177$ (BT method) in 38.5 m.

"State-space truncation methods for parallel model reduction of large-scale systems".
P. Benner, E. S. Quintana, G. Quintana.
Parallel Computing, 2003

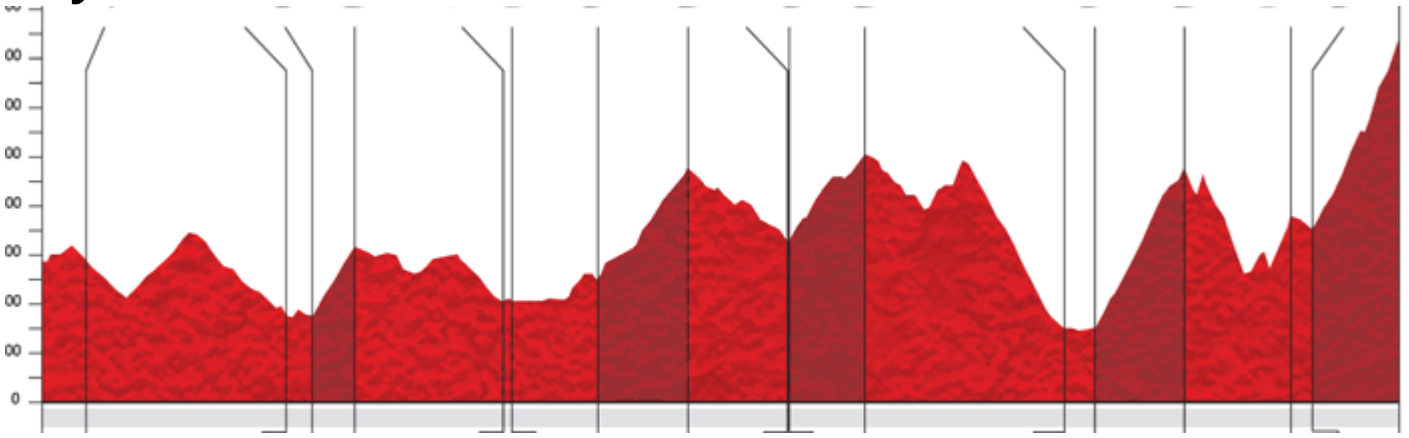
/40

- 8 cores Intel Xeon (2.3 GHz. Peak: 9.2 DP GFLOPS/core) and Tesla C1060 (240 cores):
  - Lyapunov eqn. $n = 5{,}177$ (BT method) in 55.5 s.

"A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms".
P. Benner, P. Ezzatti, D. Kressner, E. S. Quintana, A. Remón.
Parallel Computing, 2011

Index

- Motivation
- Model reduction and solvers for matrix equations
- Matrix inversion on CPU-GPU
- Mixed precision and iterative refinement
- OOC
- Energy efficiency



Only 6 slides!

Model reduction and matrix equations

- Modeling physical processes:

$$\dot{x}(t) = A\,x(t) + B\,u(t) \qquad G(s) = C(sI_n\text{-}A)^{-1}B$$

$$y(t) = C\,x(t)$$

order n (size of A), m inputs and p outputs; $x(0) = x_0$

- Find an alternative *reduced order model:*

$$\dot{x}_r(t) = A_r\,x_r(t) + B_r\,u(t) \qquad G_r(s) = C_r(sI_r\text{-}A_r)^{-1}B_r$$

$$y_r(t) = C_r\,x_r(t)$$

order $r << n$, with $\|y - y_r\|$ small

Model reduction and matrix equations

- SVD-based approximation methods for model reduction
  - Preserve stability
  - Provide a global error bound of the approximation
  - Costly but highly parallel!

- Balanced Truncation: Absolute error method, aiming at:

$$min \; || \, G - G_r \, ||_\infty$$

as

$$|| \, y - y_r \, ||_2 \leq || \, u \, ||_2 \;\; ||G - G_r \, ||_\infty$$

Model reduction and matrix equations

- Balanced Truncation: By far, the most expensive stage!
  - Solve the coupled Lyapunov equations:

    $A\,W_c + W_c\,A^T + BB^T = 0, \qquad A^T\,W_o + W_o\,A + C^TC = 0,$

    for the Cholesky factors $W_c = S^TS$ and $W_o = R^TR$
  - Compute the Hankel singular values from the SVD

    $$SR^T = U\,\Sigma\,V^T = [U_1\ U_2]\ diag(\Sigma_1, \Sigma_2)\ [V_1\ V_2]^T$$

    with partitionings of the "appropriate" dimension r
  - Then, $(A_r, B_r, C_r) := (T_l\,A\,T_r,\ T_l\,B,\ C_r\,T)$ with

    $$T_l := \Sigma_1^{-1/2} V_1^T\ R, \qquad T_r := S^T\ U_1 \Sigma_1^{-1/2}$$

Model reduction and matrix equations

- Balanced Stochastic Truncation:
  - Relative error method
  - Requires the solution of a Riccati equation and a Lyapunov equation

- Matrix sign function-based solvers for the Lyapunov equation
  - Costly ($\approx 2n^3$ flops per iteration)
  - Numerically reliable
  - Easy to optimize and parallelize: multi-core, cluster, GPUs,…

Model reduction and matrix equations

- Consider the Lyapunov eqn.:

$$A\,W_c + W_c\,A^T + BB^T = 0$$

with $W_c = S^T S$

- Newton iteration for the matrix sign function:

By far, the most expensive part!

$$A_{k+1} := (A_k + A_k^{-1})/2, \qquad A_0 := A$$

$$S_{k+1} := [\,S_k,\ A_k^{-1}S_k]/\sqrt{2}, \qquad S_0 := B$$

Upon convergence (after s iterations), $W_c = S^T S \approx S_s S_s^T$

Index

- Motivation
- Model reduction and solvers for matrix equations
- Matrix inversion on CPU-GPU
  - LU vs GJE

    "Matrix inversion on CPU-GPU platforms with applications in control".
    P. Benner, P. Ezzatti, E. S. Quintana, A. Remón.
    Concurrency and Computation: Practice and Experience, 2012

  - Runtime
- Mixed precision and iterative refinement
- OOC
- Energy efficiency

Matrix inversion on CPU-GPU
LU vs GJE. Introduction

- A dense, banded, structured or sparse $\rightarrow$ A^{-1} dense
- $2n^3$ flops for matrix inversion (half if symmetric)
- Permutations needed for numerical stability (except for A symmetric positive definite, s.p.d.). Not shown during the presentation

Matrix inversion on CPU-GPU
LU vs GJE. Introduction

- Blocked, loop-based procedure alike many other dense linear algebra operations

```
for (k=1; k<=n; k+=b){

   ...

}
```

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$

where A_{11} is $b \times b$

Update some of the blocks

$$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$

- At each iteration, $O(n^2 b)$ flops vs $O(n^2)$ memops

Matrix inversion on CPU-GPU
LU vs GJE. Introduction

- Based on LU factorization:
  1. $A = LU$
  2. $U \rightarrow U^{-1} = \tilde{U}$
  3. $X L = \tilde{U}$ for $X = A^{-1}$
- Level-3 BLAS
- Three-sweep algorithm
- Stages 2 and 3 work on triangular matrices: balancing?

Matrix inversion on CPU-GPU
LU vs GJE. Introduction

- Based on GJE:

$$\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := \mathrm{GJE} \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}$$

$$A_{00} := A_{00} + A_{01}A_{10}$$
$$A_{20} := A_{20} + A_{21}A_{10}$$
$$A_{10} := A_{11}A_{10}$$
$$A_{02} := A_{02} + A_{01}A_{12}$$
$$A_{22} := A_{22} + A_{21}A_{12}$$
$$A_{12} := A_{11}A_{12}$$

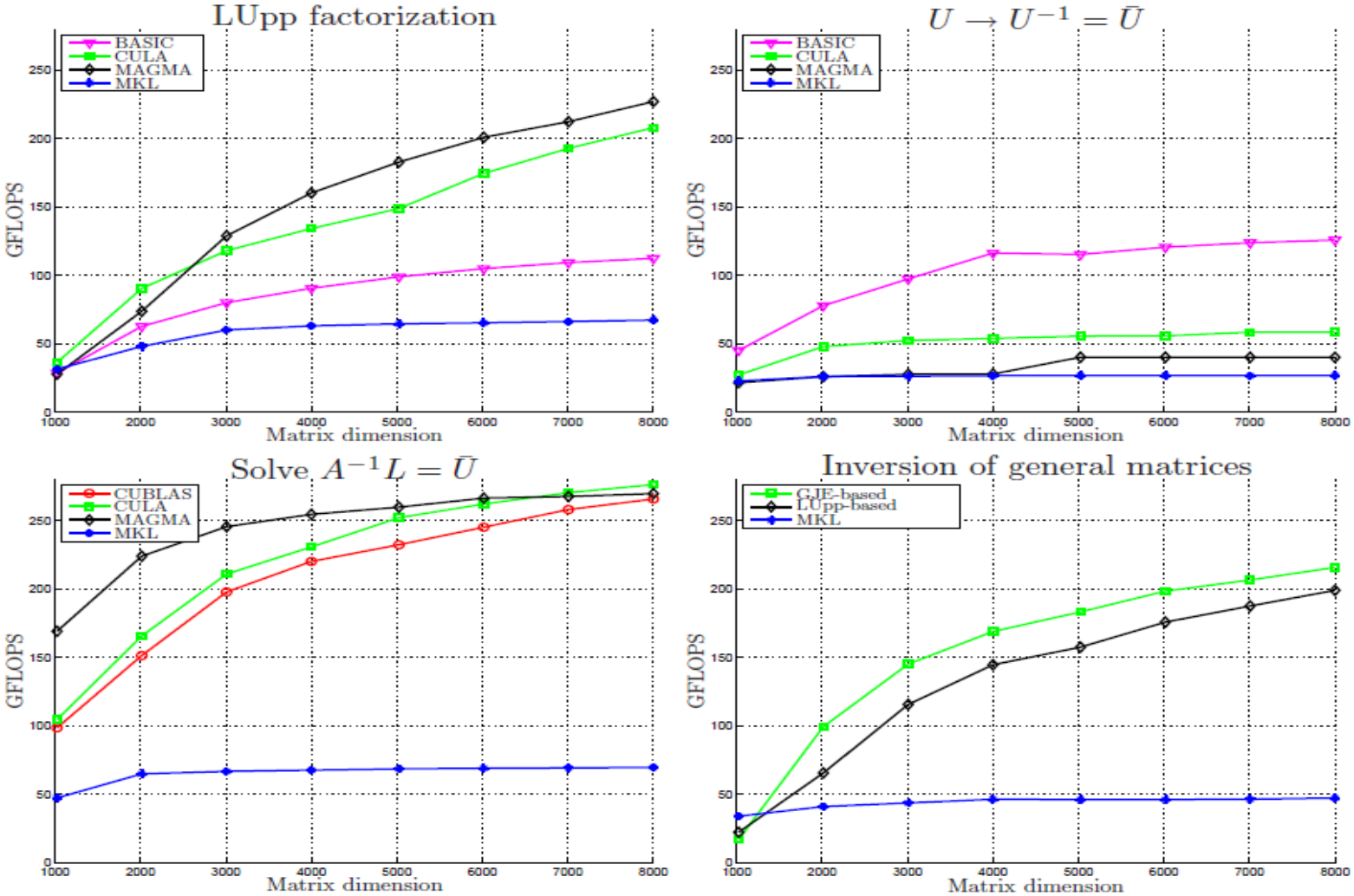
- Level-3 BLAS rich in matrix-matrix products
- One-sweep algorithm
- Same cost and numerical stability as LU-based procedure
- All iterations work on full dense matrix, performing the same number of flops: balancing

Matrix inversion on CPU-GPU
LU vs GJE. Results

- Experimental setup
  - IEEE double precision arithmetic (must!)
  - Performance in GFLOPS
  - Platform
    - Two Intel Xeon E5520 quadcore (8 cores total)
    - NVIDIA Tesla C2050 (Fermi)
  - Software
    - Intel MKL (v11.1) for BLAS and LAPACK
    - NVIDIA CUBLAS (v4.0) for BLAS
    - EM Photonics CULA (R13a) for BLAS and LAPACK
    - UTK MAGMA (v1.1.0) for BLAS and LAPACK
    - BASIC (ad-hoc) implementations of certain kernels

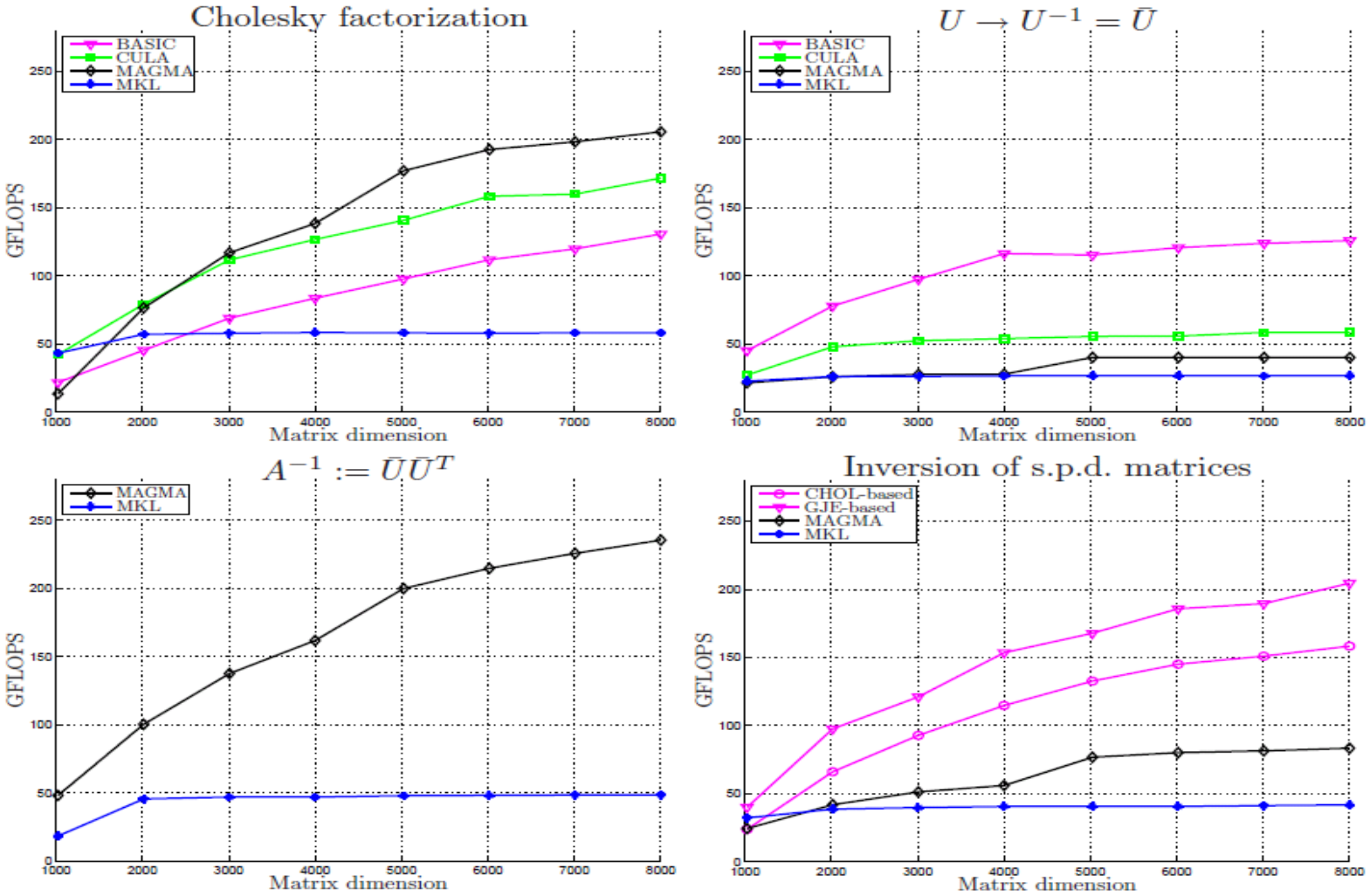
Matrix inversion on CPU-GPU
LU vs GJE. Results

- General matrices



Matrix inversion on CPU-GPU
LU vs GJE. Results

- s.p.d. matrices



Matrix inversion on CPU-GPU
LU vs GJE

- Optimizations
  - Hybrid CPU-GPU acceleration
    - Optimize for CPU (block algorithm, etc.)
    - Optimize for GPU (padding, merge, etc.)
  - Reduce PCI data transfers (slow PCI-e)
  - Overlap communication and computation
  - Avoiding bottlenecks with look-ahead

Techniques applicable to many dense (and sparse) linear algebra kernels!

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Hybrid CPU-GPU acceleration

$$\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := \mathrm{GJE} \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}$$

$$A_{00} := A_{00} + A_{01} A_{10}$$
$$A_{20} := A_{20} + A_{21} A_{10}$$
$$A_{10} := A_{11} A_{10}$$
$$A_{02} := A_{02} + A_{01} A_{12}$$
$$A_{22} := A_{22} + A_{21} A_{12}$$
$$A_{12} := A_{11} A_{12}$$

$$\left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

  - Computations on CPU-only, GPU-only or both?

    Obvious answer for HeteroPAR audience!

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Hybrid CPU-GPU acceleration

$$\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := \mathrm{GJE} \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}$$

Panel factorization

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$

$$A_{00} := A_{00} + A_{01} A_{10}$$
$$A_{20} := A_{20} + A_{21} A_{10}$$
$$A_{10} := A_{11} A_{10}$$
$$A_{02} := A_{02} + A_{01} A_{12}$$
$$A_{22} := A_{22} + A_{21} A_{12}$$
$$A_{12} := A_{11} A_{12}$$

  - Due to complexity and fine granularity of pivoting, clear candidate for CPU!

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Hybrid CPU-GPU acceleration

$$\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := \mathrm{GJE} \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}$$

$$\begin{array}{l} A_{00} := A_{00} + A_{01}A_{10} \\ A_{20} := A_{20} + A_{21}A_{10} \\ A_{10} := A_{11}A_{10} \end{array}$$

$$\begin{array}{l} A_{02} := A_{02} + A_{01}A_{12} \\ A_{22} := A_{22} + A_{21}A_{12} \\ A_{12} := A_{11}A_{12} \end{array}$$

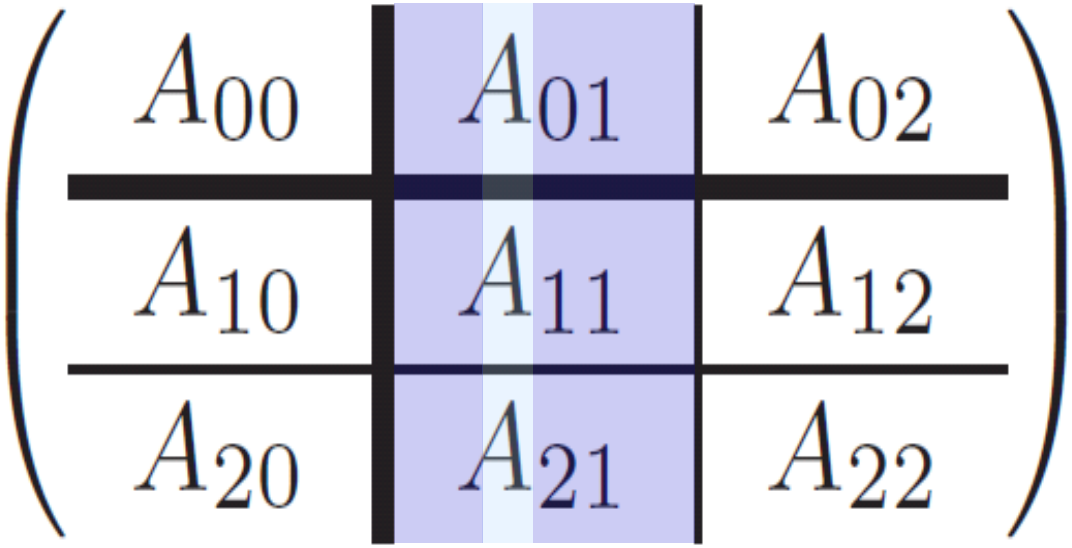
Concurrent update

$$\left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

- 1-D workload distribution!
- In general, 2-D distributions reduce volume of communications but, because factorization is on CPU, there is no clear advantage
- Width of CPU/GPU blocks? Easily tunable
- With look-ahead, factorization and update can be overlapped

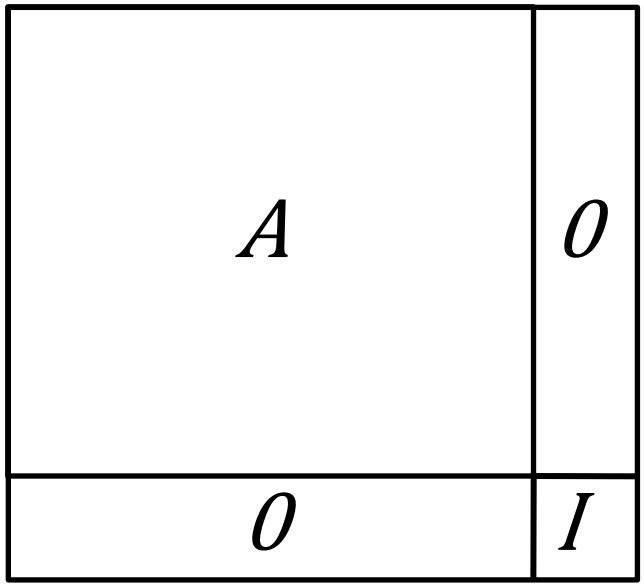
Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Optimization on CPU
  - Blocked panel factorization



Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Optimization on GPU
  - Padding
    - Ensure that GPU operations always work with blocks of size that is an integer multiple of 32



    - Size of I <32. Overhead negligible, provided n is large

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Optimization on GPU
  - Merge of update

$$\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := \text{GJE} \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}$$

$$\begin{aligned} A_{00} &:= A_{00} + A_{01}A_{10} \\ A_{20} &:= A_{20} + A_{21}A_{10} \\ A_{10} &:= A_{11}A_{10} \end{aligned}$$

$$\begin{aligned} A_{02} &:= A_{02} + A_{01}A_{12} \\ A_{22} &:= A_{22} + A_{21}A_{12} \\ A_{12} &:= A_{11}A_{12} \end{aligned}$$

$$\hat{A} := A_{10}, \quad A_{10} := 0$$

$$\begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \end{pmatrix} := \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \end{pmatrix} + \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} \hat{A}$$

Same idea

  - GPU kernels perform much better with larger data

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Reduce PCI data transfers
  - Keep bulk of data close to GPU cores
  - View GPU memory as cache of data in CPU memory
  - Data does not fit into GPU memory?

    → Old ideas from OOC computing (later)

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Overlap communication and computation
  - Initially, matrix A in CPU memory
  - Copy it to GPU memory:
    - n^2 memops (PCI) for $2n^3$ flops: negligible ratio for large n
  - Part of matrix sign function iteration:

$$A_{k+1} := (A_k + A_k^{-1})/2, \qquad A_0 := A$$

    - n^2 memops (PCI) for $2n^3 s$ flops: negligible ratio even for moderate n

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Overlap communication and computation
  - During the iteration, matrix A in GPU memory

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$

$$\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := \mathrm{GJE} \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}$$

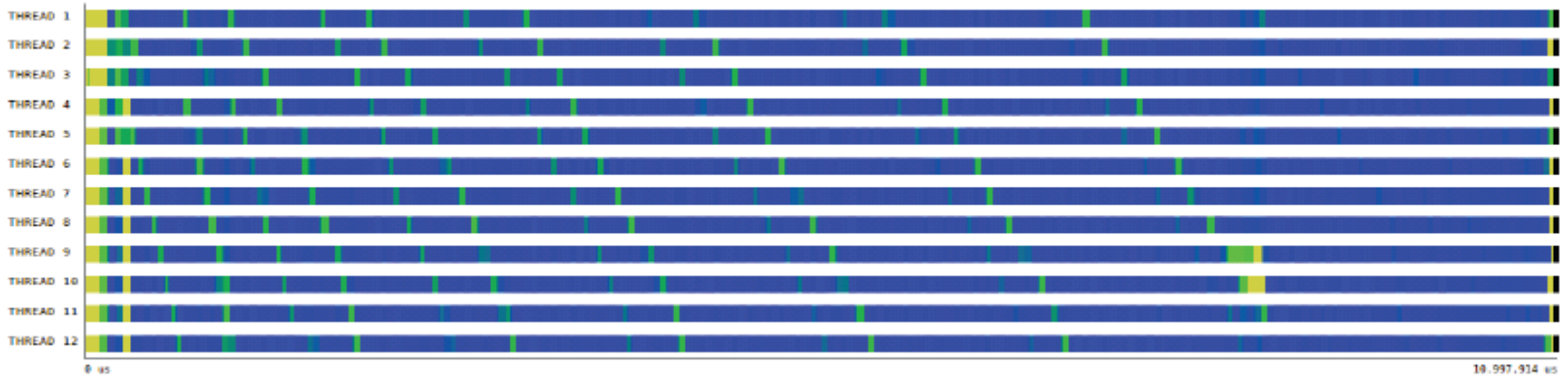
$$\begin{aligned} A_{00} &:= A_{00} + A_{01} A_{10} \\ A_{20} &:= A_{20} + A_{21} A_{10} \\ A_{10} &:= A_{11} A_{10} \end{aligned}$$

$$\begin{aligned} A_{02} &:= A_{02} + A_{01} A_{12} \\ A_{22} &:= A_{22} + A_{21} A_{12} \\ A_{12} &:= A_{11} A_{12} \end{aligned}$$

  - Copy panel to CPU memory:
    - $2nb$ memops (PCI) for $2n^2b$ flops: favorable ratio for large n
  - Apply double buffering? Look-ahead!

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

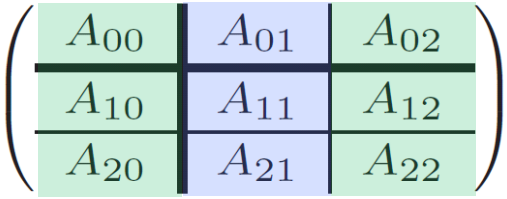
- Look-ahead:
  - Reduce the impact of “slow” computations
  - Works for LU, Cholesky, QR,…
  - Integrated, e.g., into Intel MKL

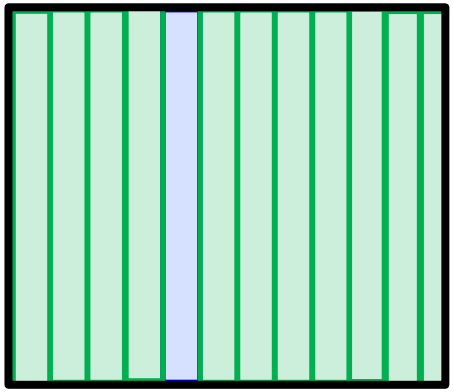


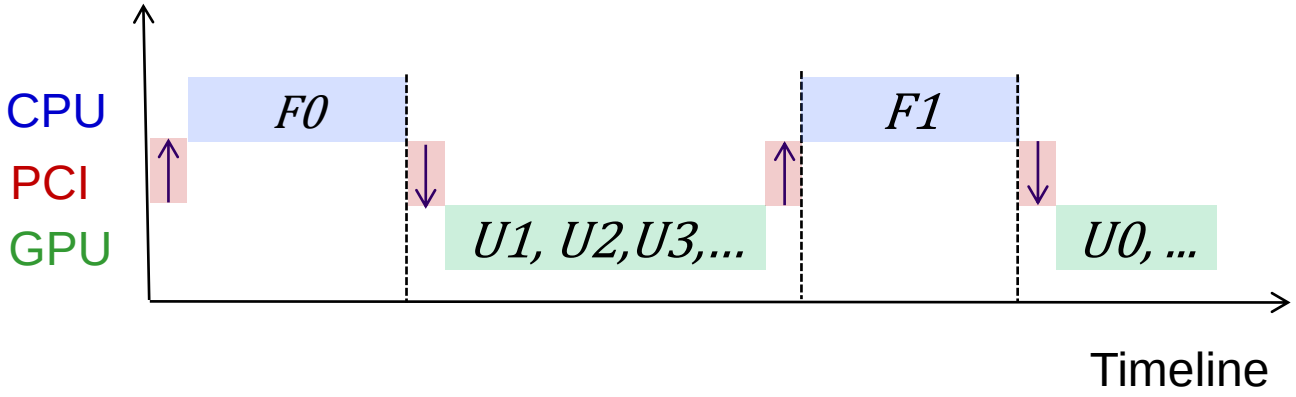
“A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization”.
P. Strazdins.
Techn. Report TR-CS-98-07, Dept. Computer Science, The Australian National University, 1998

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Matrix inversion via GJE without look-ahead





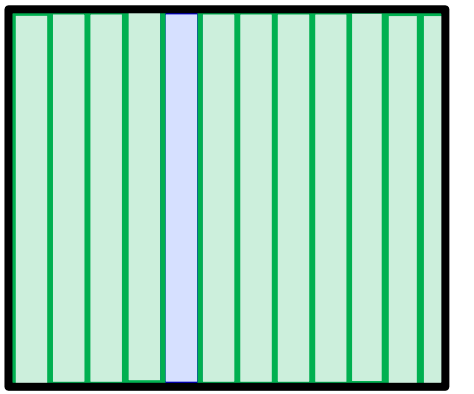


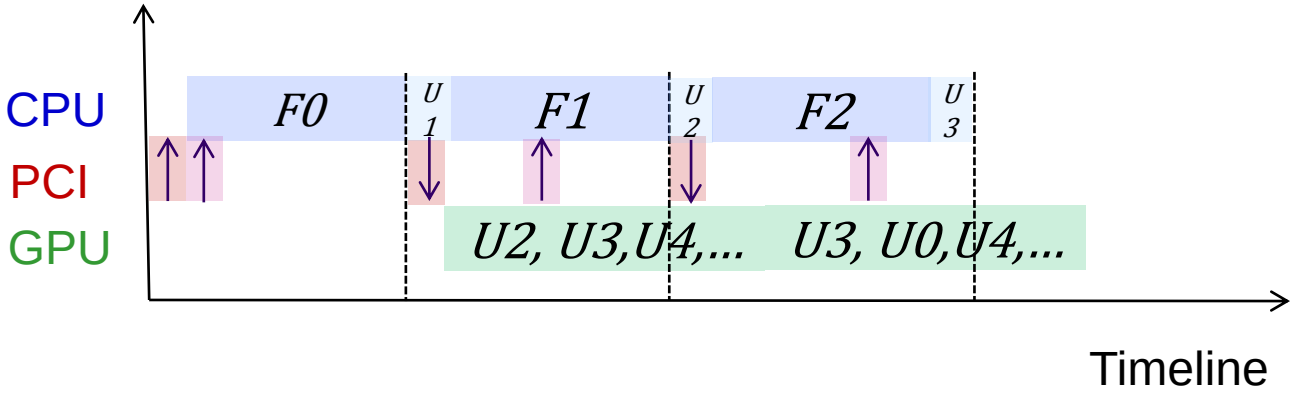
  - Panel factorization is mostly sequential
  - As #cores/GPU power grows, execution time dominated by panel factorization!

Matrix inversion on CPU-GPU LU vs GJE. Optimization

- Matrix inversion via GJE with look-ahead

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$





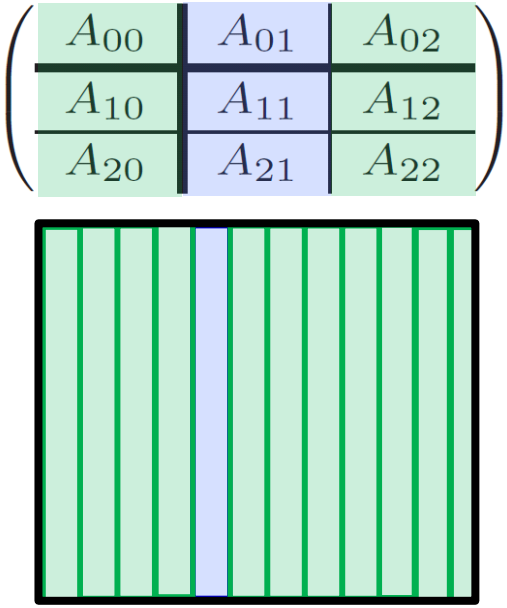
- Originality: apply to matrix inversion
- Hybrid CPU-GPU platform
- Flops and memops remain constant during procedure

Matrix inversion on CPU-GPU
LU vs GJE. Optimization

- Look-ahead of depth 2, 3,…?
  - Complex programming
  - Optimal depth of look-ahead depends on problem size, architecture, etc.

- Dynamic look-ahead?

  $\rightarrow$ Runtime!



Matrix inversion on CPU-GPU Runtime. Overview

- "Taxonomy"

	CPU (multicore)	CPU-GPU
Dense linear algebra	libflame+SuperMatrix - UT PLASMA - UTK	libflame+SuperMatrix - UT MAGMA – UTK
Generic	SMPSs (OMPSs) - BSC	GPUSs (OMPSs) – BSC StarPU - INRIA Bordeaux

Matrix inversion on CPU-GPU Runtime. Overview

- Principles of operation:
  - Exploitation of task parallelism
  - Dynamic detection of data dependencies (data-flow parallelism)
  - Scheduling tasks to resources on-the-fly

  - Surely not a new idea!

    "An Efficient Algorithm for Exploiting Multiple Arithmetic Units".
    R. M. Tomasulo.
    IBM J. of R&D, 1967

Matrix inversion on CPU-GPU Runtime. Overview

- OMPSs runtime

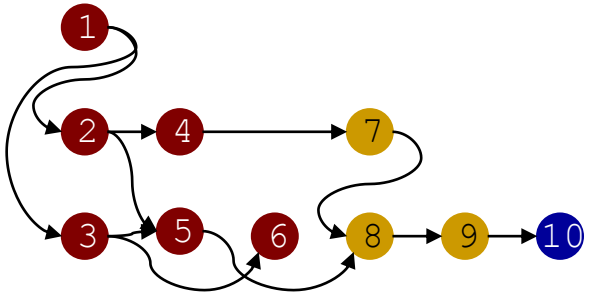
Annotated code with OpenMP-like pragmas

```
// Task
#pragma omp task input  (oper1,…)  \
                 output (oper2,…) \
                 inout  (oper3,…)

void task_function( oper1,
                    oper2,
                    oper3,... )
{
   ...
}
```

- Automatic identification of tasks/dependencies

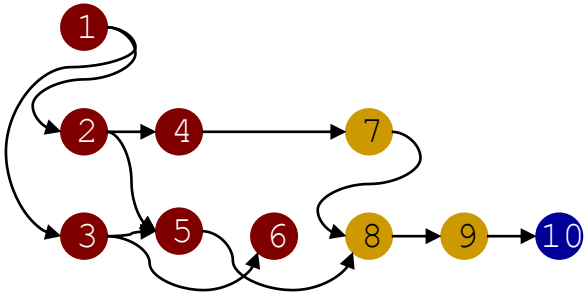
How?

Strict order of invocations to operations (tasks) and directionality of operands (input, output, inout) identify dependencies

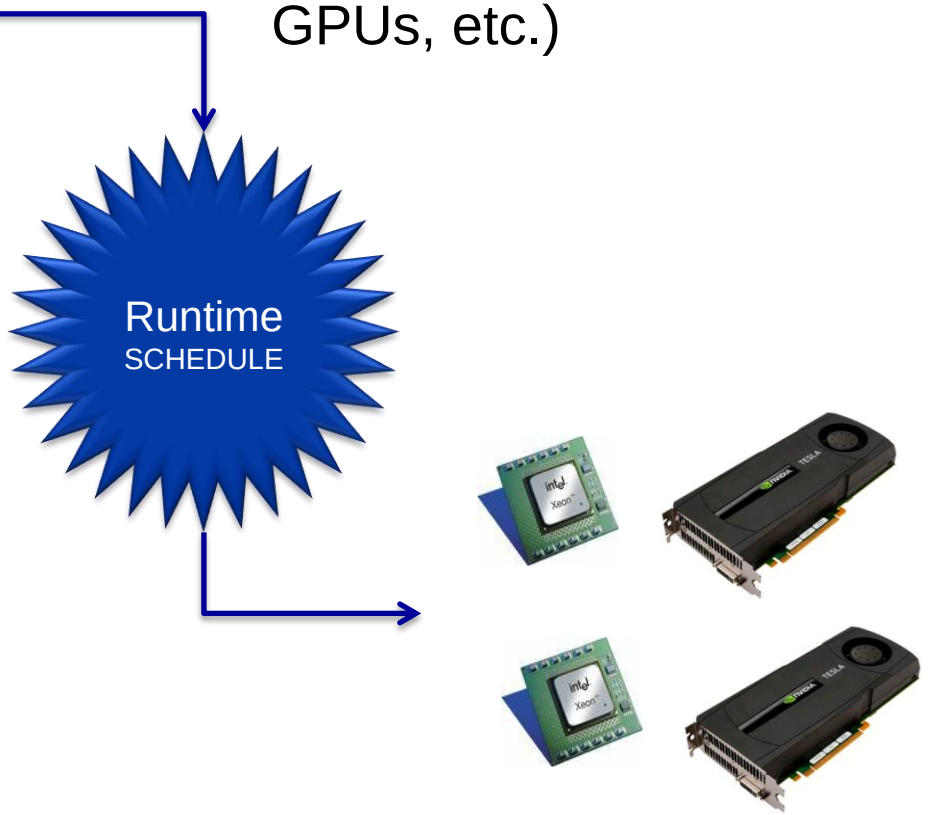


Matrix inversion on CPU-GPU Runtime. Overview

- OMPSs runtime



- Scheduling of tasks to computational resources (cores, GPUs, etc.)

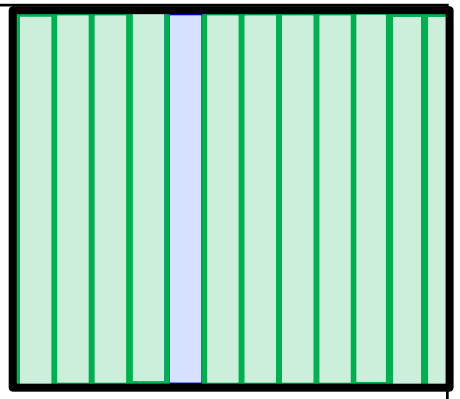


Matrix inversion on CPU-GPU Runtime. Overview

- OMPSs
  - Automatic handling of Multi-GPU execution
    - Transparent data-management on GPU side (allocation, transfers,...) and synchronization
    - One manager thread in the host per GPU. Responsible for:
      - Transferring data from/to GPUs
      - Executing GPU tasks
      - Synchronization
    - Overlap of computation and communication
    - Data pre-fetch

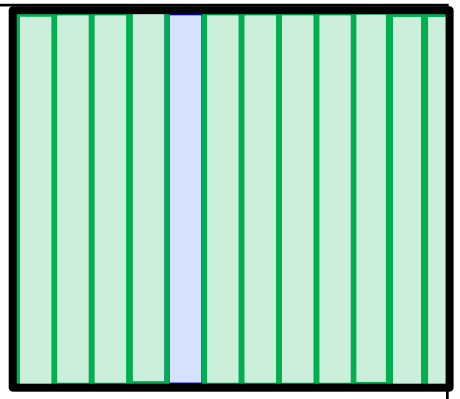
Matrix inversion on CPU-GPU Runtime. OMPSs GJE

```
void main (...) {...

   for (k=1; k<=n; k+=b){
     Factorize( n, b, &Aref( 1, k ) );
     for (j=1; j<k; j+=b)
        Update( n, b, &Aref( 1, k ), &Aref( 1, j ) );
     for (j=k+b; j<=n; j+=b)
        Update( n, b, &Aref( 1, k ), &Aref( 1, j ) );
   }

}


void Factorize( n, b, A ){ … }


void Update( n, b, A, B ){ … }
```



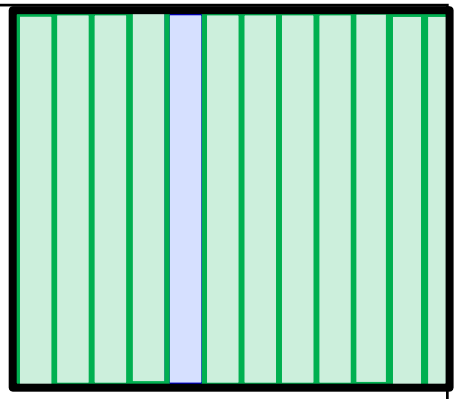
Matrix inversion on CPU-GPU Runtime. OMPSs GJE

```
void main (...) {...
   #pragma omp start
   for (k=1; k<=n; k+=b){
     Factorize( n, b, &Aref( 1, k ) );
     for (j=1; j<k; j+=b)
        Update( n, b, &Aref( 1, k ), &Aref( 1, j ) );
     for (j=k+b; j<=n; j+=b)
        Update( n, b, &Aref( 1, k ), &Aref( 1, j ) );
   }
   #pragma omp stop
}

#pragma omp task input( n, b ) inout([n][b]A )
void Factorize( n, b, A ){ … }

#pragma omp task input( n, b, [n][b]A ) inout([n][b]B )
void Update( n, b, A, B ){ … }
```



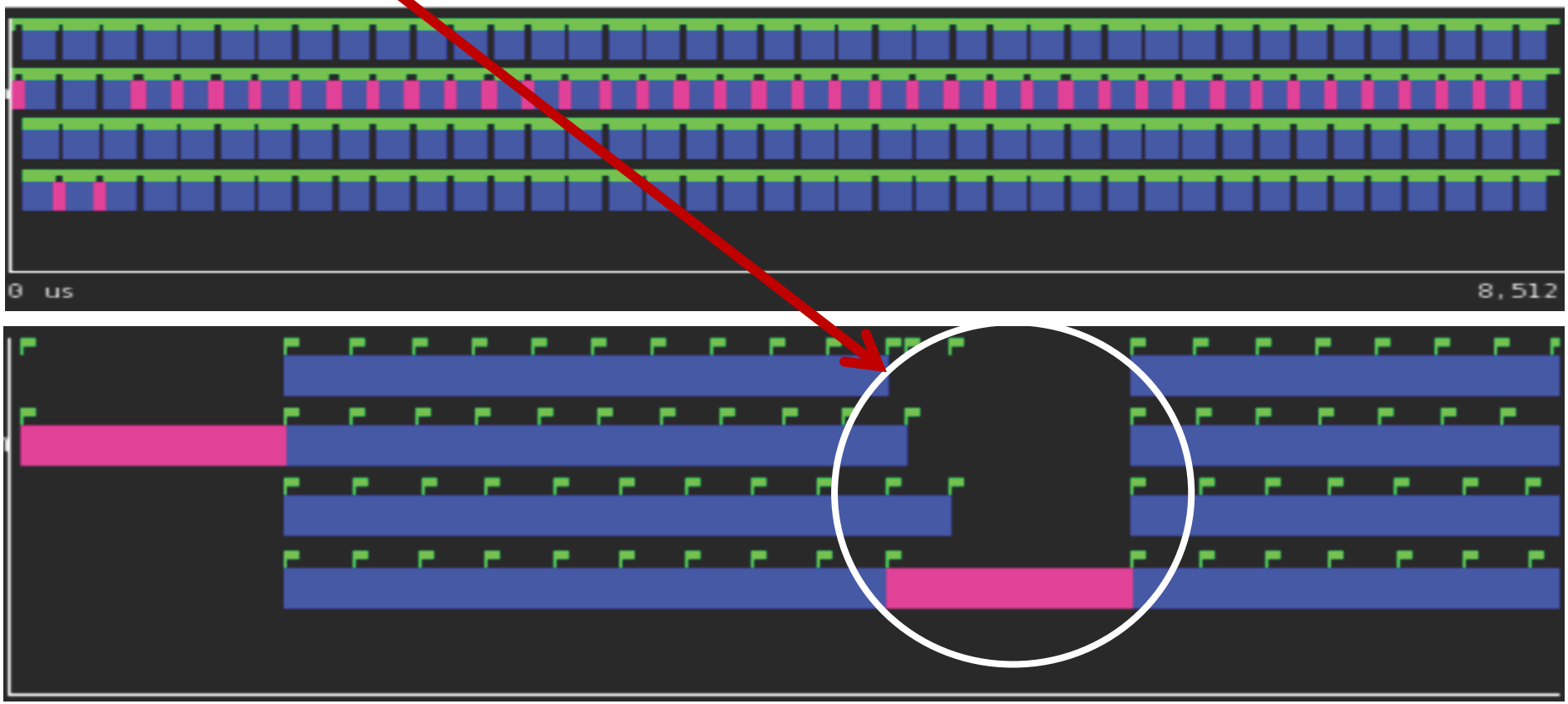
Matrix inversion on CPU-GPU Runtime. OMPSs GJE

```
void main (...) {...
   #pragma omp start
   for (k=1; k<=n; k+=b){
     Factorize( n, b, &Aref( 1, k ) );
     for (j=1; j<k; j+=b)
        Update( n, b, &Aref( 1, k ), &Aref( 1, j ) );
     for (j=k+b; j<=n; j+=b)
        Update( n, b, &Aref( 1, k ), &Aref( 1, j ) );
   }
   #pragma omp stop
}
#pragma omp target ( smp )
#pragma omp task input( n, b ) inout([n][b]A )
void Factorize( n, b, A ){ … }
#pragma omp target ( cuda ) copy_deps
#pragma omp task input( n, b, [n][b]A ) inout([n][b]B )
void Update( n, b, A, B ){ … }
```



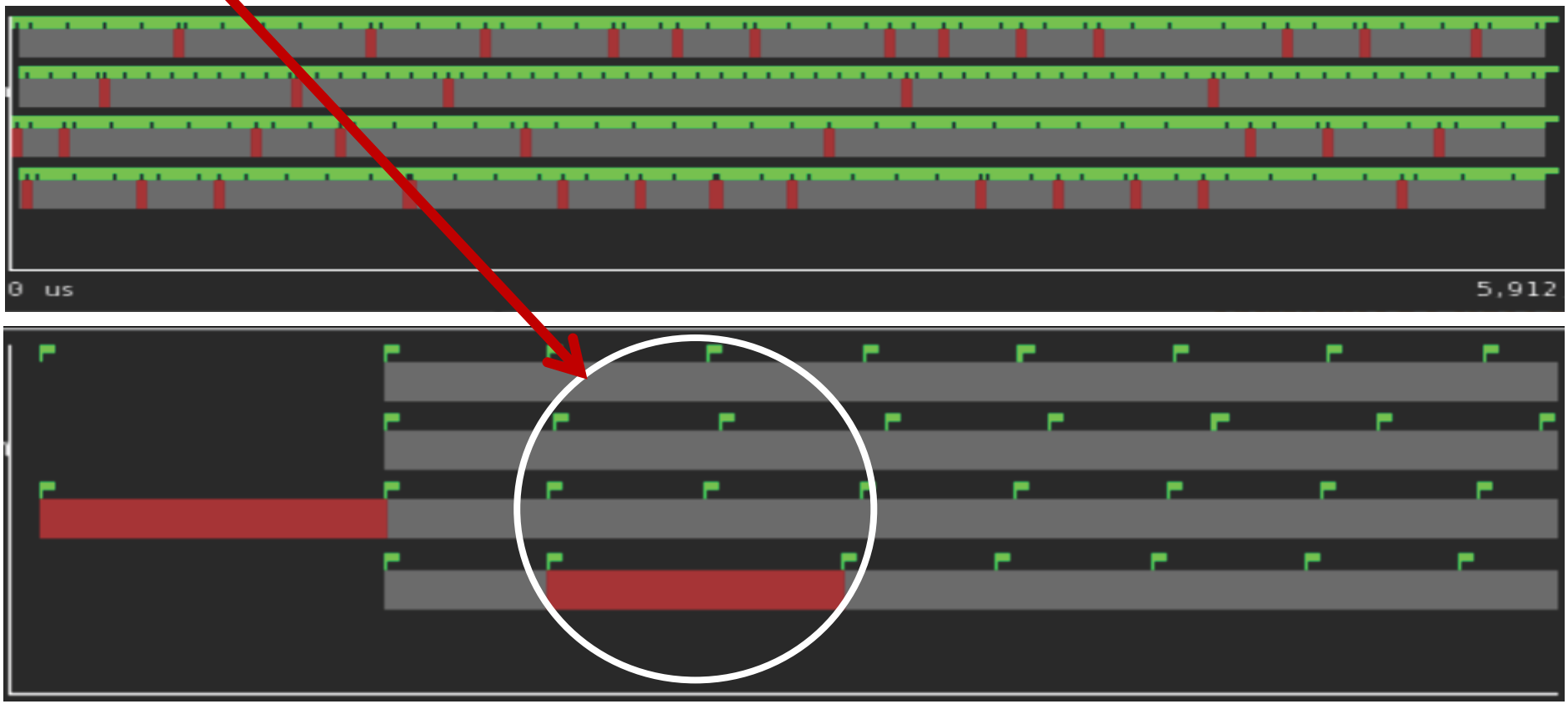
Matrix inversion on CPU-GPU Runtime. OMPSs GJE

- Inversion $A \rightarrow 5{,}120x5{,}120$, AMD Opteron 6172 (4 cores)
- Without look-ahead: 8.512 s



Matrix inversion on CPU-GPU Runtime. OMPSs GJE

- Inversion $A \rightarrow 5{,}120x5{,}120$, AMD Opteron 6172 (4 cores)
- Without look-ahead: 8.512 s
- With look-ahead (OmpSs): 5.912 s



Index

- Motivation
- Model reduction and solvers for matrix equations
- Matrix inversion on CPU-GPU
- Mixed precision and iterative refinement

  "A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms".
  P. Benner, P. Ezzatti, D. Kressner, E. S. Quintana, A. Remón.
  Parallel Computing, 2011

- OOC
- Energy efficiency

Mixed precision and iterative refinement Motivation

- SP faster than DP
  - Intel and AMD FPU (SIMD): 2x
  - NVIDIA (http://www.anandtech.com/show/5699/nvidia-geforce-gtx-680-review):

Geforce	GTX 480	GTX 580 Fermi	GTX 680 Kepler
#cores	480	512	1.536
SP/DP ratio*	12x	8x	24x

* 8x, 2x and ?x in Tesla

  - Texas Instruments DSP: 4x
- Halve volume of data communication (64 → 32 bits)

- Surely not a new idea! (at least, for linear systems)

> *"Progress report on the automatic computing engine".*
> *J. H. Wilkinson.*
> *Report MA/17/1024, Mathematics Division, National Physical Lab., UK, 1948*

Mixed precision and iterative refinement

- Assume L_0 is an initial SP approximation to the solution
- Refinement step:

$$\mathcal{R}(L_k) := AL_kL_k^T + L_kL_k^TA^T + BB^T$$
$$\text{Decompose } \mathcal{R}(L_k) \rightarrow B_+B_+^T - B_-B_-^T$$
$$L_+ := \texttt{ApproxLyap}(B_+), \quad L_- := \texttt{ApproxLyap}(B_-)$$
$$X_{k+1} := L_kL_k^T + L_+L_+^T - L_-L_-^T$$
$$\text{Decompose } X_{k+1} \rightarrow L_{k+1}L_{k+1}^T - \widetilde{L}_-\widetilde{L}_-^T$$

Not explicitely constructed

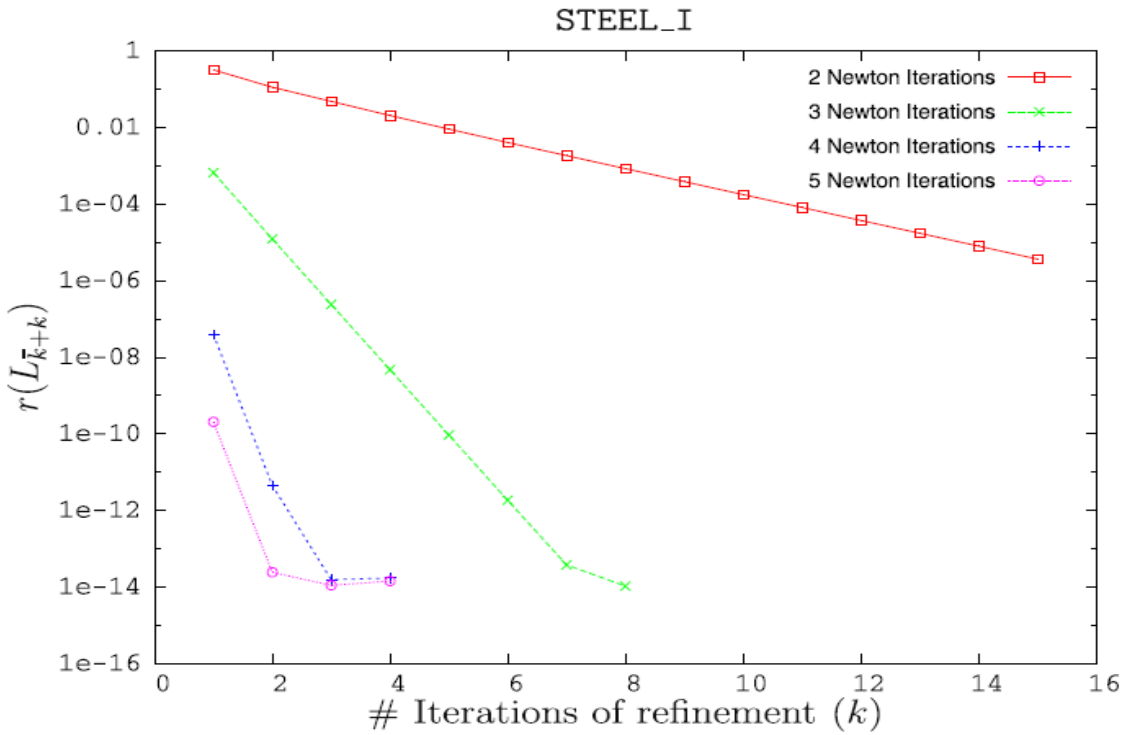
SP

DP, but cheap

  - Requires careful implementation
  - Compute (in SP) and store all matrix inverses: A_0^{-1}, A_1^{-1}, A_2^{-1},... is key to cheap **`ApproxLyap`**
  - Open questions

Mixed precision and iterative refinement

- Optimal cooling of steel profiles. Solve Lyapunov matrix equation $X := F(A,B) = 0, \quad A \rightarrow 5{,}177 x 5{,}177$
  - Double precision: 6 iterations, 13.67s, residual 3.55e-13
  - Mixed precision: 4+3 iterations, 10.19 s, residual: 1.32e-13



Index

- Motivation
- Model reduction and solvers for matrix equations
- Matrix inversion on CPU-GPU
- Mixed precision and iterative refinement
- OOC

  “A run-time system for programming out-of-core matrix algorithms-by-tiles on multithreaded architectures".
  G. Quintana, F. Igual, M. Marqués, E. S. Quintana, R. van de Geijn.
  ACM Trans. on Mathematical Software, 2012

- Energy efficiency

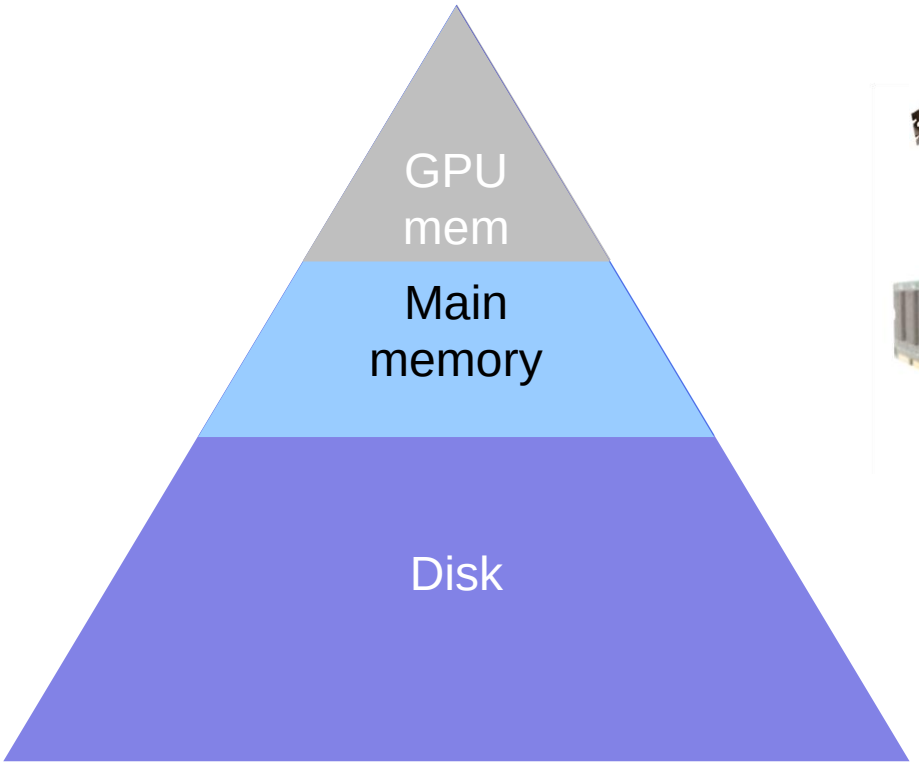
OOC
Motivation

- Memory size of GPU much smaller than that of CPU
  - Speed and cost of GDDR5
  - GTX680: 2,048 Mbytes $\rightarrow n = 16{,}384$
  - Could be enough but… compute and store all matrix inverses: A_0^{-1}, A_1^{-1}, A_2^{-1},... is key to cheap iterative refinement

- OOC dates back to the early days of scientific computing:
  - Memory consisted of magnetic rings and was called *core*
  - Size was a few Kbytes and data was stored on tape

OOC Overview

- Amortize cost of data transfers



n^2 memops (PCI) for $2n^2b$ flops

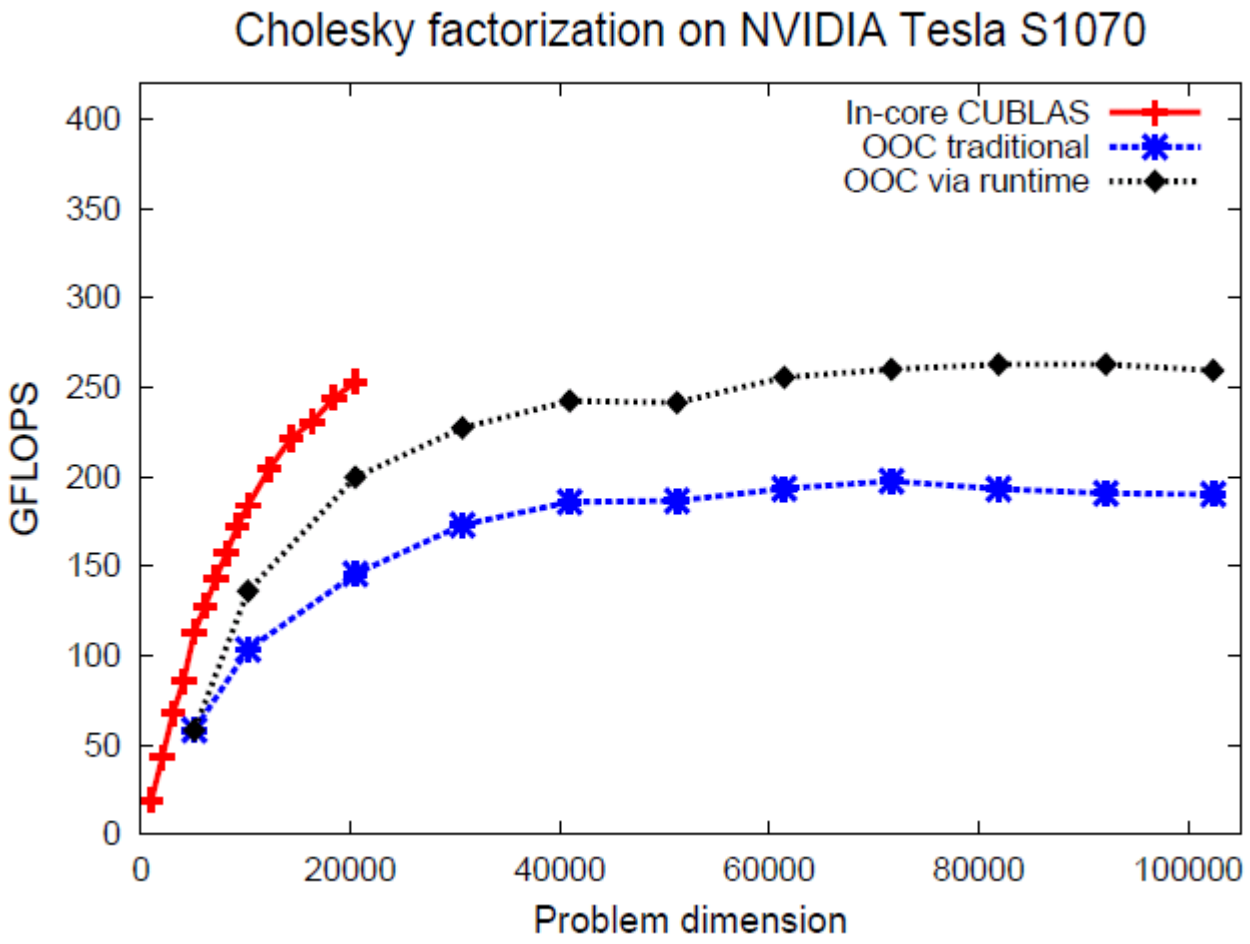
n^2 memops (I/O) for $2n^2t$ flops

OOC
Overview

- Traditional OOC
  - Manually insert I/O instructions into code
- OOC via runtime
  - Software-cache operated by runtime to reduce #disk transfers
  - Asynchronous I/O operated by runtime (perfect data prefetch from disk using code unrolling)
- Implementation
  - libflame+SuperMatrix (library+runtime)

OOC Results

- Cholesky factorization (similar to GJE)



Index

- Motivation
- Model reduction and solvers for matrix equations
- Matrix inversion on CPU-GPU
- Mixed precision and iterative refinement
- OOC
- Energy efficiency

> *"Reducing energy consumption of dense linear algebra operations on hybrid CPU-GPU platforms".*
> *P. Alonso, M. F. Dolz, F. Igual, R. Mayo, E. S. Quintana.*
> *10th IEEE Int. Symp. on Parallel and Distributed Processing with App. - ISPA 2012*

Energy efficiency Motivation

- Green500/Top500 (June 2012)

Rank Green/Top	Site, Computer	#Cores	MFLOPS/W	LINPACK (TFLOPS)	MW to EXAFLOPS?
1/252	DOE/NNSA/LLNL BlueGene/Q, Power BQC 16C 1.60GHz	8,192	2,100.88	86.35	475.99
20/1	DOE/NNSA/LLNL BlueGene/Q, Power BQC 16C 1.60GHz	1,572,864	2,069.04	86.35	483.31

Most powerful reactor under construction in France
Flamanville (EDF, 2017 for US $9 billion):
1,630 MWe

30% !

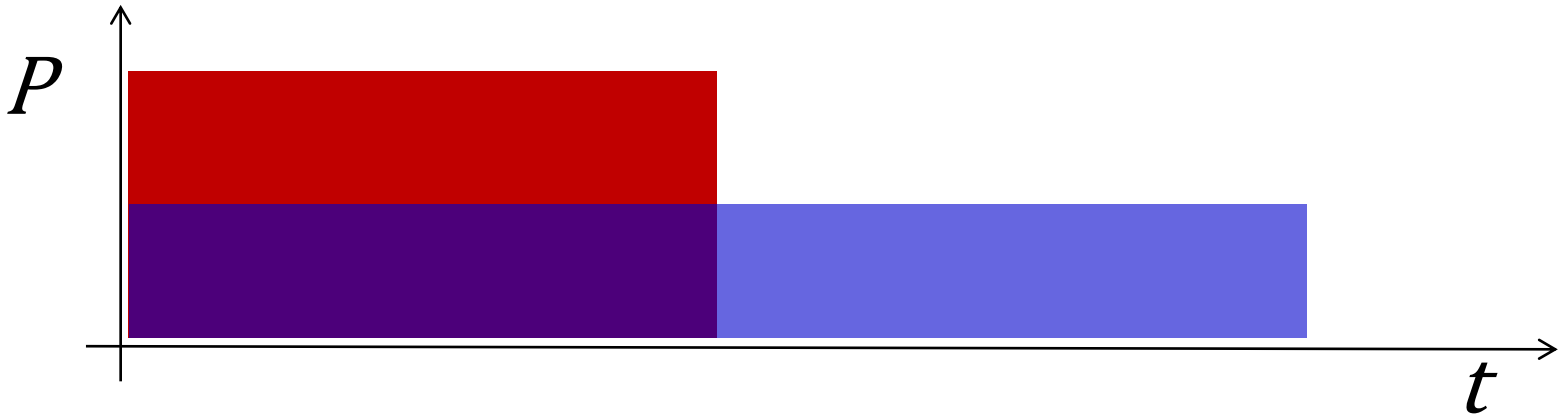
Energy efficiency
Motivation

- Reduce energy consumption!
  - Costs over lifetime of an HPC facility often exceed acquisition costs
  - Carbon dioxide is a hazard for health and environment
  - Heat reduces hw reliability

- Personal view
  - Hardware features energy saving mechanisms:
    - C-states, P-states (DVFS)
  - Scientific apps are in general energy oblivious

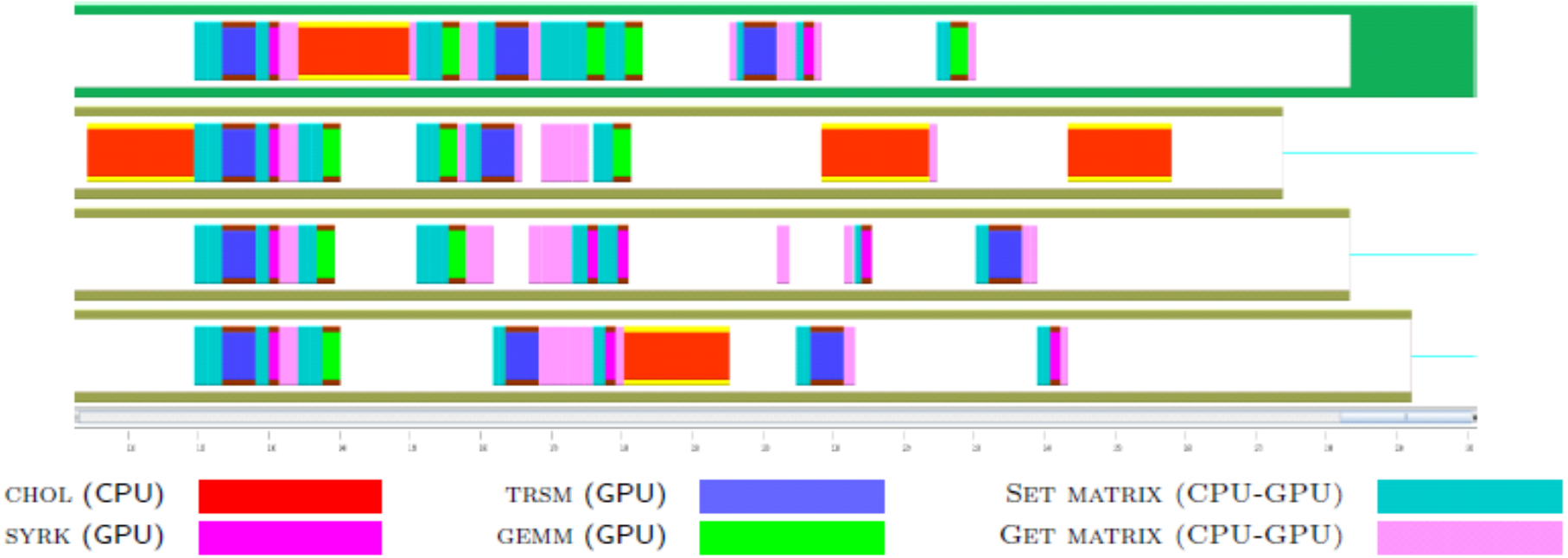
Energy efficiency
Opportunities

- $Energy = \int_{0,T} Power \ dt$
  - Optimizing energy is equivalent to reducing execution time (optimizing performance)
  - Reducing power can be necessary if there are power constraints
  - Reducing power may result in an increase of time (e.g., DVFS)



Energy efficiency Opportunities

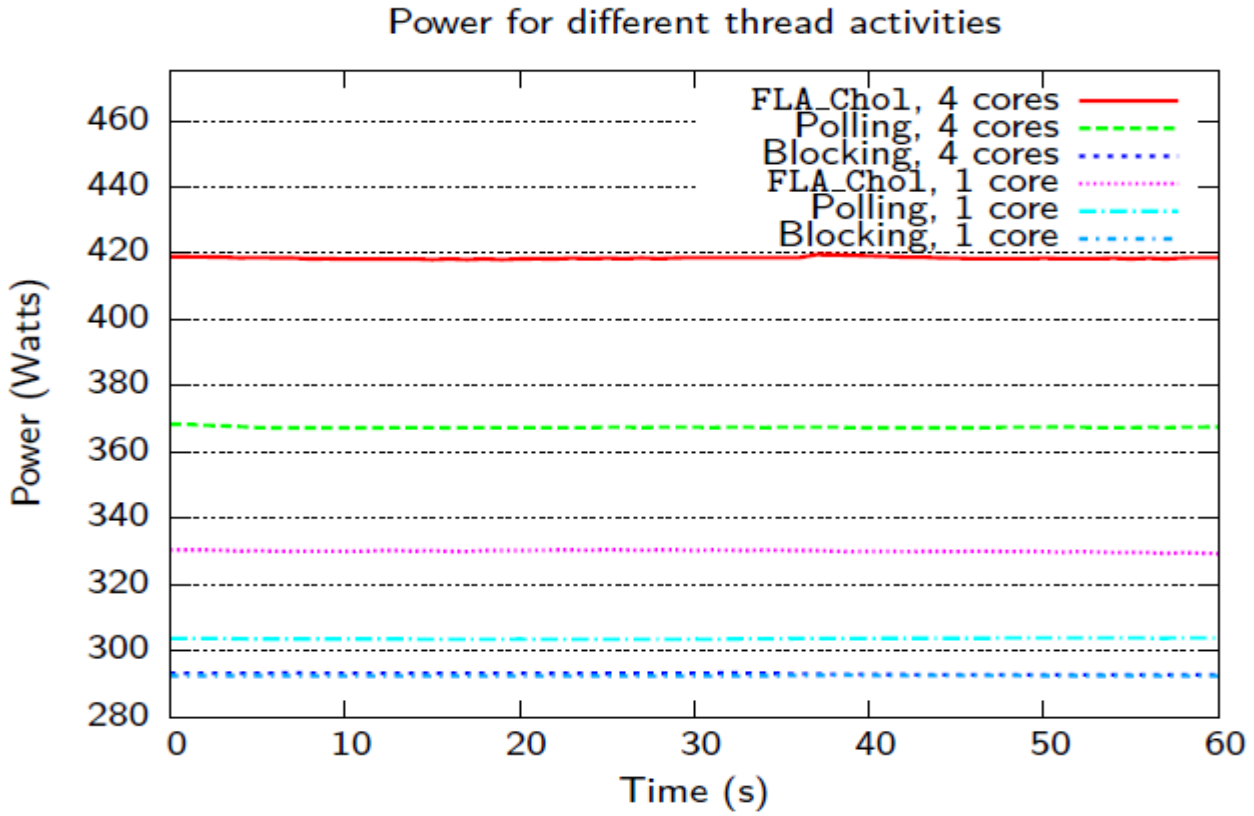
- Cholesky factorization (`FLA_Chol` from libflame, 7,680x7,680) parallelized with SuperMatrix on 4 NVIDIA “Fermi” GPUs



CPU cores inactive during significant time!

Energy efficiency Opportunities

- Cost of “inactivity” (only server, not GPU): polling vs blocking



“*Do nothing, efficiently…*” (V. Pallipadi, A. Belay)
“*Doing nothing well*” (D. E. Culler)

Energy efficiency
Opportunities

- Active polling:
  - CPU thread waiting for work in the ready queue

    → use POSIX semaphores

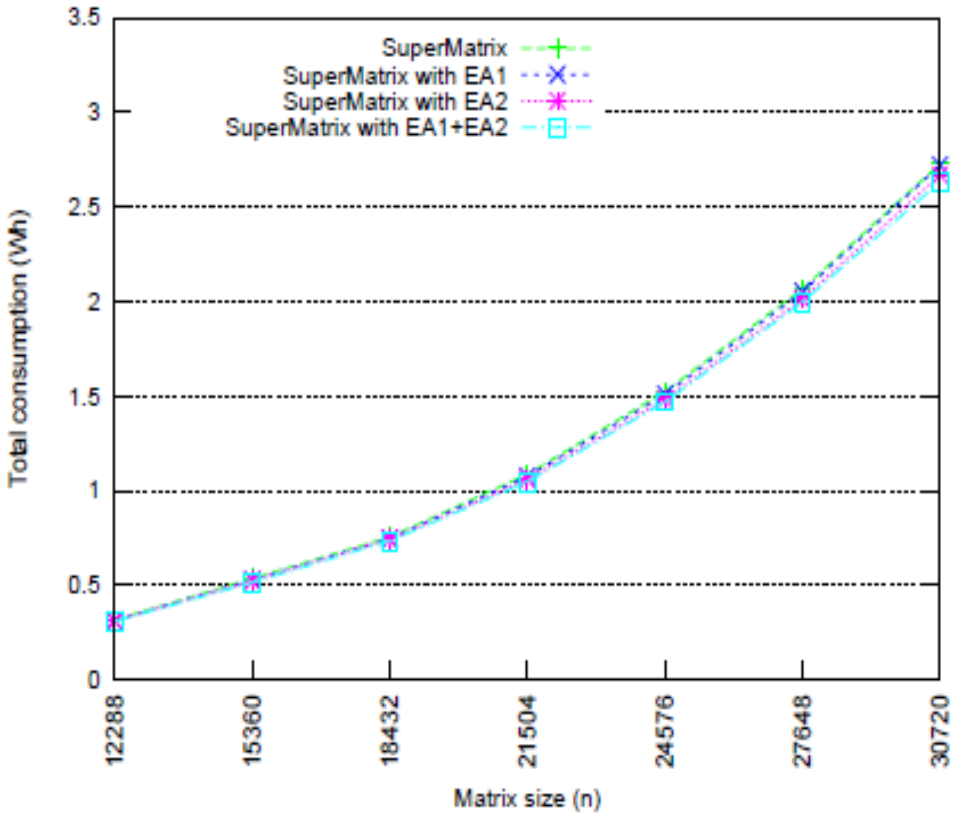
  - CPU thread waiting for a GPU to compute a certain task

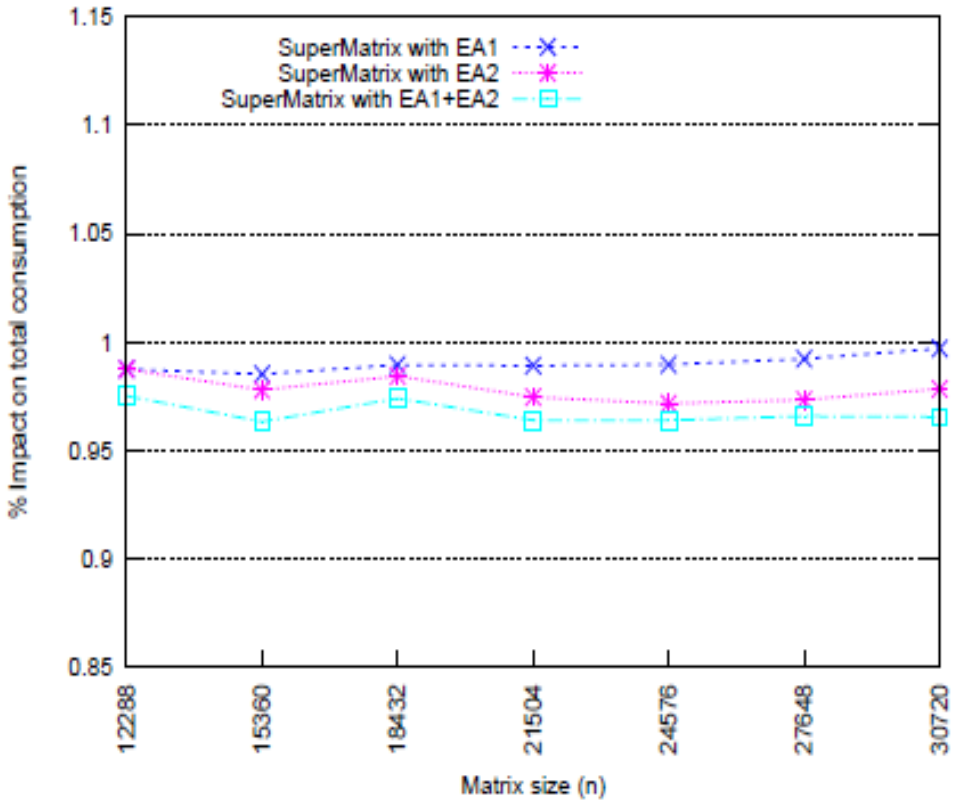
    CUBLAS kernels are asynchronous but, if two kernels are invoked consecutively, the second blocks

    → block CPU thread

Energy efficiency Results

- Cholesky factorization, Intel Xeon E5540+NVIDIA Tesla S2050





Conclusions (summary)

- CPU-GPU platforms: A big leap in less than 10 years for control apps.
- Optimization of control algorithms (dense linear algebra):
  A *deja vú* in HPC
  - Blocked algorithms
  - Concurrent execution (balancing)
  - Overlap computation/communication and avoid “serial” bottlenecks (look-ahead)
  - Reducing communication

Conclusions (summary)

- CPU-GPU platforms: A big leap in less than 10 years for control apps.
- Optimization of control algorithms (dense linear algebra):
  A *deja vú* in HPC
- Advanced techniques can render large benefits:
  - Dynamic (transparent) scheduling via runtime
  - Mixed precision and iterative refinement
  - Out-of-core computing
  - Energy-aware computing

Thanks to…

UJI:	J. I. Aliaga, M. F. Dolz, A. Remón
U. Politécnica de Valencia:	P. Alonso
KIT (Germany):	H. Anzt
BSC (Spain):	R. M. Badia, J. Planas
MPI Magdeburg (Germany):	P. Benner
EPFL (Switzerland):	D. Kressner
Univ. República (Uruguay):	P. Ezzatti
U. Complutense Madrid (Spain):	F. D. Igual
The University of Texas at Austin:	R. van de Geijn

QUESTIONS?

10th International Workshop HeteroPar'2012:

Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms

August 27, 2012, Rhodes Island, Greece