# Dynamically balanced synchronization-avoiding LU factorization with multicore and GPUs

Simplice DONFACK[*] Stanimire TOMOV[†] Jack DONGARRA[‡]

**presenter:** Piotr LUSZCZEK[§]

[*]formerly: University of Tennessee, currently: CSCS Lugano, Switzerland

[†]University of Tennessee

[‡]University of Tennessee, Oak Ridge National Laboratory, and University of Manchester

[§]University of Tennessee

## HPC Hardware Zoo

- Intel
  - x86 tick-tock: ➥ Nehalem ➥ Westmere ➥ Sandy Bridge ➥ Ivy Bridge ➥ Haswell ➥ Broadwell
  - MIC/Phi core-counts: Knights Corner: 57, 62, . . .
- AMD
  - x86 architectures: ➥ Bulldozer ➥ Piledriver
  - x86 models: ➥ Barcelona ➥ Shanghai ➥ Istanbul ➥ Magny-Cours ➥ Warsaw ➥ Seattle
- NVIDIA: ➥ Tesla ➥ Fermi ➥ Kepler
- Per-core flop/s: 10, 20, 40
- Per-socket flop/s: 100 – 600
- Per-accelerator flop/s: 500 – 1500

Balance between CPU and accelerator: 2x – 10x

## Motivation for Communication Avoiding Algorithm

- Running time is a function of :
  - Time for arithmetic operations $=$ Total(flops) $\times$ time/flop.
  - Time for moving data $=$ Total(messages) $\times$ latency $+$ Total(bytes) / bandwidth.
- Exponentially growing gaps between communication and computation.
  - Annual improvements predictions [FOSC'04].

| time/flop | | Bandwidth | Latency |
|---|---|---|---|
| 59% | Network | 26% | 15% |
| | DRAM | 23% | 5% |

## Communication avoiding algorithms:

- aim at reducing communication by doing some redundant computations.
  - Work more, talk less.
- are becoming a part of the numerical algorithm design.

## Communication avoiding LU (CALU):

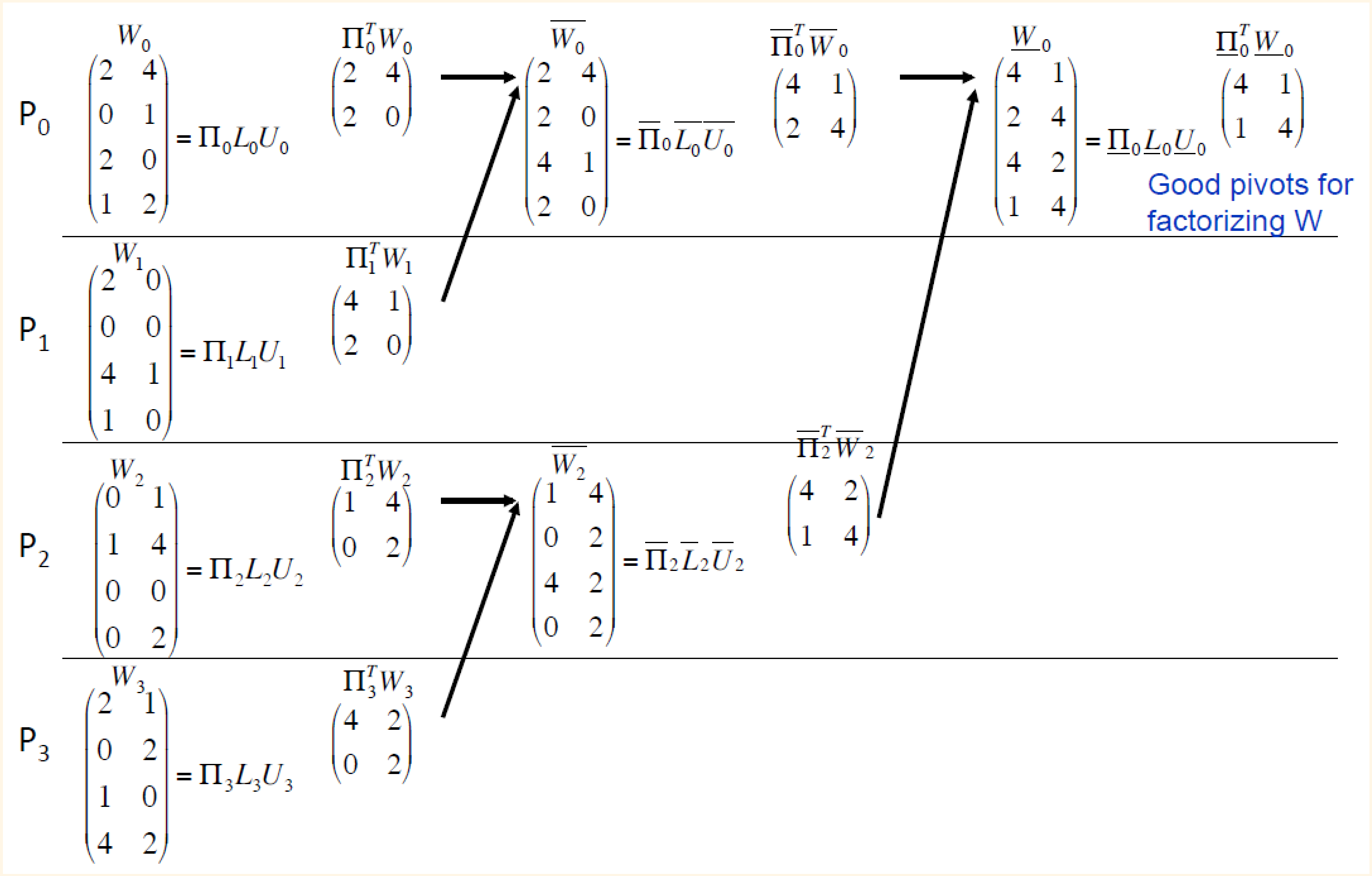
- removes the bottleneck in classic LU by performing the panel as a reduction operation.
  - Tournament pivoting replaces partial pivoting.
- factorizes the panel twice.

## CALU [Grigori, Demmel, Xiang '08]

The main difference with classic approach lies on the panel factorization. The panel factorization is performed in two steps.

- A preprocessing step aims at identifying at low communication cost good pivot rows.
- The pivot rows are permuted in the first positions of the panel and LU without pivoting of the panel is performed.
- The update of the trailing matrix is performed as in classic LU (Gaussian Elimination with Partial Pivoting – GEPP).
- The main difference lies on the panel factorization. In classic approach as ScaLAPACK, panel is factorized column by column, while with CALU it is factorized block by block using a reduction tree.
- The algorithm was first introduce for QR. The obvious generalization of CAQR to CALU was not stable in practice. CALU uses a new pivoting strategy.
- CALU is stable in practice (and so is classic LU).

# CALU's Tournament Pivoting


P$_0$
$W_0 = \begin{pmatrix} 2 & 4 \\ 0 & 1 \\ 2 & 0 \\ 1 & 2 \end{pmatrix} = \Pi_0 L_0 U_0$
$\Pi_0^T W_0 = \begin{pmatrix} 2 & 4 \\ 2 & 0 \end{pmatrix}$
$\overline{W_0} = \begin{pmatrix} 2 & 4 \\ 2 & 0 \\ 4 & 1 \\ 2 & 0 \end{pmatrix} = \overline{\Pi_0}\,\overline{L_0}\,\overline{U_0}$
$\overline{\Pi_0}^T \overline{W_0} = \begin{pmatrix} 4 & 1 \\ 2 & 4 \end{pmatrix}$
$\underline{W_0} = \begin{pmatrix} 4 & 1 \\ 2 & 4 \\ 4 & 2 \\ 1 & 4 \end{pmatrix} = \underline{\Pi_0}\,\underline{L_0}\,\underline{U_0}$
$\underline{\Pi_0}^T \underline{W_0} = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$
Good pivots for factorizing W
P$_1$
$W_1 = \begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 4 & 1 \\ 1 & 0 \end{pmatrix} = \Pi_1 L_1 U_1$
$\Pi_1^T W_1 = \begin{pmatrix} 4 & 1 \\ 2 & 0 \end{pmatrix}$
P$_2$
$W_2 = \begin{pmatrix} 0 & 1 \\ 1 & 4 \\ 0 & 0 \\ 0 & 2 \end{pmatrix} = \Pi_2 L_2 U_2$
$\Pi_2^T W_2 = \begin{pmatrix} 1 & 4 \\ 0 & 2 \end{pmatrix}$
$\overline{W_2} = \begin{pmatrix} 1 & 4 \\ 0 & 2 \\ 4 & 2 \\ 0 & 2 \end{pmatrix} = \overline{\Pi_2}\,\overline{L_2}\,\overline{U_2}$
$\overline{\Pi_2}^T \overline{W_2} = \begin{pmatrix} 4 & 2 \\ 1 & 4 \end{pmatrix}$
P$_3$
$W_3 = \begin{pmatrix} 2 & 1 \\ 0 & 2 \\ 1 & 0 \\ 4 & 2 \end{pmatrix} = \Pi_3 L_3 U_3$
$\Pi_3^T W_3 = \begin{pmatrix} 4 & 2 \\ 0 & 2 \end{pmatrix}$

## Communication Avoiding Algorithm Lowers Bounds

- General lower bounds for all direct linear algebra.
  - Total(bytes moved) = $\Omega(\frac{\text{Total(flops)}}{\sqrt{M}}) = \Omega(\frac{n^2}{\sqrt{P}})$
  - Total(messages) = $\Omega(\frac{\text{Total(flops)}}{M\sqrt{M}})$ [Ballard, Demmel, Holtz, Schwartz '11]
- Performance model of CALU, PDGETRF with optimal layout for general matrix. $M = O(\frac{n^2}{P})$

| | PDGETRF | CALU | Optimal Lower bounds |
|---|---|---|---|
| Total(messages) | $n \log P + \frac{3}{2}\sqrt{P}\log P$ | $3\sqrt{P}\log^3 P$ | $\Omega(\sqrt{P})$ |
| Total(words) | $\frac{n^2}{\sqrt{P}}\log P$ | $\frac{n^2}{\sqrt{P}}\log P$ | $\Omega(\frac{n^2}{\sqrt{P}})$ |
| Total(flops) | $\frac{2}{3}\frac{n^3}{P}$ | $\frac{2}{3}\frac{n^3}{P} + O(\frac{n^3}{P\log^2 P})$ | $\frac{2}{3}\frac{n^3}{P}$ |

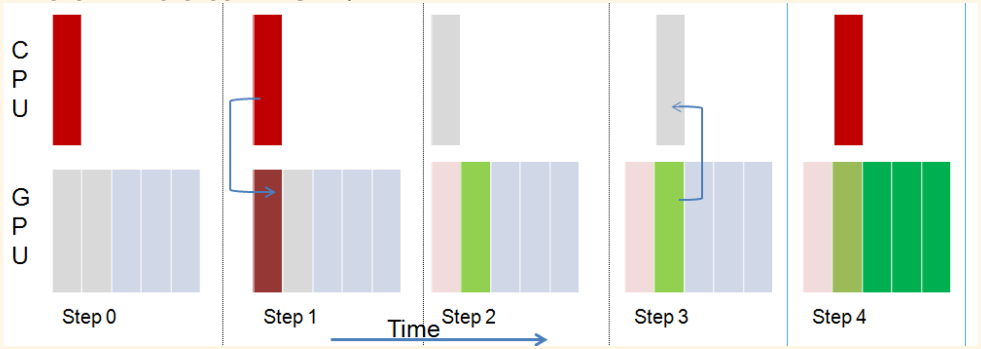
# MAGMA's Approach to LU Factorization

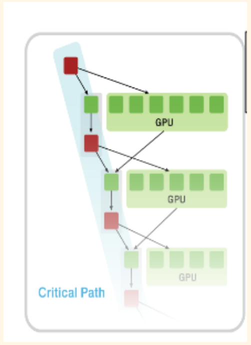
- MAGMA = Matrix Algebra on GPU and Multicore Architectures
- Hybrid LU factorization in MAGMA
  - Panel are factorized on the CPUs.
  - Update of the trailing submatrices are performed on the GPUs.

Example of execution of **magma_dgetrf()** on a square matrix in 4 steps.

matrix/data view:

DAG view:





- Efficient updates and optimal use of the GPUs.

- Load imbalance between CPUs and GPUs.
- Poor multicore scalability.

## CALU for MAGMA

**First goal**

- Adapt and evaluate CALU as panel factorization in MAGMA.

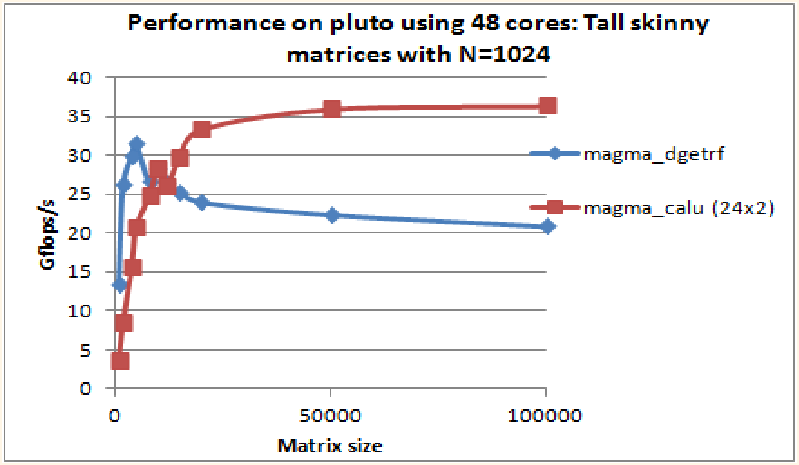
**Approach**

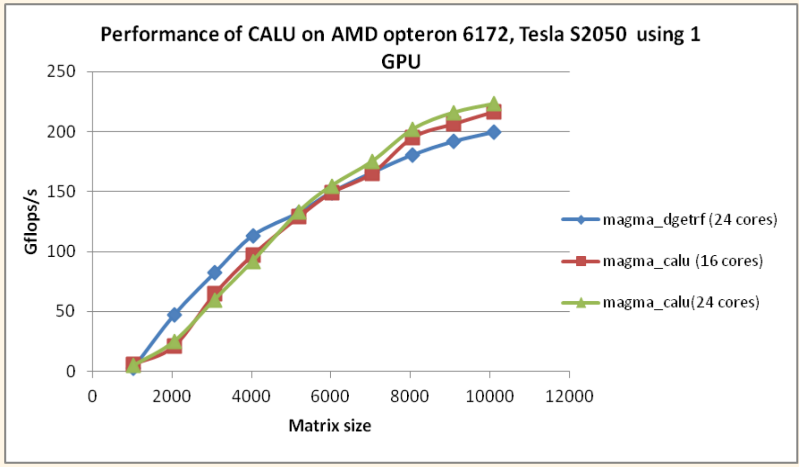
- Replace standard panel factorization in MAGMA with CALU.
- Increase then panel block size $B$ to improve the load balance.
- Introduce two (algorithmic) block sizes:
  - panel block size $B$, and
  - internal block size $ib$ for CALU.

## MAGMA approach with CALU as panel: Initial results

First performance results on AMD Opteron 6172

- 4 sockets
- 12 cores @2.1Ghz
- Peak performance CPU: 403.2Gflops/s
- NVIDIA Fermi GPU: 504 Gflops/s
- Total: 907.2G flops/s.

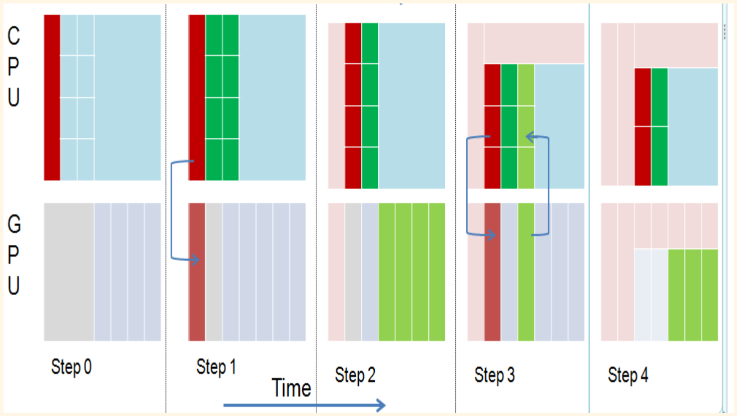




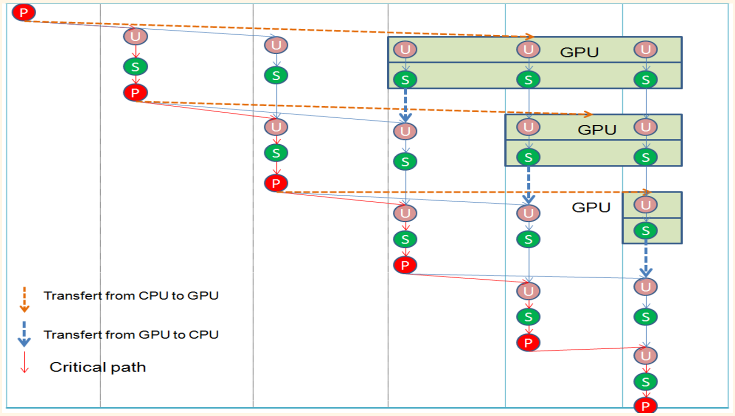
Fast panel factorization technique is not enough.

## Balanced Approach to Accelerated CALU

- The matrix is partitioned into two parts for the CPUs and the GPU.
- Each factorized panel is asynchronously sent to the GPU.
- A block column is dynamically sent to the CPUs during the runtime to balance work.



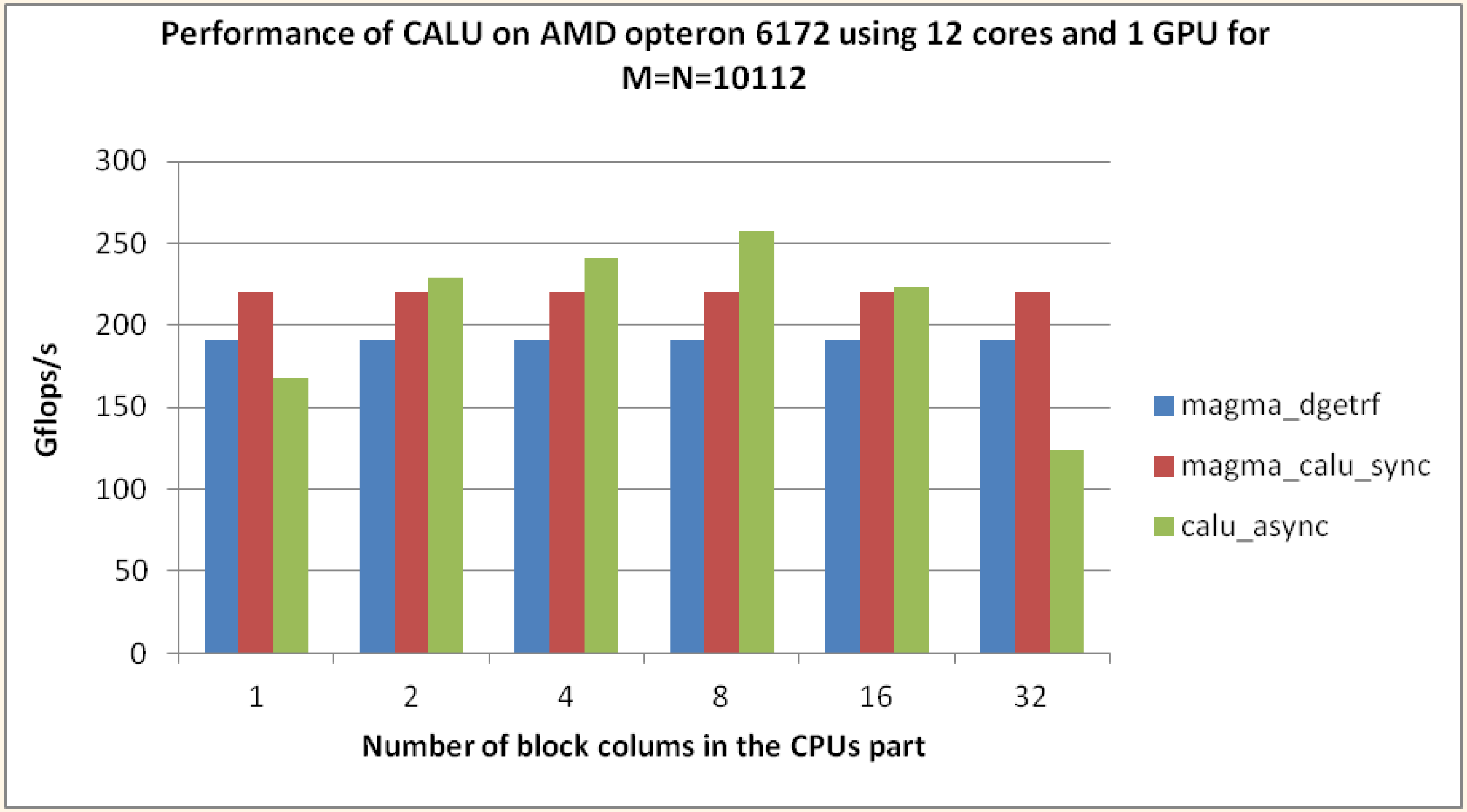
a. Example of execution.



b. Corresponding DAG.

# Performance of Asynchronous CALU with Fixed Parameters

Variants of CALU on AMD Opteron 6172 using 12 cores and 1 GPU:



Results on: ✧ AMD Opteron 6172 ✧ 4 sockets ✧ 12 cores @2.1Ghz ✧ Peak performance CPU: 403.2Gflops/s ✧ NVIDIA Fermi GPU: 504 Gflops/s ✧ Total: 907.2 Gflops/s.

How to determine the initial amount of work for the CPUs part?

## Performance Model Parameters

Global parameters:

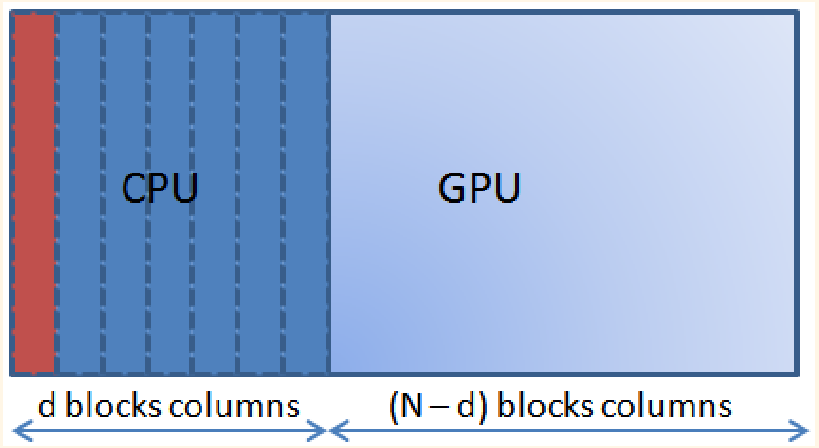
- $d$ — the number of block column in the CPU's part.
- $P$ — the number of processors for the CPU's part.
- $g_1$ and $g_2$ — the peak performance of one CPU and one GPU respectively.

At each step of the factorization $K$, temporal parameters:

- $N_K$ — the number of block column of the remaining matrix.
- $W_{CPUs}$ and $W_{GPU}$ — the amount of work required to compute the CPU's part and GPU's part, respectively.
- $T_{CPUs}$ and $T_{GPU}$ — the time required to complete $W_{CPUs}$ and $W_{GPU}$, respectively.

## Performance Model's Details

Initial matrix decomposition:



$W_{\text{CPUs}} = W_{1\text{panel}} + (d-1)W_{1\text{update}}$ and $T_{\text{CPUs}} = \frac{W_{\text{CPUs}}}{P \times g_1}$

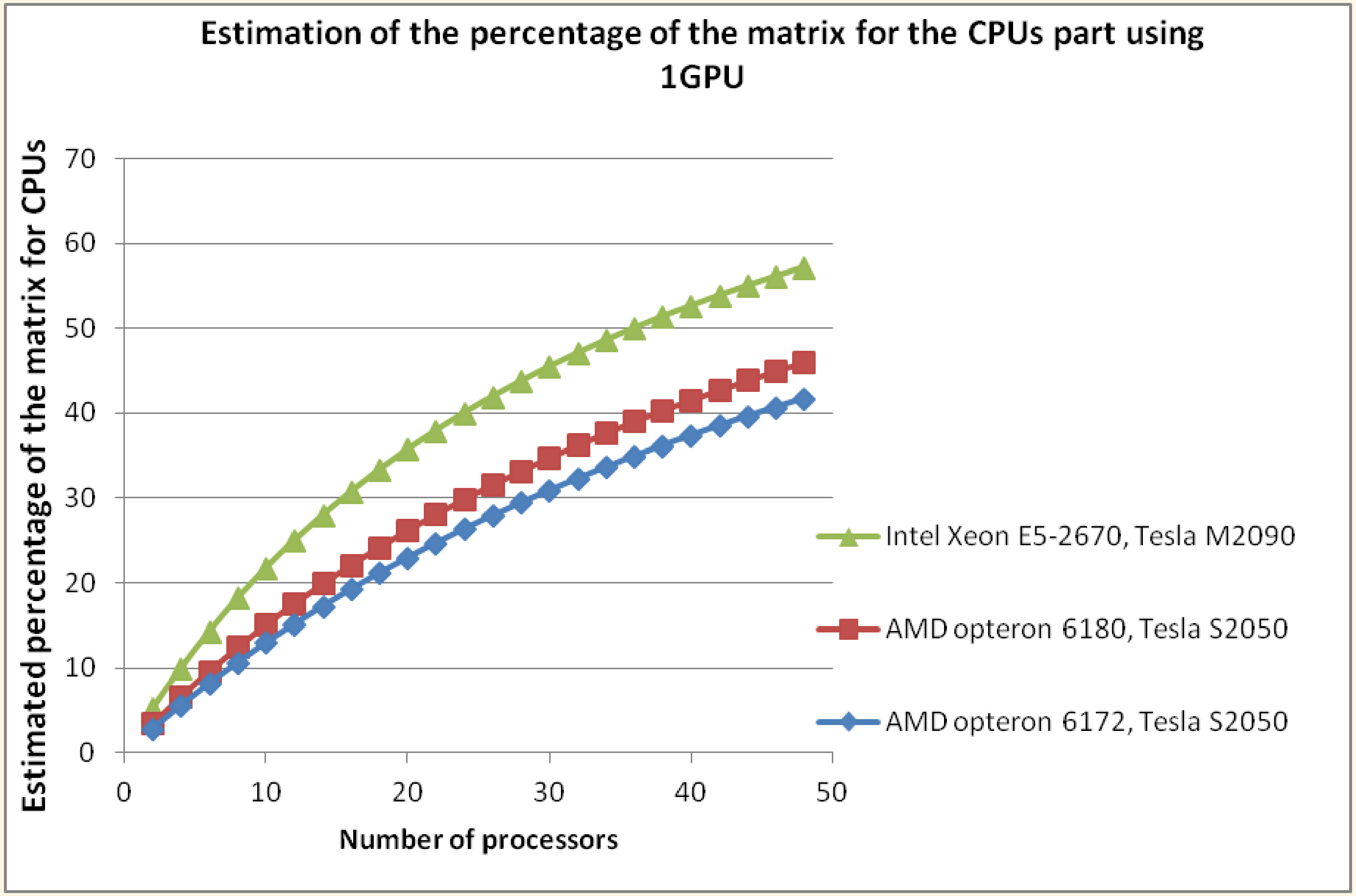
$W_{\text{GPU}} = (N_K - d)W_{1\text{update}}$ and $T_{\text{GPU}} = \frac{W_{\text{GPUs}}}{g_2}$

By solving $T_{\text{CPUs}} = T_{\text{GPU}}$, we obtain:

$$\frac{d}{N_K} = \frac{Pg_1}{Pg_1 + g_2}$$

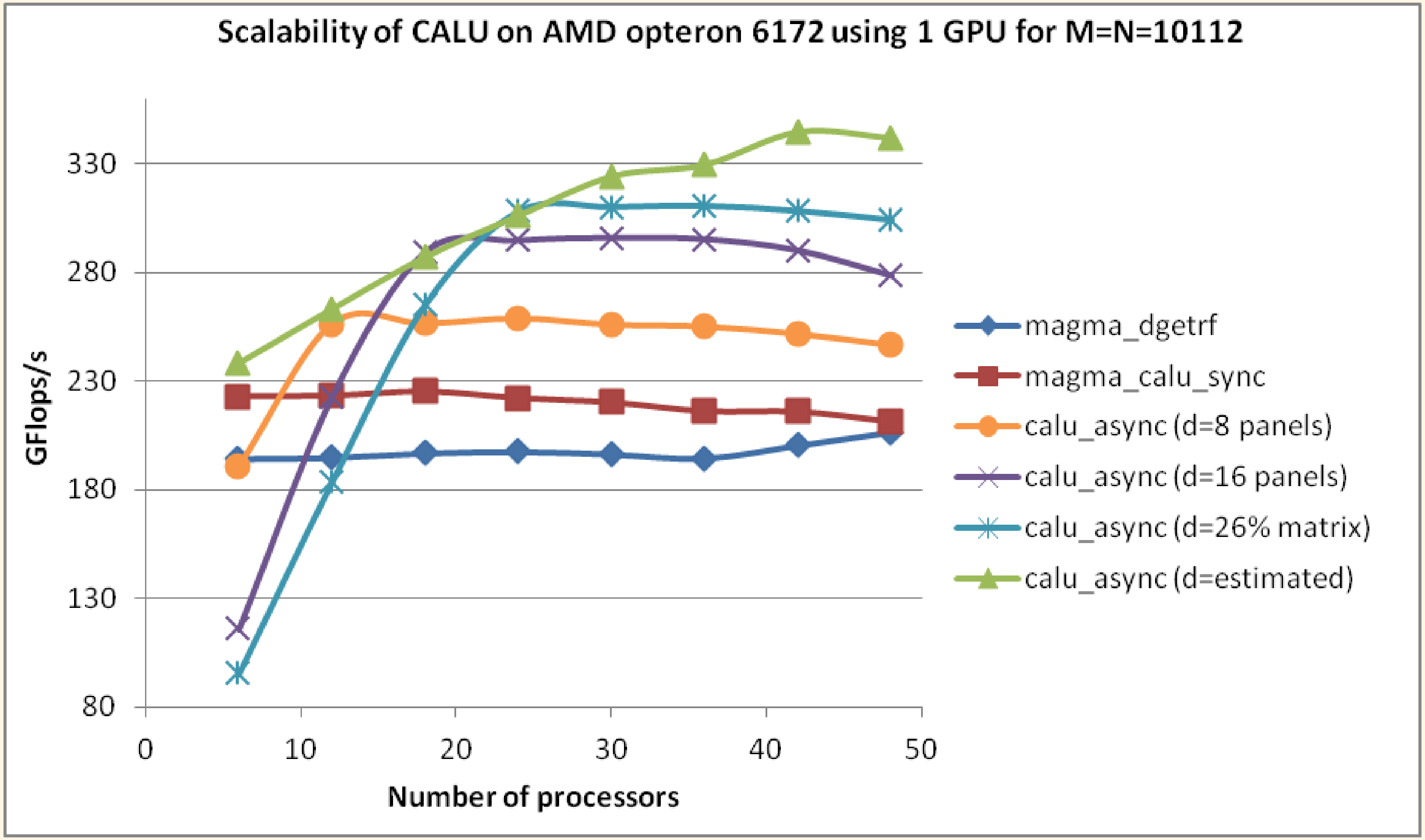
$\frac{d}{N_K}$ represents the percentage of the matrix to assign to the CPUs.

## Performance Model's Prediction



- AMD Opteron 6172: 4x12 cores @2.1Ghz; Peak performance CPU: 403.2 Gflops/s, GPU: 504 Gflops/s, Total: 907.2 Gflops/s.
- AMD Opteron 6180: 4x12 cores @2.5Ghz; Peak performance CPU: 480.0 Gflops/s, GPU: 504 Gflops/s, Total: 984.0 Gflops/s.
- Intel Xeon E5-2670: 2x8 cores @2.6Ghz; Peak performance CPU: 332.8 Gflops/s, GPU: 665 Gflops/s, Total: 997.8 Gflops/s.

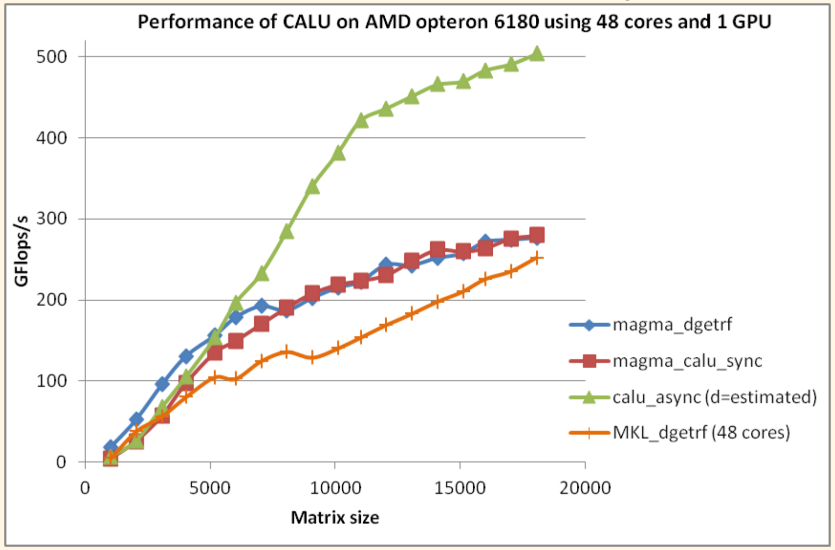
## Scalability Experiments

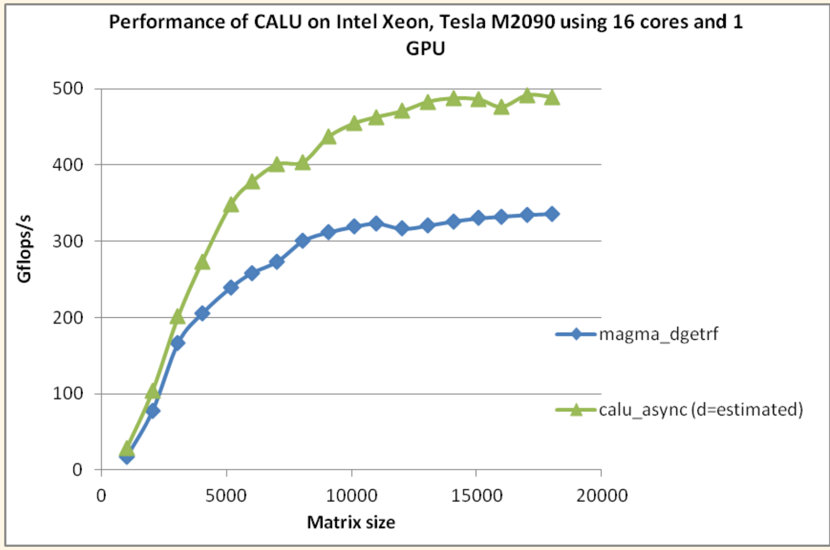


- AMD Opteron 6172: 4x12 cores @2.1Ghz; Peak performance CPU: 403.2 Gflops/s, GPU: 504 Gflops/s, Total: 907.2 Gflops/s.

# Performance of Asynchronous CALU with Estimated Parameters

Performance of CALU for square matrices.

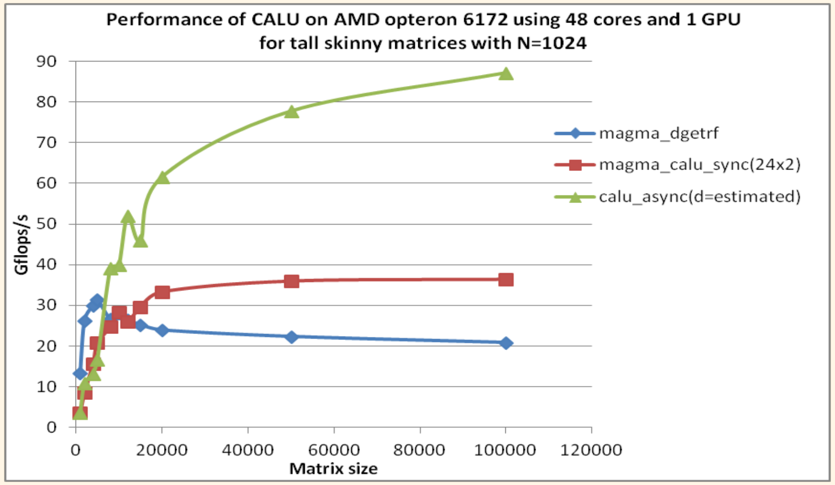


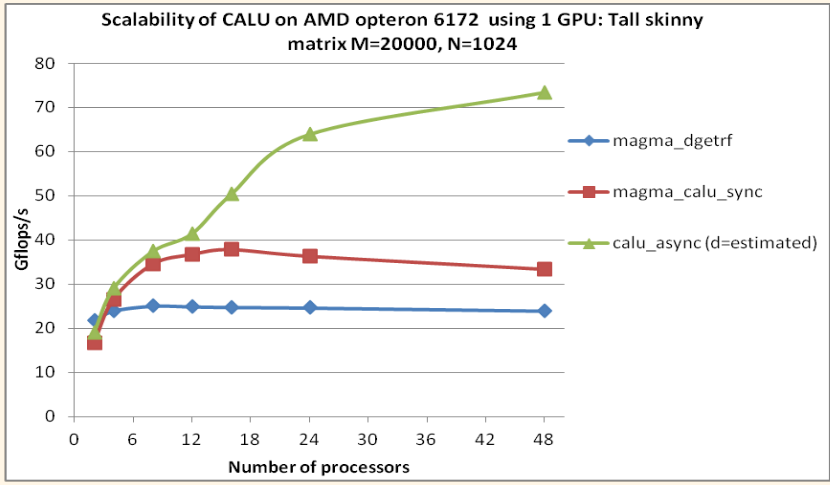


- AMD Opteron 6180: 4x12 cores @2.5Ghz; Peak performance CPU: 480.0 Gflops/s, GPU: 504 Gflops/s, Total: 984.0 Gflops/s.
- Intel Xeon E5-2670: 2x8 cores @2.6Ghz; Peak performance CPU: 332.8 Gflops/s, GPU: 665 Gflops/s, Total: 997.8 Gflops/s.

# Scalability of Asynchronous CALU for Tall-and-Skinny Matrices

Performance and scalability using 48 cores.





Results on: ✧ AMD Opteron 6172 ✧ 4 sockets ✧ 12 cores @2.1Ghz ✧ Peak performance CPU: 403.2Gflops/s ✧ NVIDIA Fermi GPU: 504 Gflops/s ✧ Total: 907.2 Gflops/s.

## Summary, Conclusions, and Future Work

**Contributions:**

- Accelerated CALU LU factorization for a wide range of CPU-GPU hardware combinations.
- Efficient and scalable implementation for tens of CPU cores.
- Simple model that makes the algorithm self-adapting in practice.

**Possible extensions:**

- Integrate dynamic load-balancing using runtime schedulers such as QUARK.
- Extend the approach to other algorithms
  - Recursive parallel panel LU, RRLU, QR, CAQR.
  - Two-sided factorizations: symmetric eigenvalues, SVD reduction.
    - Please attend my Friday's talk.
  - Support for multiple GPUs.
  - Support for hetergeneous accelerator configurations.
    - Please attend my Tuesday's talk.