HPC with Multicore and GPUs

Stanimire Tomov and Jack Dongarra
Innovative Computing Laboratory
University of Tennessee, Knoxville

TAMU visit

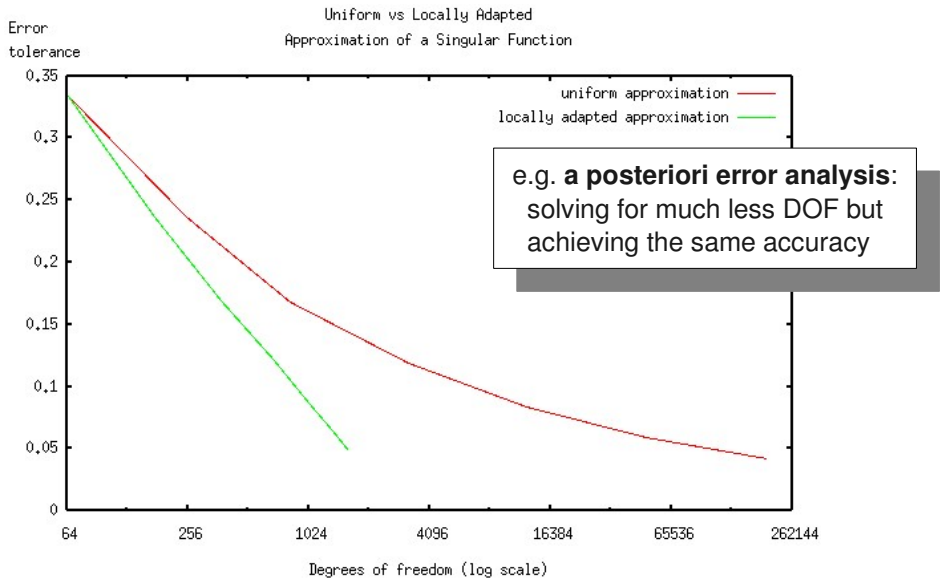
December 3-4 College Station, TX

Outline

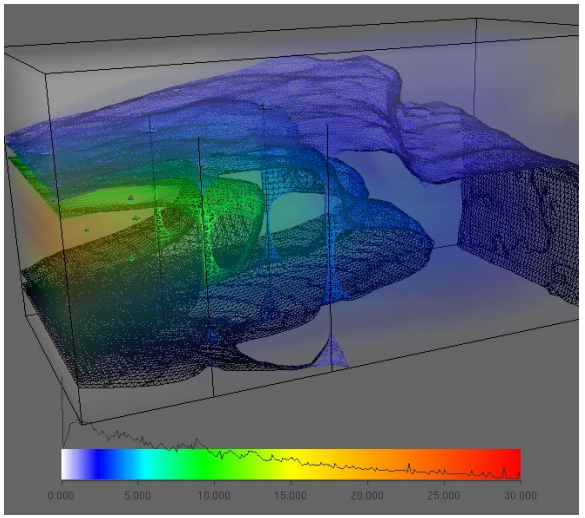
- **Introduction**
  - Hardware to Software Trends
- **LAPACK to MAGMA / PLASMA Libraries**
  - Challenges and approach
  - One-sided and two-sided factorizations
- **Performance Results**
- **Conclusions**

Speeding up Computer Simulations

Better numerical methods

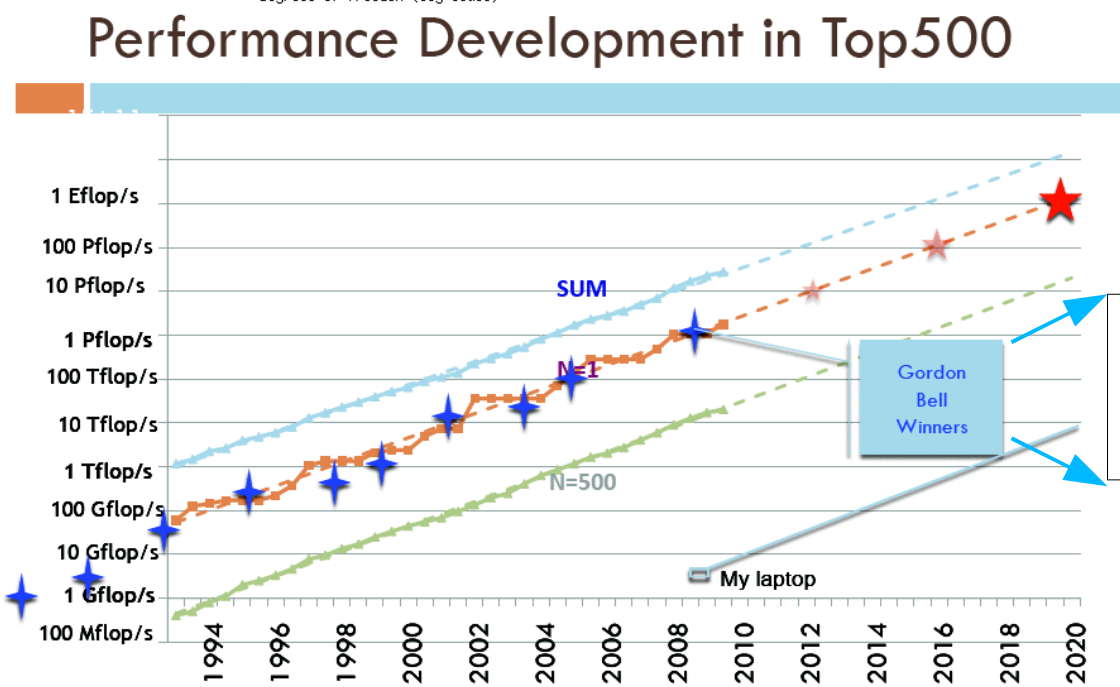


e.g. **a posteriori error analysis**: solving for much less DOF but achieving the same accuracy



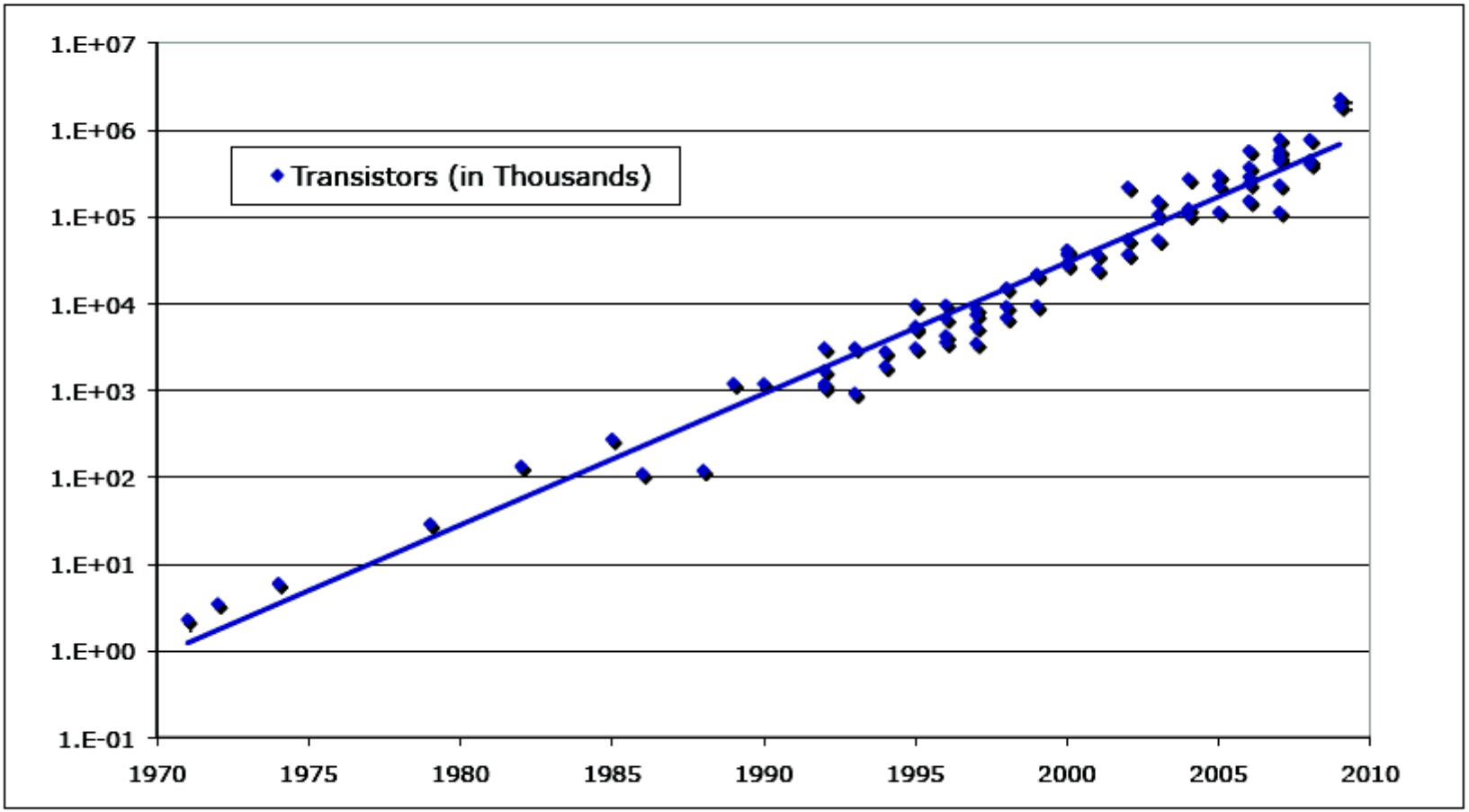
http://www.cs.utk.edu/~tomov/cflow/

Exploit advances in hardware



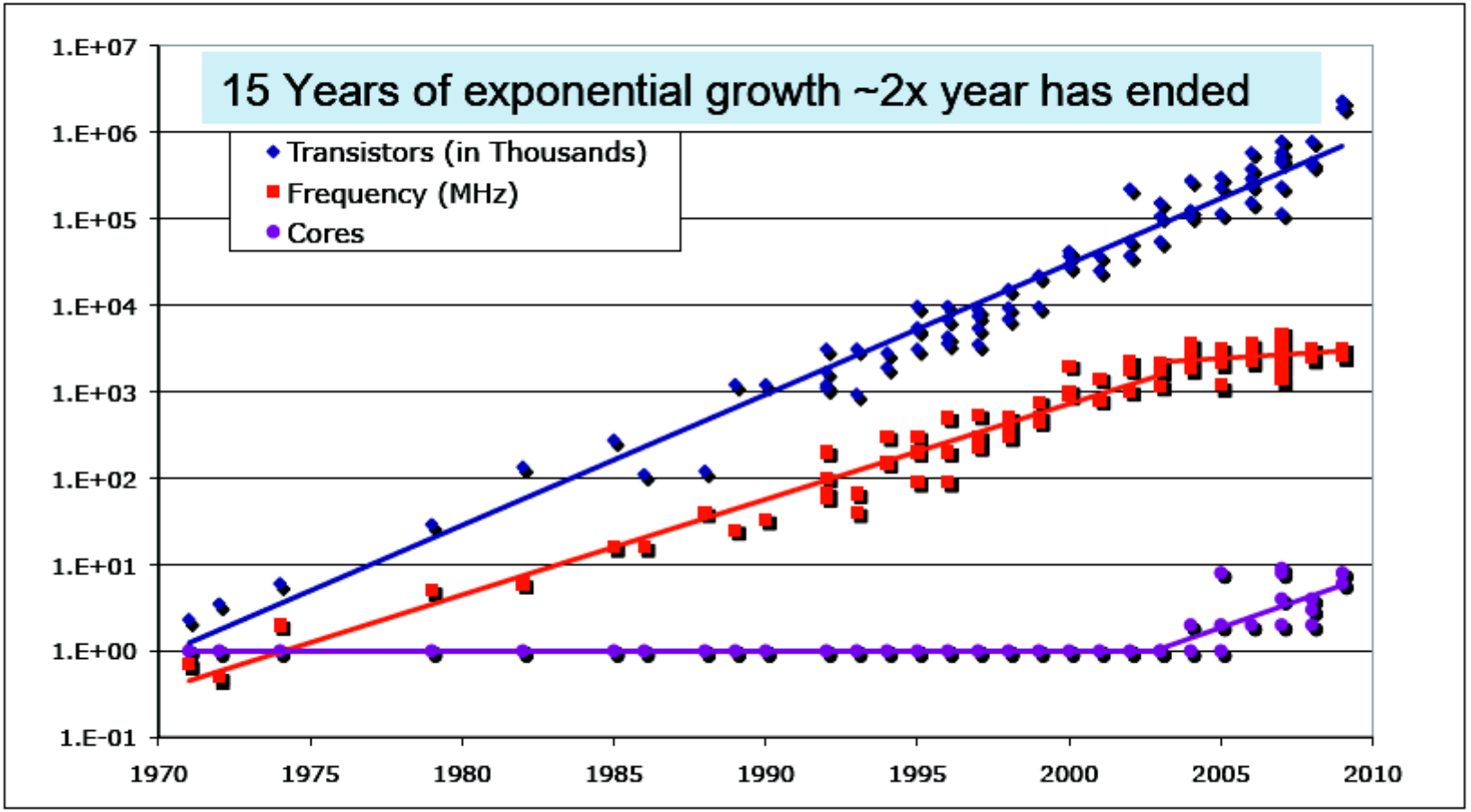
- Manage to use hardware efficiently for real-world HPC applications
- Match LU benchmark in performance !

Moore's Law is Alive and Well



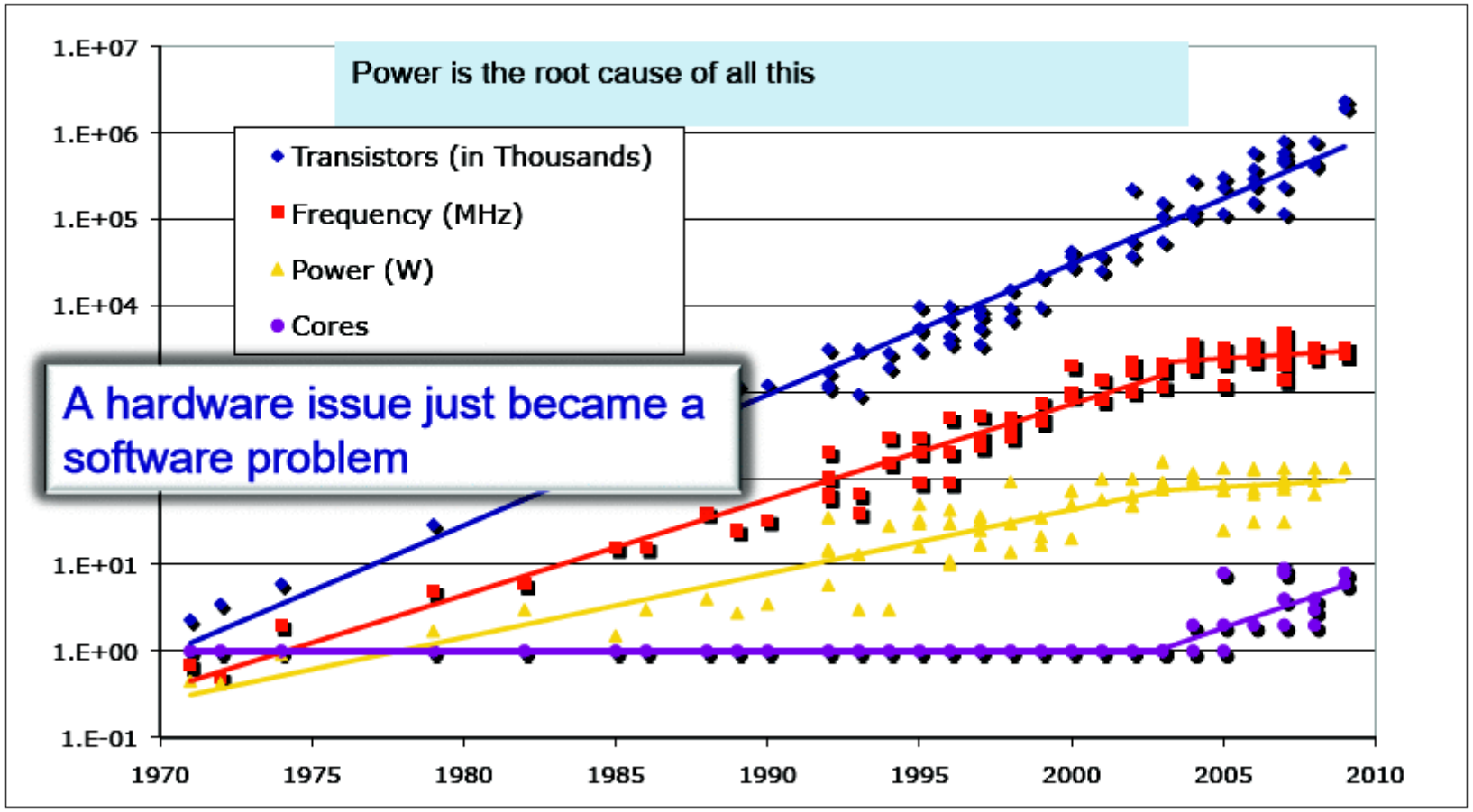
Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

But Clock Frequency Scaling Replaced by Scaling Cores / Chip



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

Performance Has Also Slowed, Along with Power



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

Power Cost of Frequency

- Power ∝ Voltage^2 x Frequency (V^{2}F)
- Frequency ∝ Voltage
- Power ∝ Frequency^3

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
"New" Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X
Multicore	2X	0.75X	0.75X	1.5X	0.8X	1.88X

(Bigger # is better)

A multicore chip is a single chip (socket) that combines two or more independent processing units that provide independent threads of control

50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device

Why GPU-based Computing ?

- Hardware Trends

Hardware

INCREASE IN PARALLELISM

INCREASE IN COMMUNICATION COST (vs COMPUTATION)

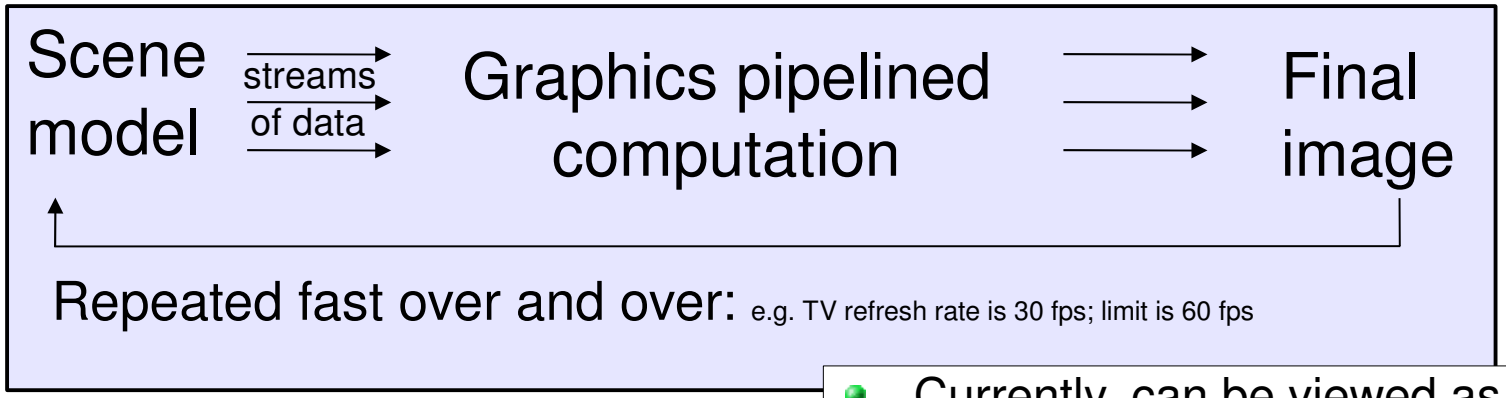
Hybrid / Heterogeneous Designs

Multicore + GPUs

Processor speed improves 59% / year
but memory bandwidth only by 23%
latency by 5.5%

Evolution of GPUs

GPUs: excelling in graphics rendering



- Currently, can be viewed as **multithreaded multicore vector units**

This type of computation:

- Requires **enormous computational power**
- Allows for **high parallelism**
- Needs **high bandwidth *vs* low latency**

(as low latencies can be compensated with deep graphics pipeline)

Obviously, this pattern of computation is common with many other applications

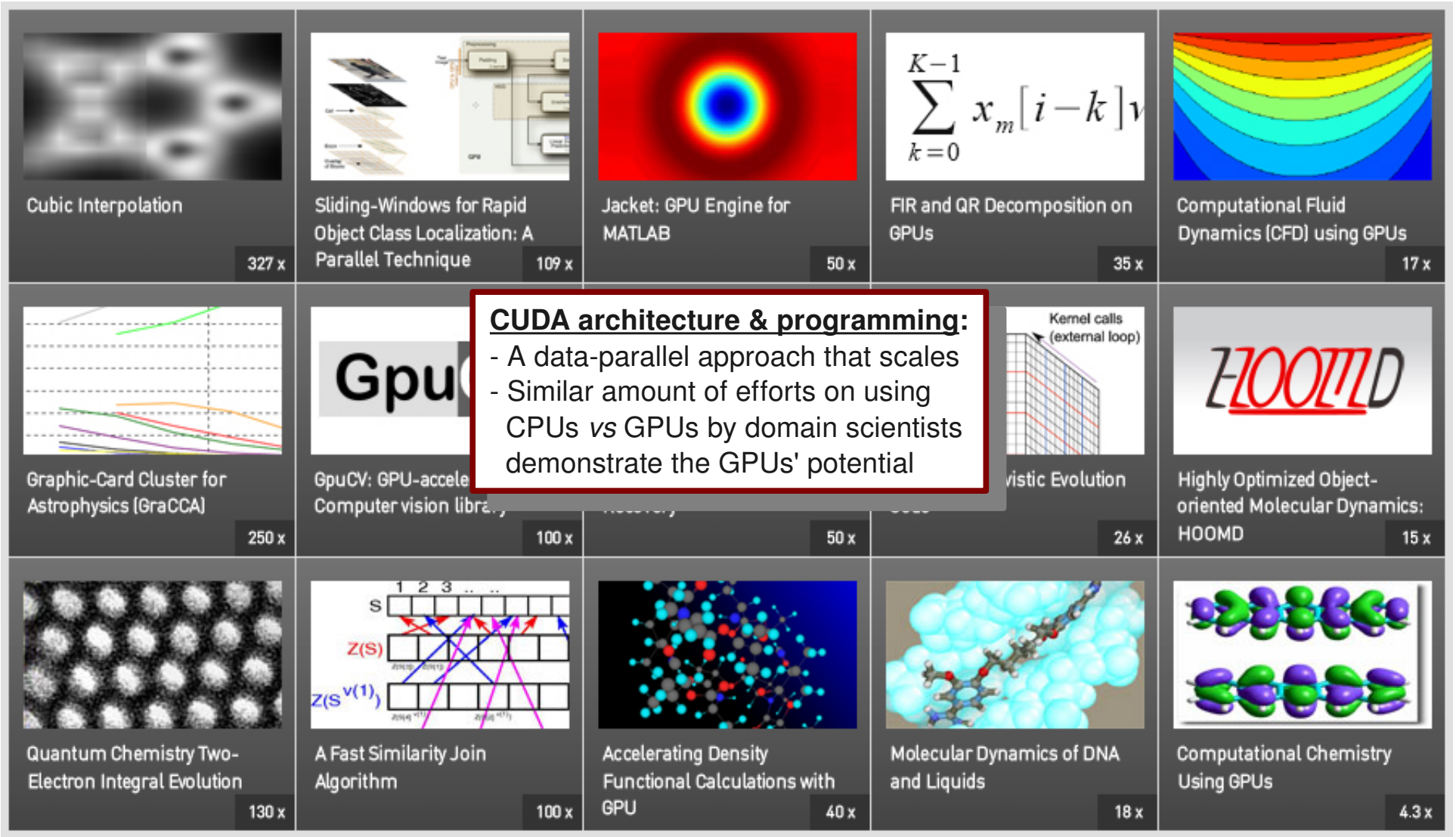
Current NVIDIA GPUs

	GeForce GTX 280	GeForce GTX 260	Tesla C1060	Tesla S1070
Form Factor	Dual slot card	Dual slot card	Dual slot card	Rackmount
TPCs	10	8	10	4x10
SMs	30	24	30	4x30
SPs	240	192	240	4x240
Graphics Freq.	602MHz	576MHz		
Processor Freq.	1296MHz	1242MHz	1300MHz	1500MHz
Memory Freq.	1107MHz	999MHz	800MHz	800MHz
Memory Bandwidth	141.7GB/s	127.9GB/s	102.4GB/s	4x102.4GB/s
Memory Capacity	1GB	896MB	4GB	4x4GB
Power	236W TDP	183W TDP	160W "Typical"	700W "Typical"
SP GFLOP/s (wo/MUL)	622.1	476.9	624.0	4x720.0
SP GFLOP/s (w/MUL)	933.1	715.4	936.0	4x1080.0
DP GFLOP/s	77.8	59.6	78.0	4x72.0

(Source: "*NVIDIA's GT200: Inside a Parallel Processor*")

HPC applications and **high speedups** reported using GPUs

(from NVIDIA CUDA Zone homepage)



CUDA architecture & programming:

- A data-parallel approach that scales
- Similar amount of efforts on using CPUs *vs* GPUs by domain scientists demonstrate the GPUs' potential

Matrix **A**lgebra on **G**PU and **M**ulticore **A**rchitectures (**MAGMA**)

- **MAGMA**: **a new generation linear algebra (LA) libraries** to achieve the fastest possible time to an accurate solution **on hybrid/heterogeneous architectures**, starting with current multicore+MultiGPU systems
  Homepage: **http://icl.cs.utk.edu/magma/**

- **MAGMA** & **LAPACK**
  - **MAGMA** - based on LAPACK and extended for hybrid systems (multi-GPUs + multicore systems);
  - **MAGMA** - designed to be similar to LAPACK in functionality, data storage and interface, in order to allow scientists to effortlessly port any of their LAPACK-relying software components to take advantage of the new architectures
  - **MAGMA** - to leverage years of experience in developing open source LA software packages and systems like LAPACK, ScaLAPACK, BLAS, ATLAS as well as the newest LA developments (e.g. communication avoiding algorithms) and experiences on homogeneous multicores (e.g. PLASMA)

- **Support**
  - NSF, Microsoft, NVIDIA [now **CUDA Center of Excellence at UTK** on the development of **Linear Algebra Libraries for CUDA-based Hybrid Architectures**]

- **MAGMA developers**
  - University of Tennessee, **Knoxville**; University of California, **Berkeley**; University of Colorado, **Denver**

Challenges of using GPUs

- **Massive parallelism**

  Many GPU cores, serial kernel execution

  [e.g. 240 in the GTX280; up to 512 in ***Fermi*** – to have concurrent kernel execution]

- **Hybrid/heterogeneous architectures**

  Match algorithmic requirements to architectural strengths

  [e.g. small, non-parallelizable tasks to run on CPU, large and parallelizable on GPU]

- **Compute *vs* communication gap**

  Exponentially growing gap; persistent challenge

  [Processor speed improves 59%, memory bandwidth 23%, latency 5.5%]

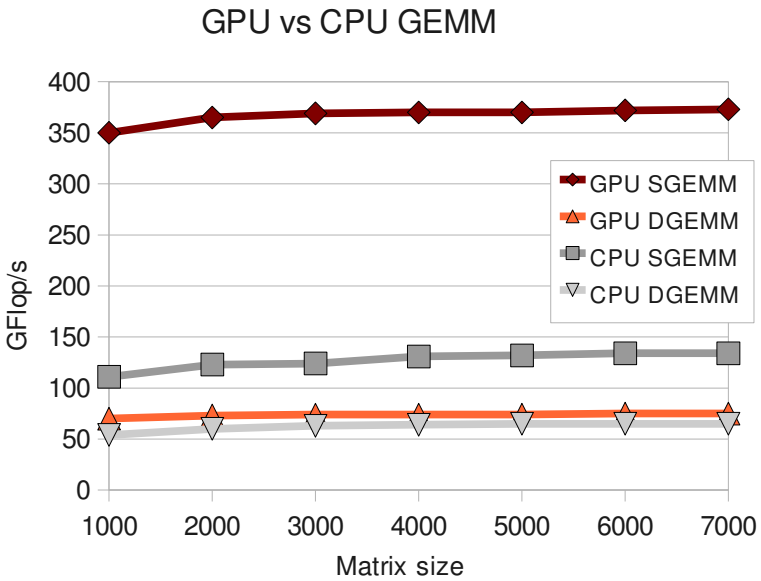
  [on all levels, e.g. a GPU Tesla C1070 (4 x C1060) has compute power of O(1,000) Gflop/s but GPUs communicate through the CPU using O(1) GB/s connection]

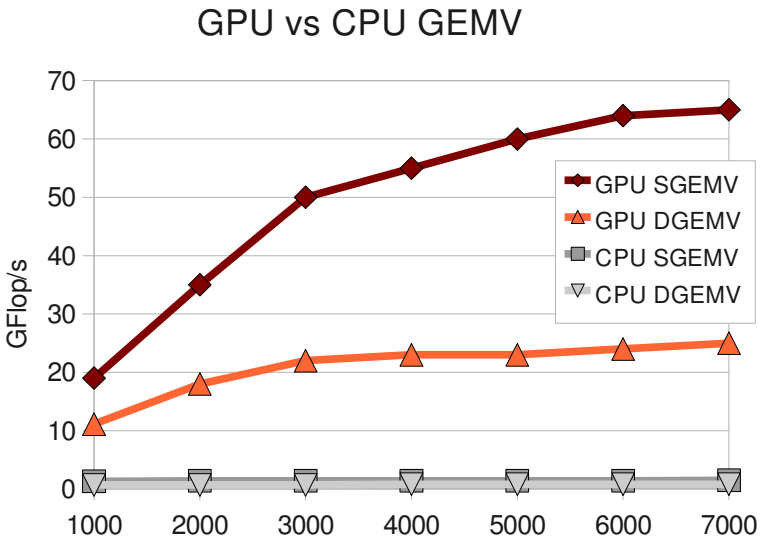
GPUs for HPC

- Programmability
- Performance
- Hybrid computing

How to Code for GPUs?

- Complex question
  - Language, programming model, user productivity, etc
- Recommendations
  - **Use CUDA / OpenCL**
    [already demonstrated benefits in many areas;
    data-based parallelism; move to support task-based]
  - **Use GPU BLAS**
    [high level; available after introduction of **shared memory** –
    can do data reuse; leverage existing developments]
  - **Use Hybrid Algorithms**
    [currently GPUs – massive parallelism but serial kernel execution;
    hybrid approach – small non-parallelizable tasks on the CPU,
    large parallelizable tasks on the GPU]





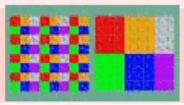
Typical order of acceleration:

dense matrix-matrix	**O(1) X**
dense matrix-vector	**O(10) X**
sparse matrix-vector	**O(100) X**

LAPACK to Multicore

A New Generation of Software:

Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing
PLASMA (00's) New Algorithms (many-core friendly)		Rely on - a DAG/scheduler - block data layout - some extra kernels

Those new algorithms
- have a very low granularity, they scale very well (multicore, petascale computing, …)
- removes a lots of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels

Those new algorithms need new kernels and rely on efficient scheduling algorithms.

"**delayed update**" to organize successive Level 2 BLAS as a single Level 3 BLAS

Split BLAS into tasks and represent algorithms as DAGs; new algorithms where Level 2 BLAS panel factorizations are now Level 3 BLAS

LAPACK to multicore+GPU (**MAGMA**)

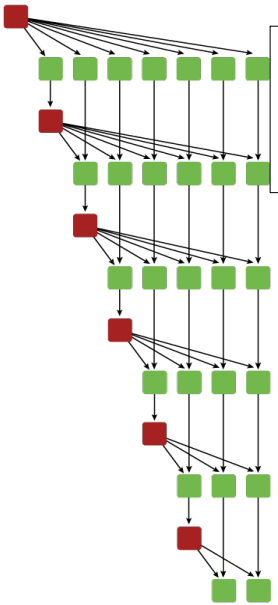
1) Development of **NEW LGORITHMS** **(parallelism, hybrid, optimal communication)**

2) **HYBRIDIZATION** of LAPACK/new algorithms

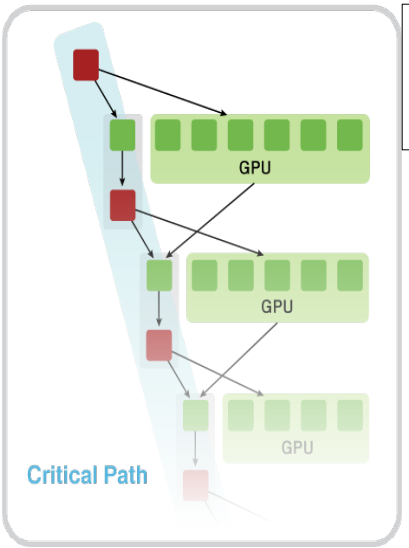
- Represent LAPACK/new algorithms as a collection of **TASKS** and **DEPENDANCIES** among them
- Properly **SCHEDULE** the tasks' execution over the multicore and the GPU

3) Development of GPU BLAS **KERNELS**

4) **AUTO-TUNED** implementations



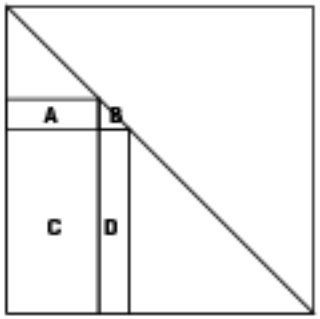
Algorithms as DAGs
(small tasks/tiles for homogeneous **multicore**)



Hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)

One-Sided Dense Matrix Factorizations (LU, QR, and Cholesky)

- Panels (Level 2 BLAS) are factored on CPU using LAPACK
- Trailing matrix updates (Level 3 BLAS) are done on the GPU using “look-ahead” (to overlap CPUs work on the critical path with the GPUs large updates)



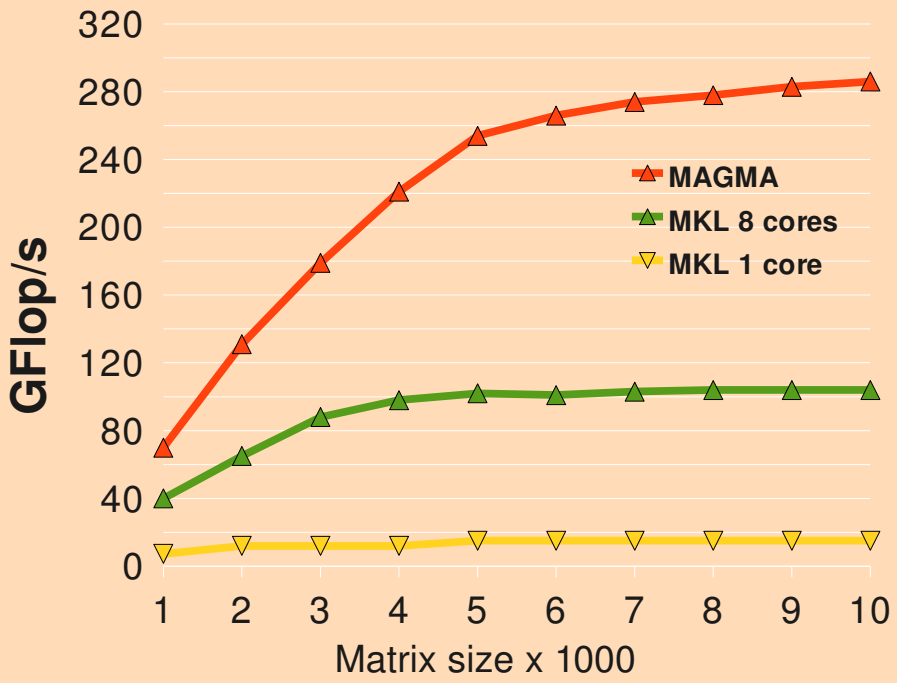
Example: Left-Looking Hybrid Cholesky factorization

MATLAB code	LAPACK code	Hybrid code
(1) B = B – A*A'	ssyrk_("L", "N", &nb, &j, &mone, hA[j,0], ...]	cublasSsyrk['L', 'N', nb, j. mone, dA[j,0], ...]
		cublasGetMatrix(nb, nb, 4, dA[j, j], *lda, hwork, nb]
(2) B = chol(B, 'lower')	spotrf_("L", &nb, hA[j, j], lda, info]	cublasSgemm['N', 'T', j, ...]
(3) D = D – C*A'	sgemm_("N", "T", &j, ...]	spotrf_("L", &nb, hwork, &nb, info]
		cublasSetMatrix(nb, nb, 4, hwork, nb, dA[j, j], *lda]
(4) D = D\B	strsm_("R", "L", "T", "N", &j, ...]	cublasStrsm['R', 'L', 'T', 'N', j, ...]

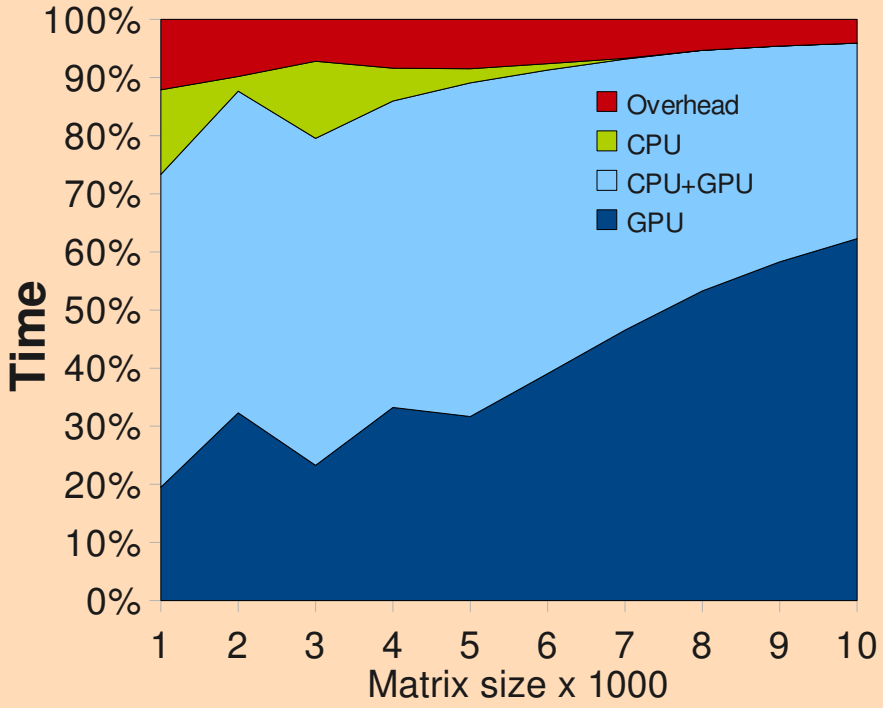
One-sided hybrid factorizations

QR factorization in single precision arithmetic, CPU interface

Performance of MAGMA *vs* MKL



MAGMA QR time breakdown



GPU : NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)
CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz)

GPU BLAS : CUBLAS 2.2, sgemm peak: 375 GFlop/s
CPU BLAS : MKL 10.0 , sgemm peak: 128 GFlop/s

[for more performance data, see **http://icl.cs.utk.edu/magma**]

Two-sided matrix factorizations

- Two-sided factorization, e.g. consider Hessenberg reduction

  $Q\,A\,Q' = H$

  H – upper Hessenberg, Q – orthogonal similarity transformation

- **Importance**

One-sided factorizations	**Two-sided factorizations**
- bases for linear solvers	- bases for eigen-solvers

- Block algorithm

  Q – a product of n-1 elementary reflectors

  $Q = H_1\,H_2 \ldots H_{n-1}, \qquad H_i = I - \tau_i\,v_i\,v_i'$

  $H_1 \ldots H_{nb} = I - V\,T\,V'$ (*WY transform*; the bases for delayed update or block algorithm)

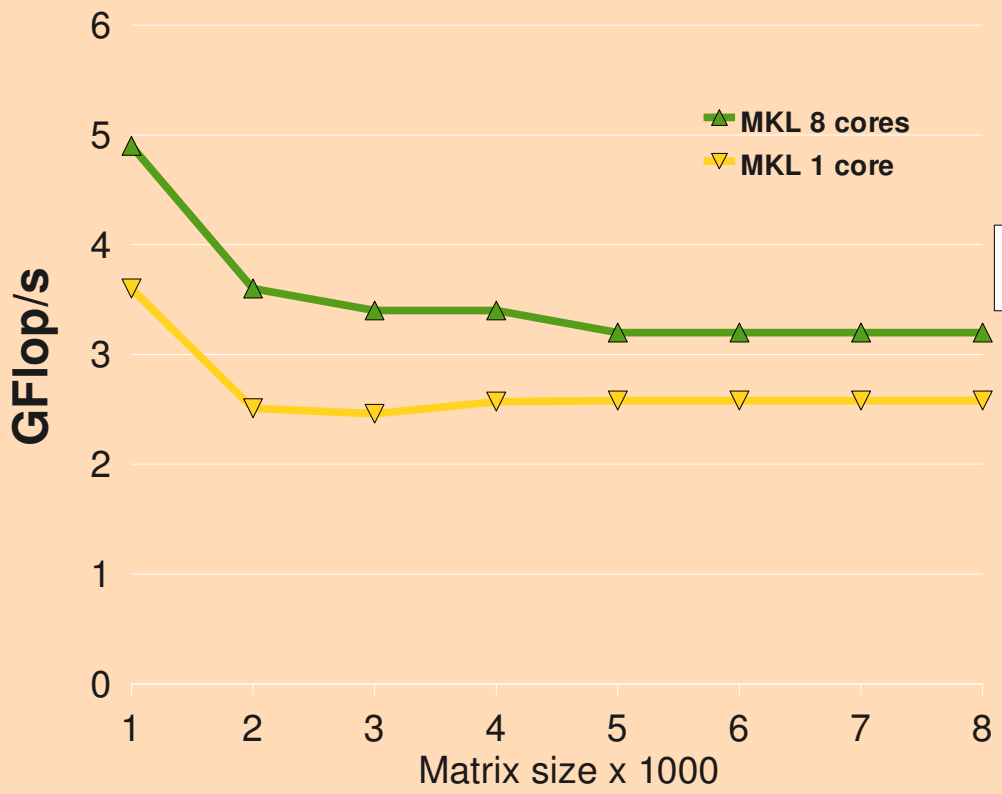
- **Can we accelerate it ?**

  [similarly to the one-sided using hybrid GPU-based computing]

  [to see **much higher acceleration** due to a removed bottleneck]

Homogeneous multicore acceleration?

Hessenberg factorization in double precision arithmetic, CPU interface
Performance of MAGMA *vs* MKL

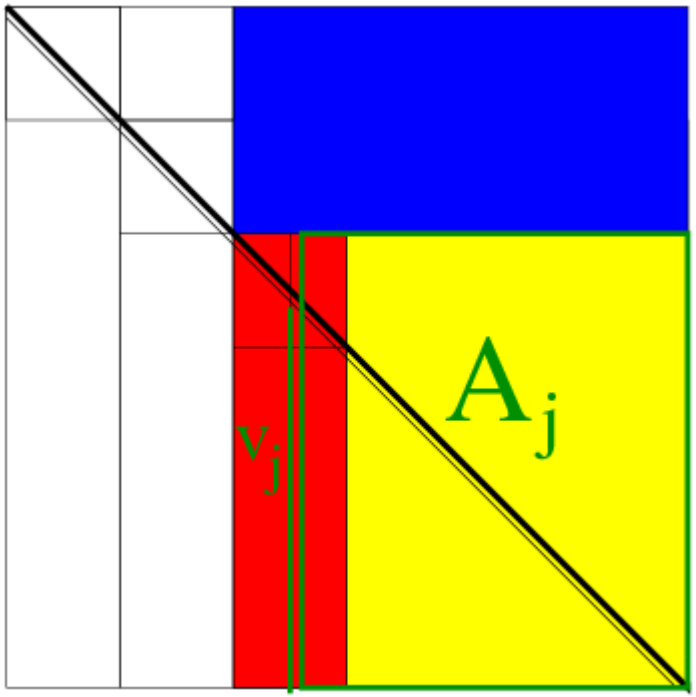


One core enough to saturate the bus
Performance does not scale

CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz) **CPU BLAS** : MKL 10.0 , dgemm peak: 65 GFlop/s

- There have been difficulties in accelerating it on homogeneous multicores

The Bottleneck



- Level 3 BLAS update
  [80% flops; ~30% of the run time]
- Level 2 BLAS update
  [20% flops; ~70% of the run time]
- $y_j = A_j v_j$
  [Line 5 of Algorithm 2]

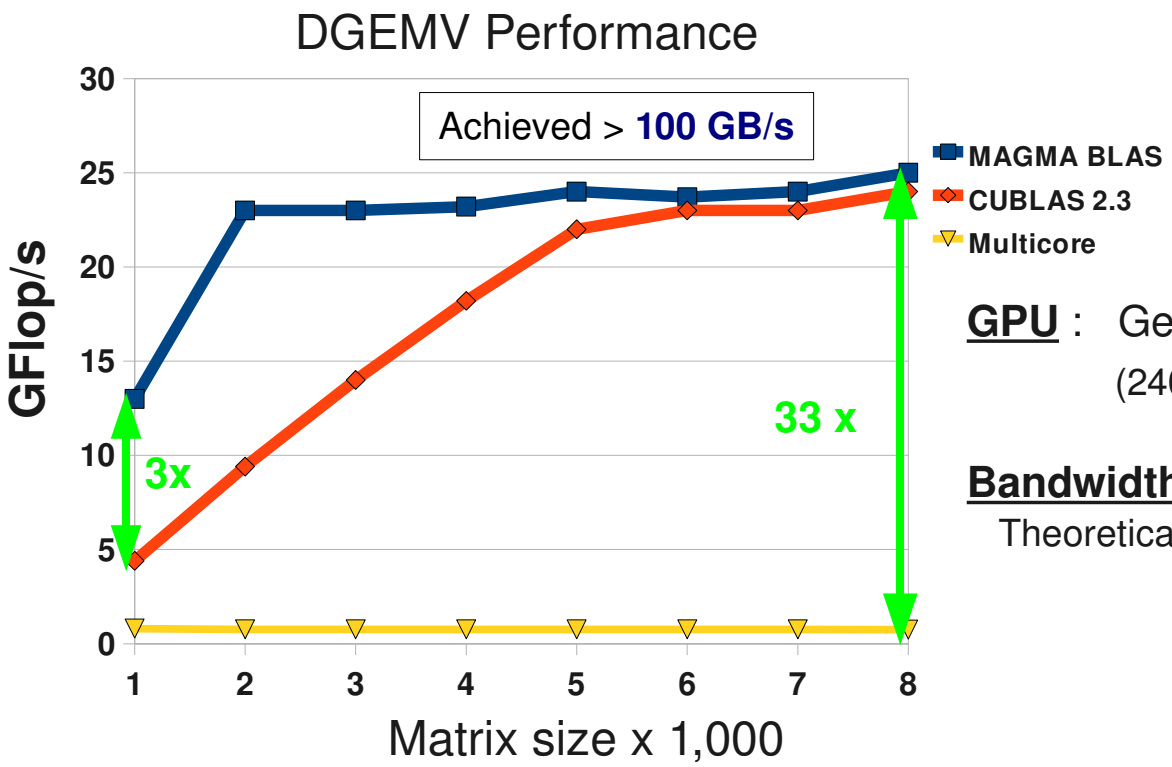
Reduction times in seconds for N = 4,000

# cores	1	8	1+GPU	8+GPU
Level 3 BLAS	**25** (30%) /	**4**	**3.5** (60%) /	**2.7**
Level 2 BLAS	**59** (70%) /	**59**	**2.3** (40%) /	**2.3**

No improvement

Hybrid computing acceleration?

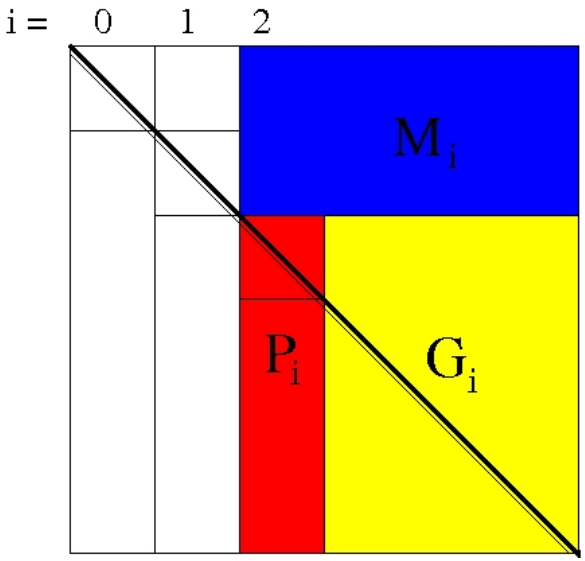
- Intuitively, yes, as **matrix-vector product is fast on GPUs**
- How to organize a hybrid computation ?

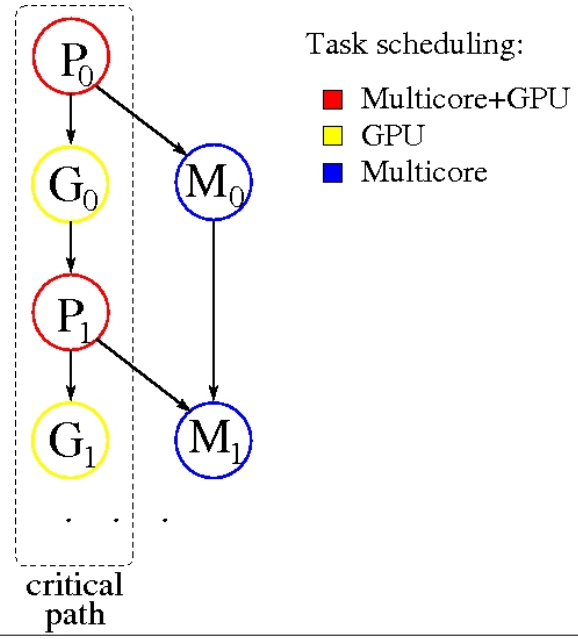


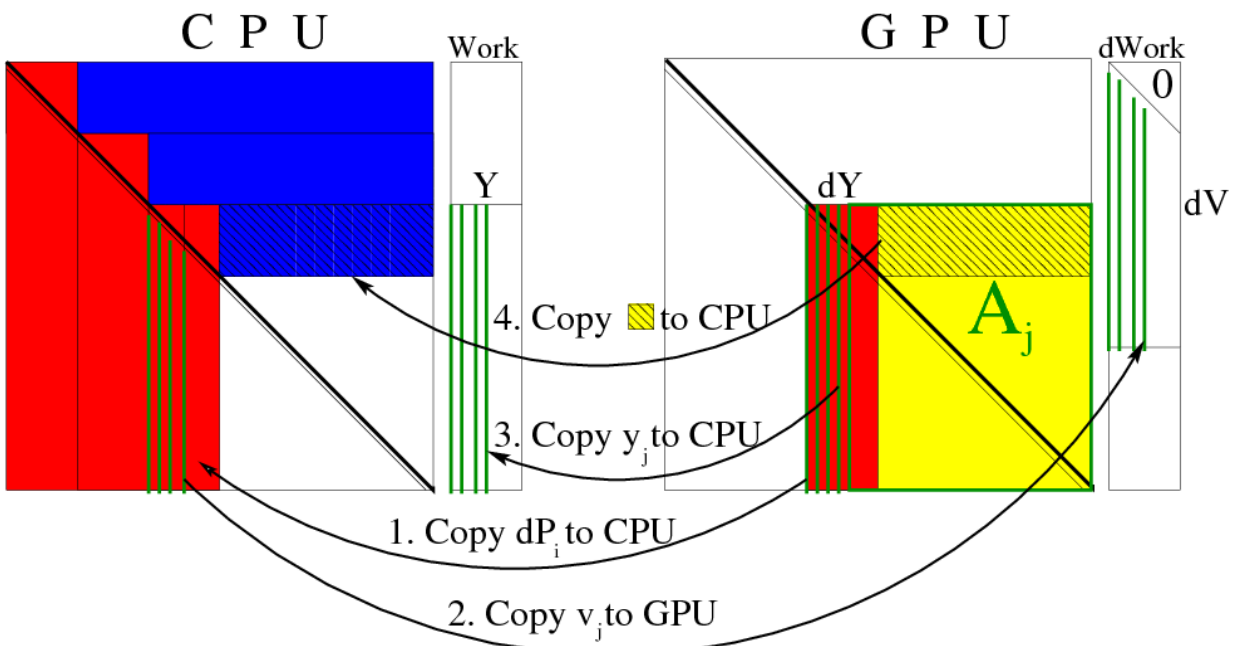
GPU : GeForce GTX 280
(240 Cores @ 1.30 GHz)

Bandwidth:
Theoretical : 141 GB/s

Task **Splitting** & Task **Scheduling**

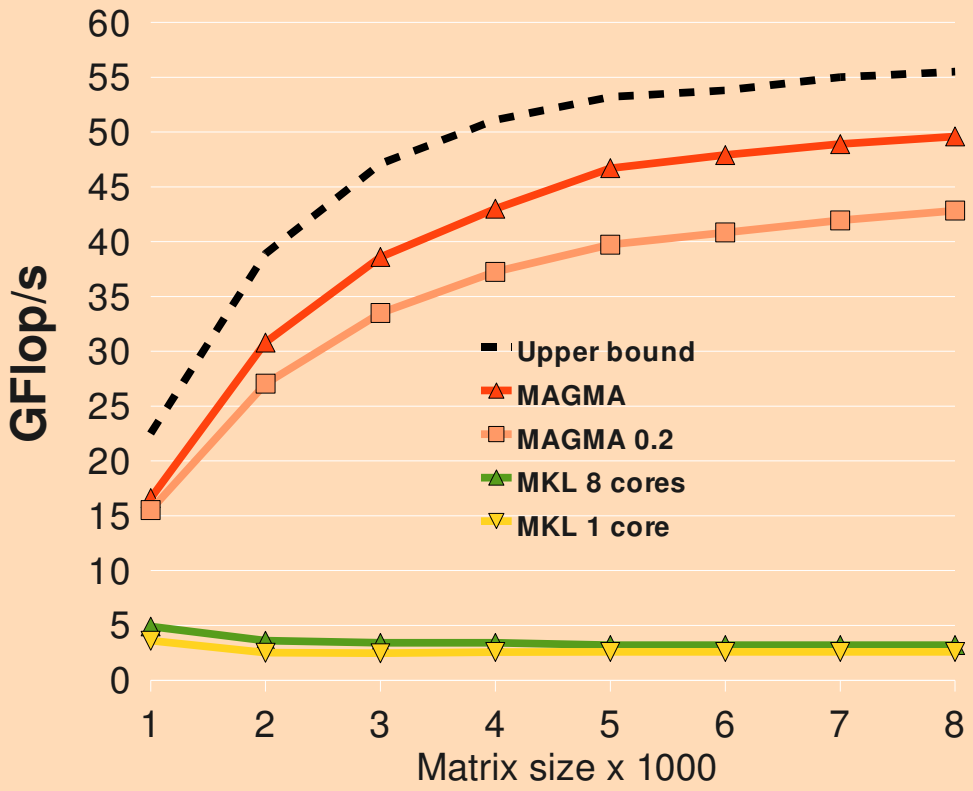






Performance

Hessenberg factorization in double precision arithmetic, CPU interface
Performance of MAGMA *vs* MKL



Acceleration is much higher than one-sided (**O(10) x**) due to removed bottleneck

GPU : NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)
CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz)

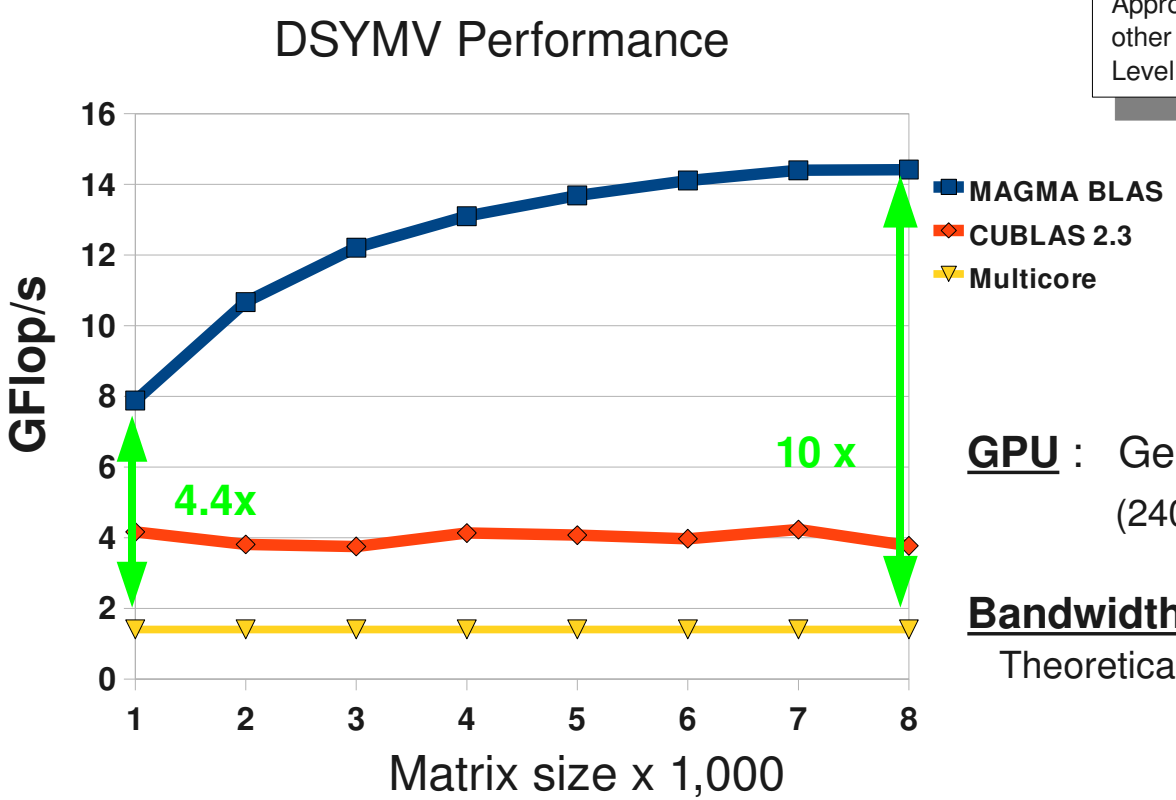
GPU BLAS : CUBLAS 2.3, dgemm peak: 75 GFlop/s
CPU BLAS : MKL 10.0 , dgemm peak: 65 GFlop/s

[for more performance data, see **http://icl.cs.utk.edu/magma**]

The symmetric case

- The same approach, using a fast DSYMV on the GPU

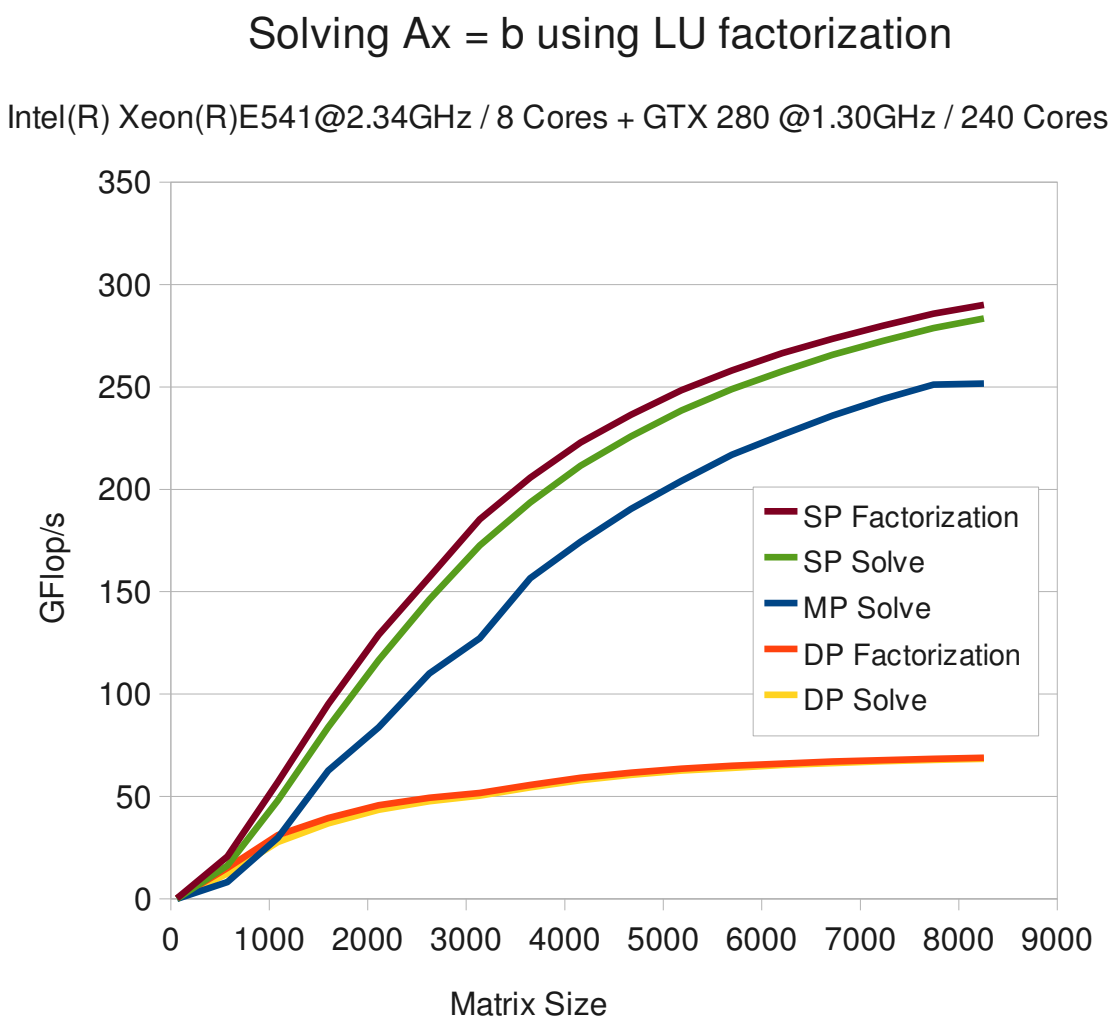
Approach can be applied similarly to other types of matrices as the fast Level 2 BLAS needed is available



GPU : GeForce GTX 280
(240 Cores @ 1.30 GHz)

Bandwidth:
Theoretical : 141 GB/s

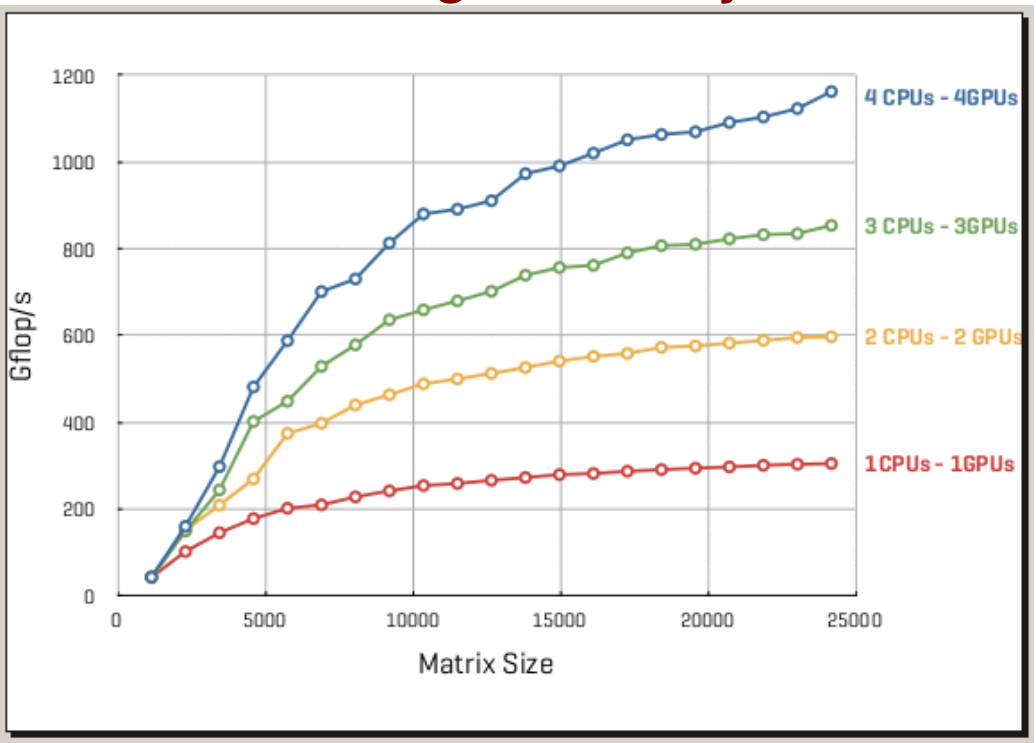
Linear Solvers



- Direct solvers
  - Factor and do triangular solves in the same, working precision
- Mixed Precision Iterative Refinement
  - Factor in single (i.e. the bulk of the computation in fast arithmetic) and use it as preconditioner in simple double precision iteration, e.g. $x_{i+1} = x_i + (LU_{SP})^{-1} P (b - A x_i)$

Extension to Multicore and Multi GPUs

Cholesky factorization in SP
Strong Scalability



HOST: 4x AMD Opteron core @1.8GHz
GPUs: 4x C1060 (240 cores each @1.44GHz)

Leveraging PLASMA developments:
- 2 level nested parallelism
  coarse: PLASMA tiled algorithm and static scheduling
  fine : tasks/tiles are redefined for hybrid 1 core+GPU computing
- Defining a "***Magnum tiles approach***"

[with Hatem Ltaief, Rajib Nath, Peng Du, and Jack Dongarra]

Conclusions

- **Hybrid Multicore+GPU computing:**
  - **Architecture trends:**
    towards heterogeneous/hybrid designs
  - **Can significantly accelerate DLA** [*vs* just multicores] **;**
    - for both one and two-sided factorizations
  - **Can significantly accelerate algorithms that are slow on homogeneous architectures**