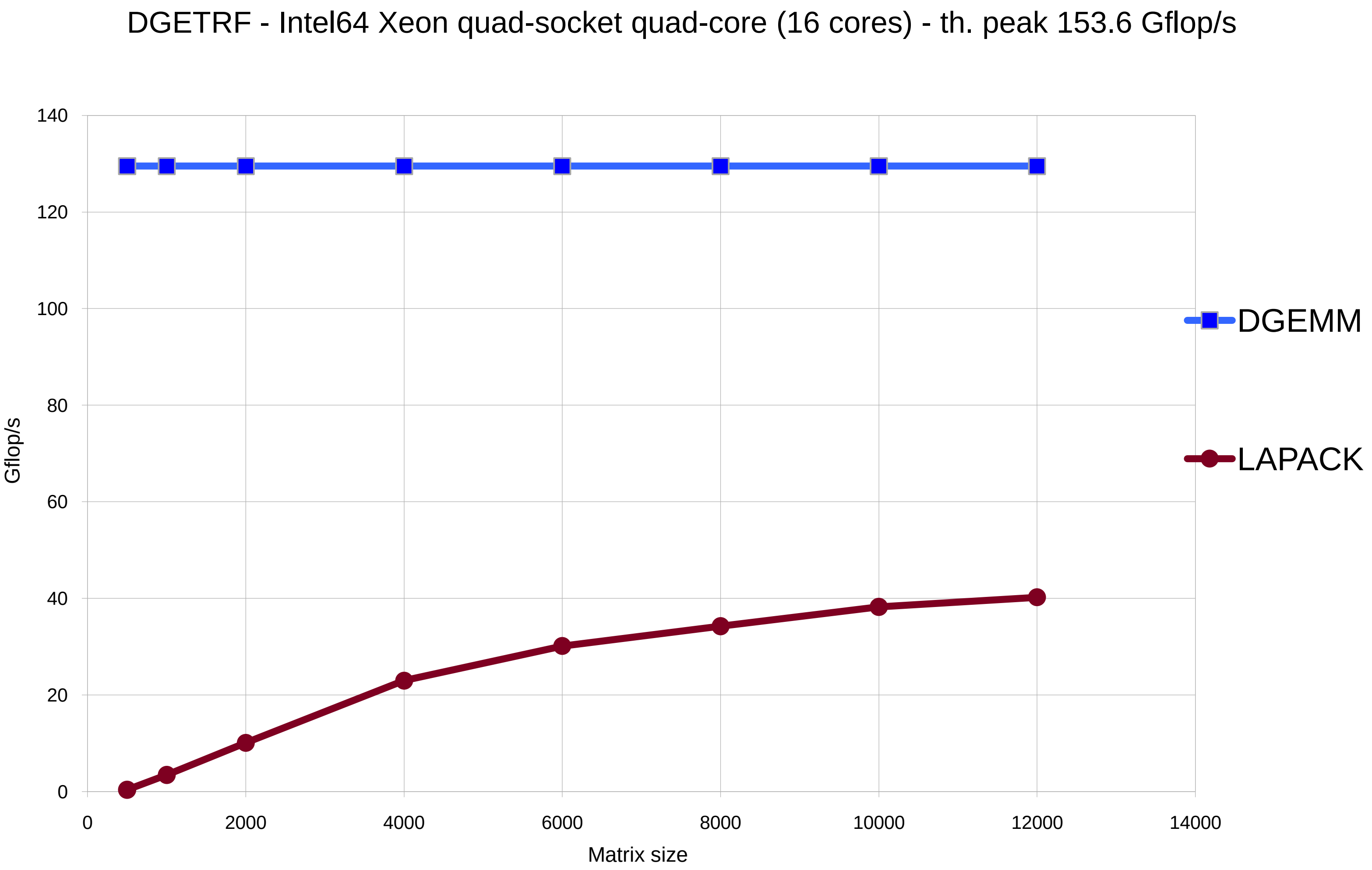
A Linear Algebra Library for Multicore/Accelerators: the PLASMA/MAGMA Collection

Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory

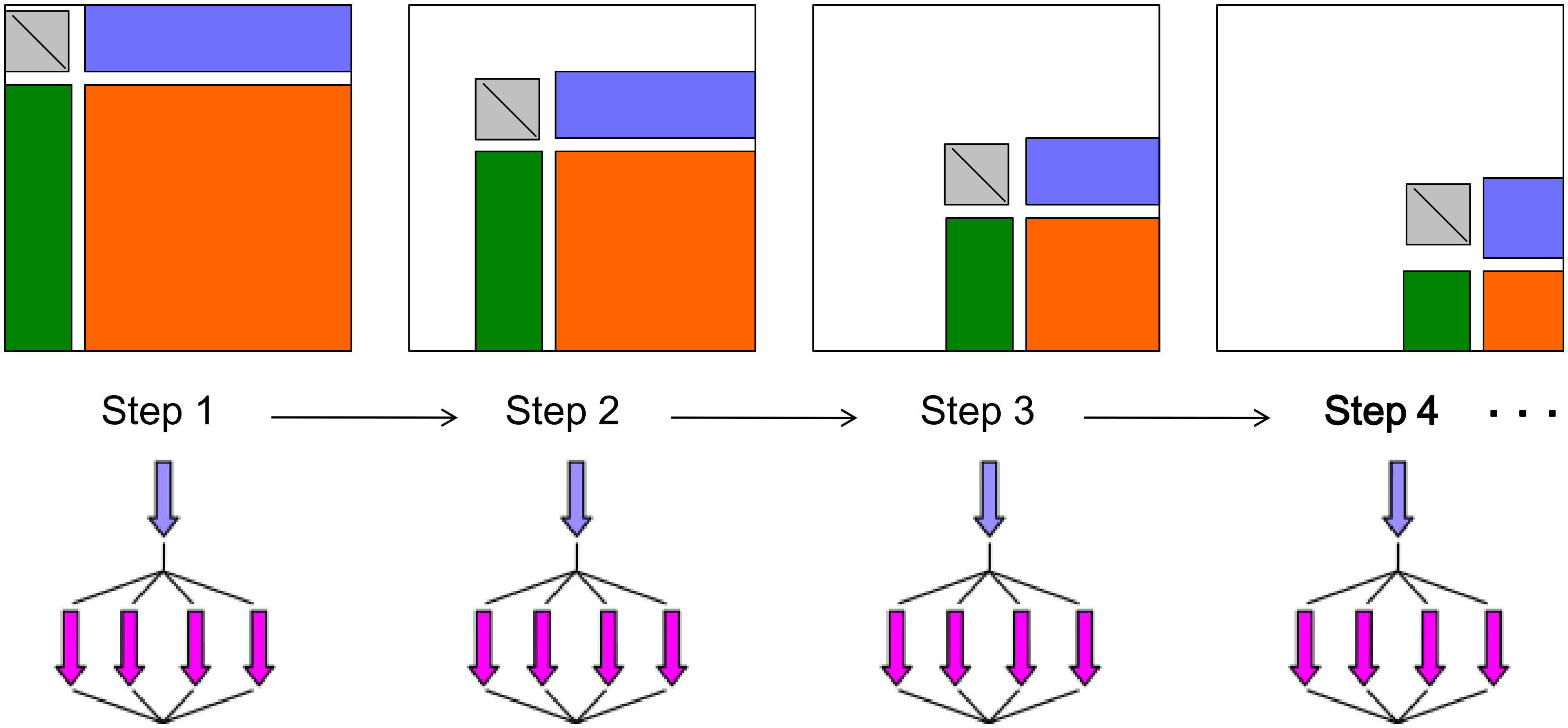
LAPACK LU - Intel64 - 16 cores



Major Changes to Software

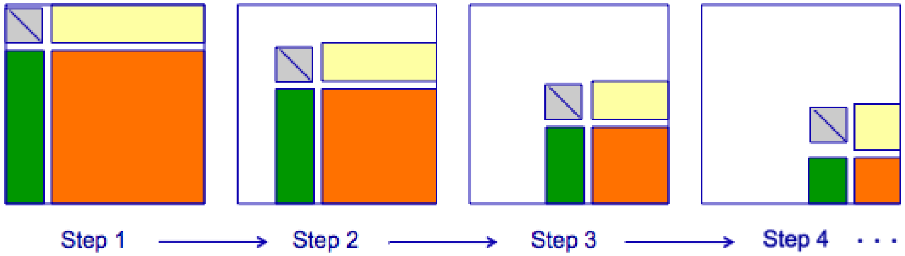
- Must rethink the design of our software
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software
- Numerical libraries for example will change
  - For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this

LAPACK LU

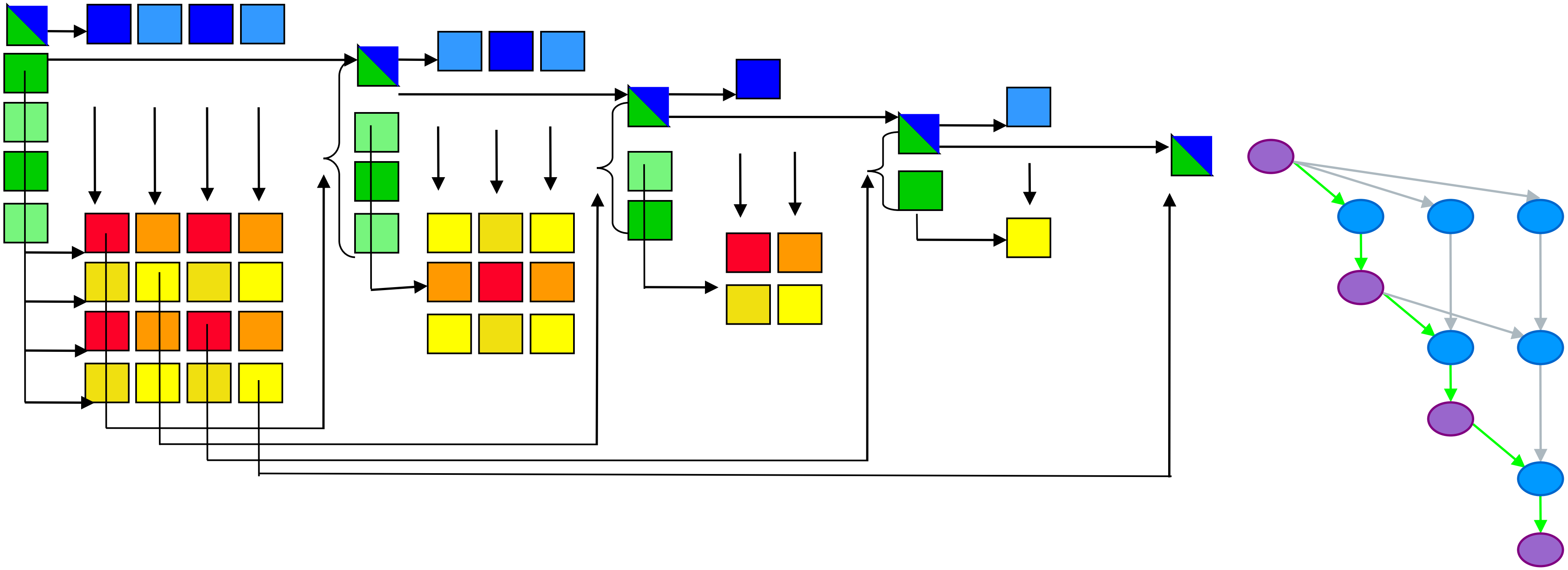


- **Fork-join, bulk synchronous processing**

Parallel Tasks in LU



- Break into smaller tasks and remove dependencies



PLASMA (Redesign LAPACK/ScaLAPACK)

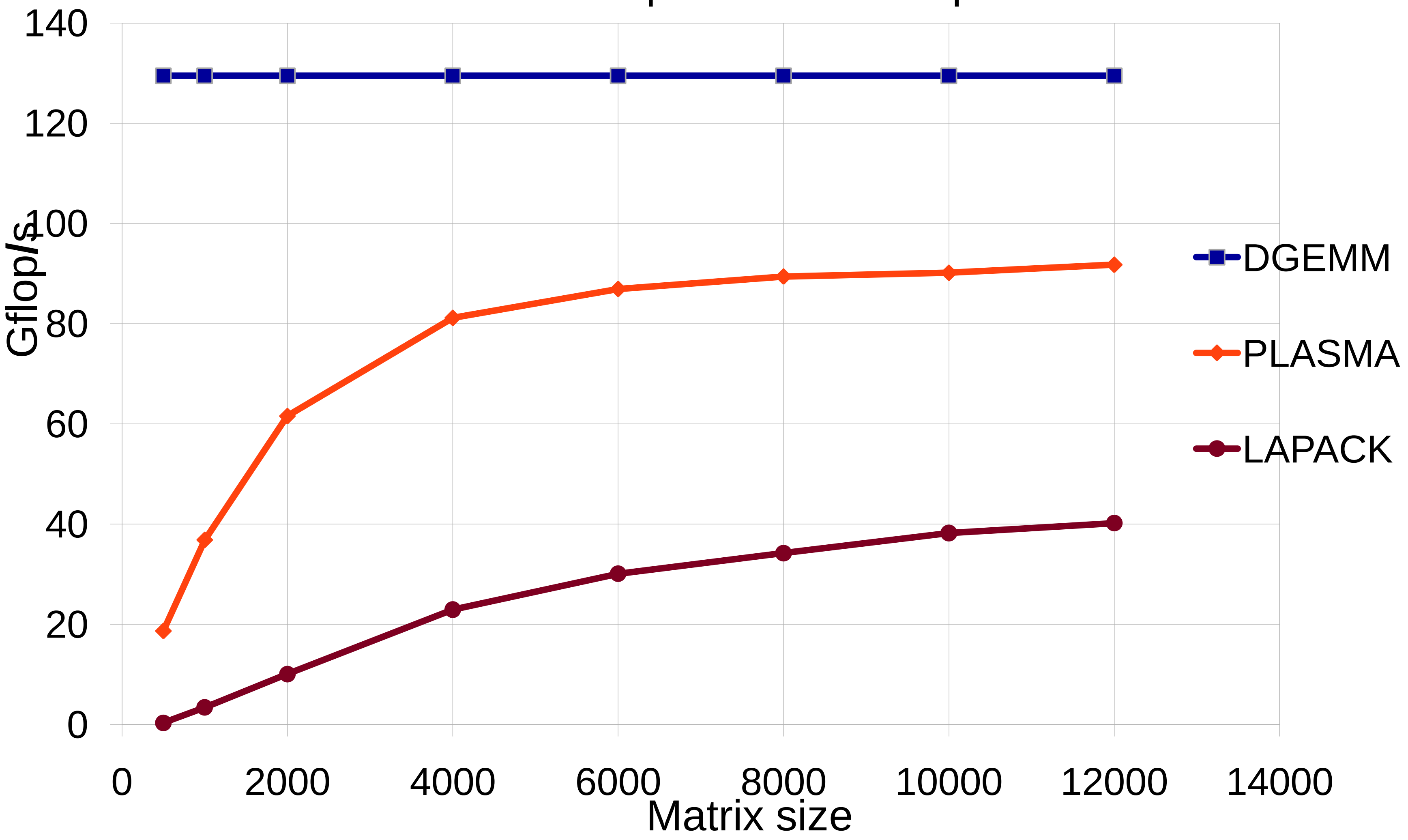
Parallel Linear Algebra Software for Multicore Architectures

- Asychronicity
  - Avoid fork-join (Bulk sync design)
- Dynamic Scheduling
  - Out of order execution
- Fine Granularity
  - Independent block operations
- Locality of Reference
  - Data storage – Block Data Layout

Lead by Tennessee and Berkeley similar to LAPACK/ScaLAPACK as a community effort

LU - Intel64 - 16 cores

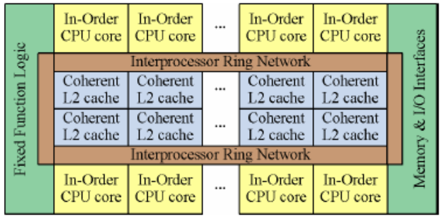
DGETRF - Intel64 Xeon quad-socket quad-core (16 cores)
theoretical peak 153.6 Gflop/s



Future Computer Systems

- Most likely be a hybrid design
- Think standard multicore chips and accelerator (GPUs)

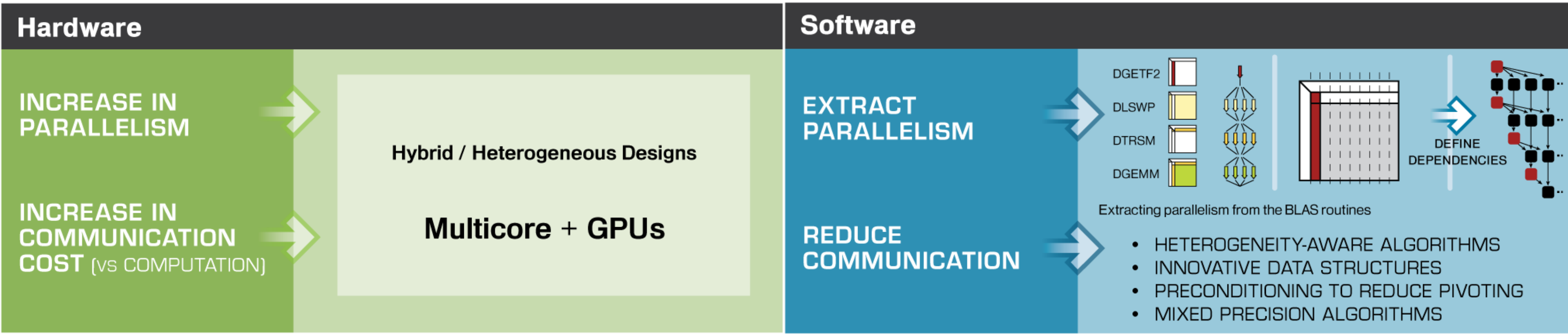
- Today accelerators are attached
- Next generation more integrated
- Intel's Larrabee in 2010
  - 8,16,32,or 64 x86 cores



Intel Larrabee

- AMD's Fusion in 2011
  - Multicore with embedded graphics ATI
- Nvidia's plans?

Hardware to Software Trends



Hardware

INCREASE IN PARALLELISM

INCREASE IN COMMUNICATION COST (VS COMPUTATION)

Hybrid / Heterogeneous Designs

Multicore + GPUs

Software

EXTRACT PARALLELISM

REDUCE COMMUNICATION

Extracting parallelism from the BLAS routines

- HETEROGENEITY-AWARE ALGORITHMS
- INNOVATIVE DATA STRUCTURES
- PRECONDITIONING TO REDUCE PIVOTING
- MIXED PRECISION ALGORITHMS

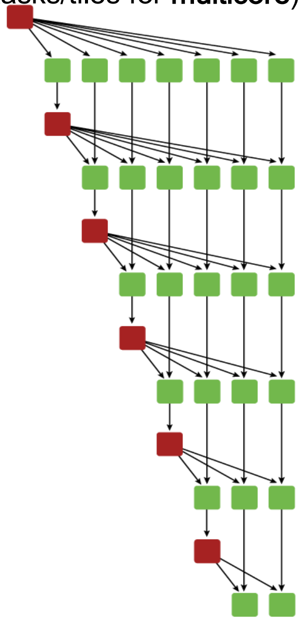
- The **MAGMA** Project [Matrix Algebra on GPU and Multicore Architectures]

  - create LA libraries for the next-generation of **highly parallel, and heterogeneous processors**
  - to be similar to LAPACK in **functionality**, **data storage**, and **interface**
  - to allow **effortless port of LAPACK-relying software** to take advantage of the new hardware
  - to run on homogeneous x86-based multicores and take advantage of GPUs (if available)

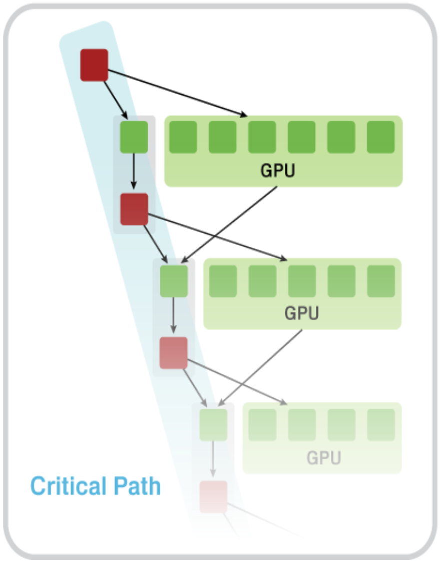
Hybrid Computing

- **Match algorithmic requirements to architectural strengths of the hybrid components**
  Multicore : small tasks/tiles
  Accelerator: large data parallel tasks

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



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

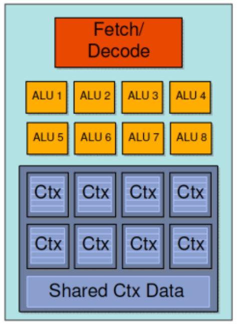


- **e.g. split the computation into tasks; define critical path that "clears" the way for other large data parallel tasks; proper schedule the tasks execution**
- **Design algorithms with well defined "*search space*" to facilitate auto-tuning**

NVIDIA GeForce GTX 280

- NVIDIA-Speak
  - 240 stream processors
- Generic speak
  - 30 processing cores
    - 8 SIMD functional units per core
  - 1 mul-add (2 flops) + 1 mul per functional unit (3 flops/cycle)
  - Best case theoretically: 240 mul-adds + 240 muls per cycle
    - 1.3 GHz clock
    - 30 * 8 * (2 + 1) * 1.33 = 933 Gflop/s peak
  - Best case reality: 240 mul-adds per clock
    - Just able to do the mul-add so 2/3 or 624 Gflop/s
  - All this is single precision
    - Double precision is 78 Gflop/s peak (Factor of 8 from SP; exploit mixed prec)
  - 141 GB/s bus, 1 GB memory
  - 4 GB/s via PCIe (we see: T = 11 us + Bytes/3.3 GB/s)
  - In SP SGEMM performance 375 Gflop/s

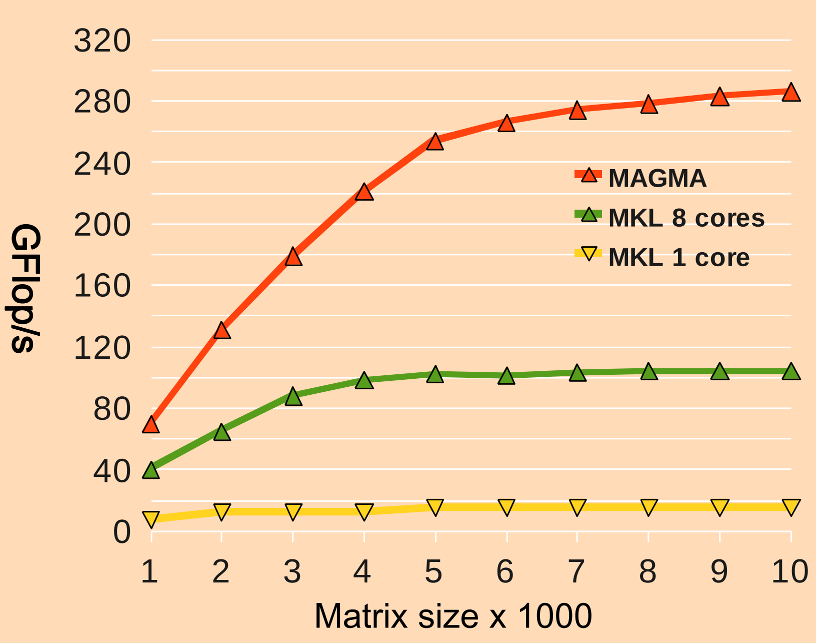
Processing Core



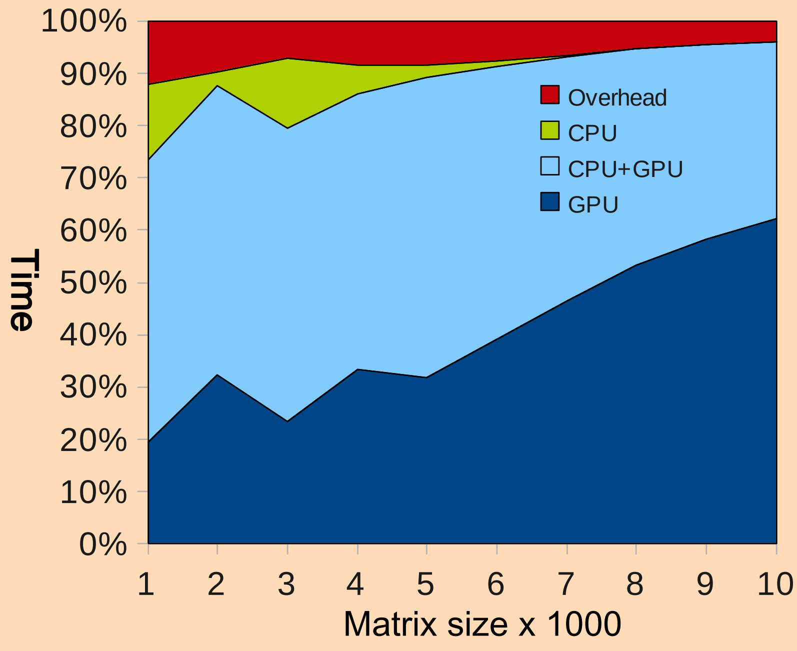
MAGMA version 0.1 performance

QR factorization in single precision arithmetic, CPU interface

Performance of MAGMA *vs* MKL



MAGMA QR time breakdown



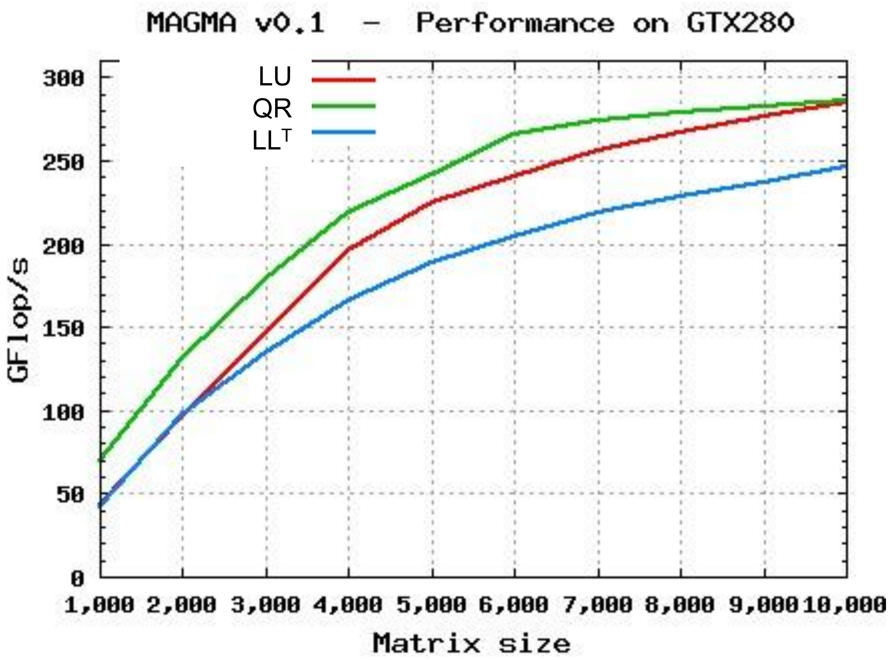
GPU : NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)
CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz)

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

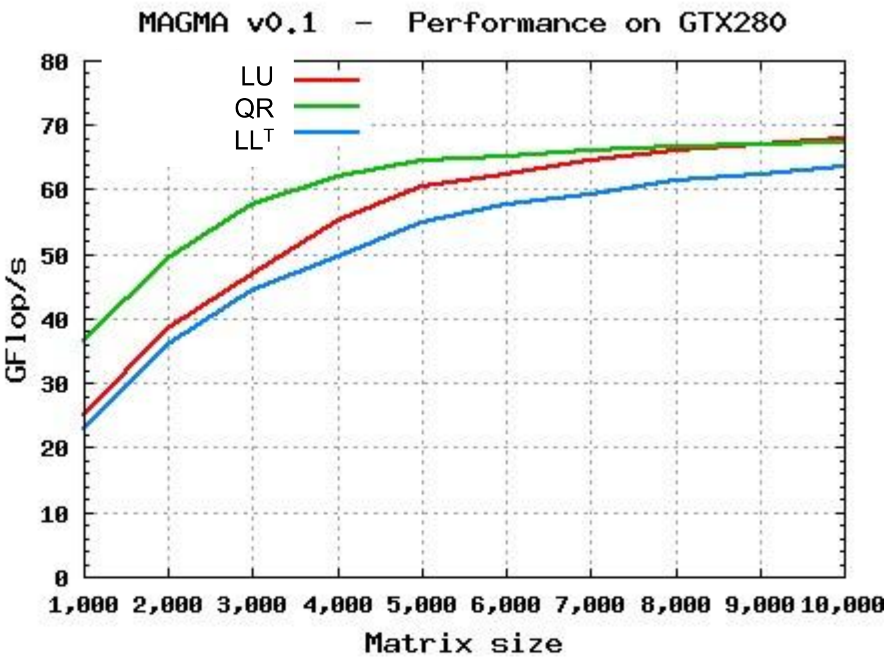
[for more performance data, see http://icl.cs.utk.edu/magma]

Single Core and GTX-280

Single Precision



Double Precision



GPU : NVIDIA GeForce GTX 280
CPU : Intel Xeon dual socket quad-core @2.33 GHz

GPU BLAS : CUBLAS 2.2, s/d gemm peak: 375 / 75 GFlop/s
CPU BLAS : MKL 10.0 , s/d gemm peak: 17.5 / 8.6 GFlop/s

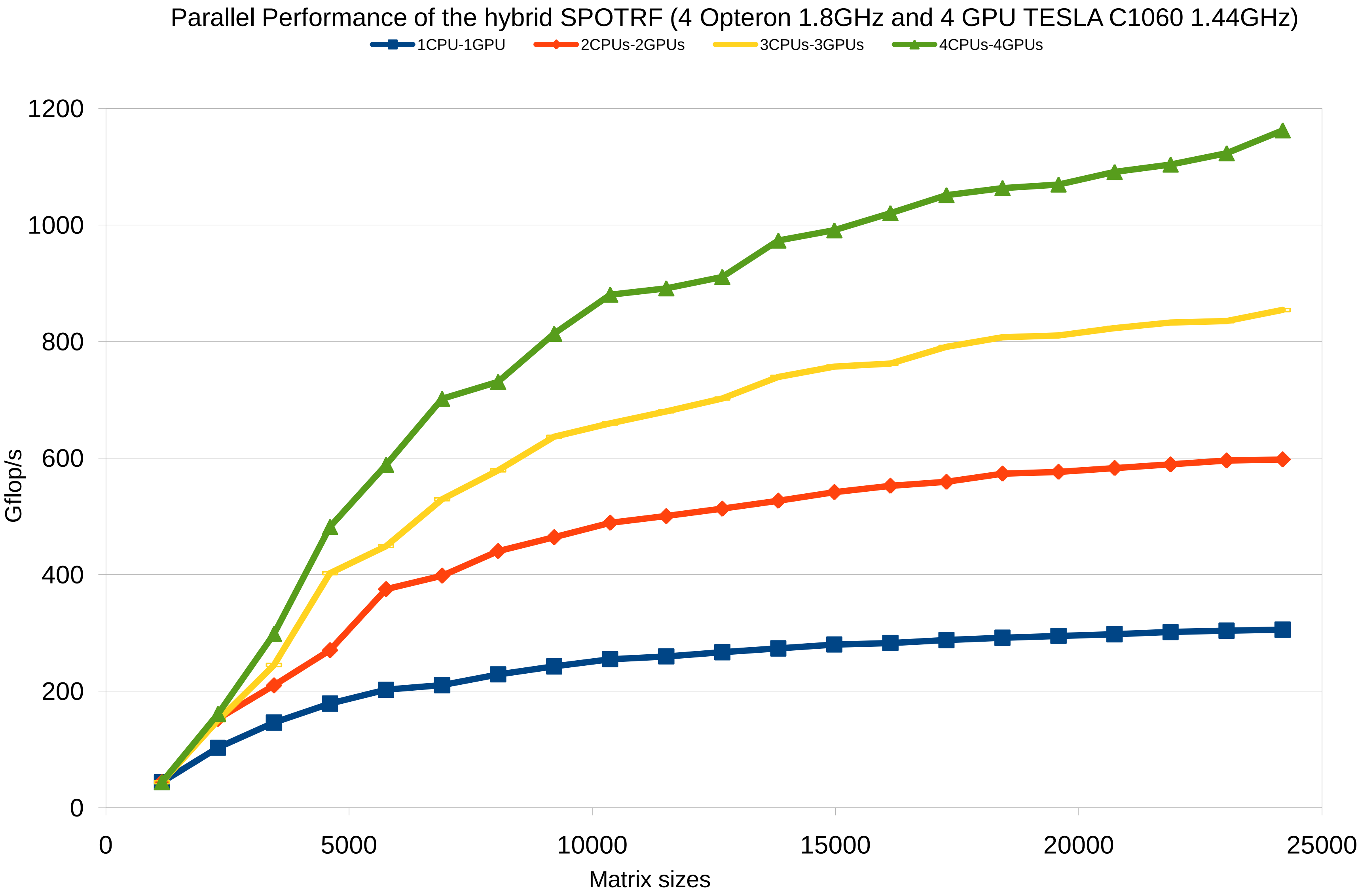
MAGMA Software

- Available through MAGMA's homepage
  http://icl.cs.utk.edu/magma/
- Included are the 3 one-sided matrix factorizations
- Iterative Refinement Algorithm (Mixed Precision)
- Standard (LAPACK) data layout and accuracy
- Two LAPACK-style interfaces
  - CPU interface: both input and output are on the CPU
  - GPU interface: both input and output are on the GPU
- This release is intended for single GPU

Cholesky on multicore + multi-GPUs

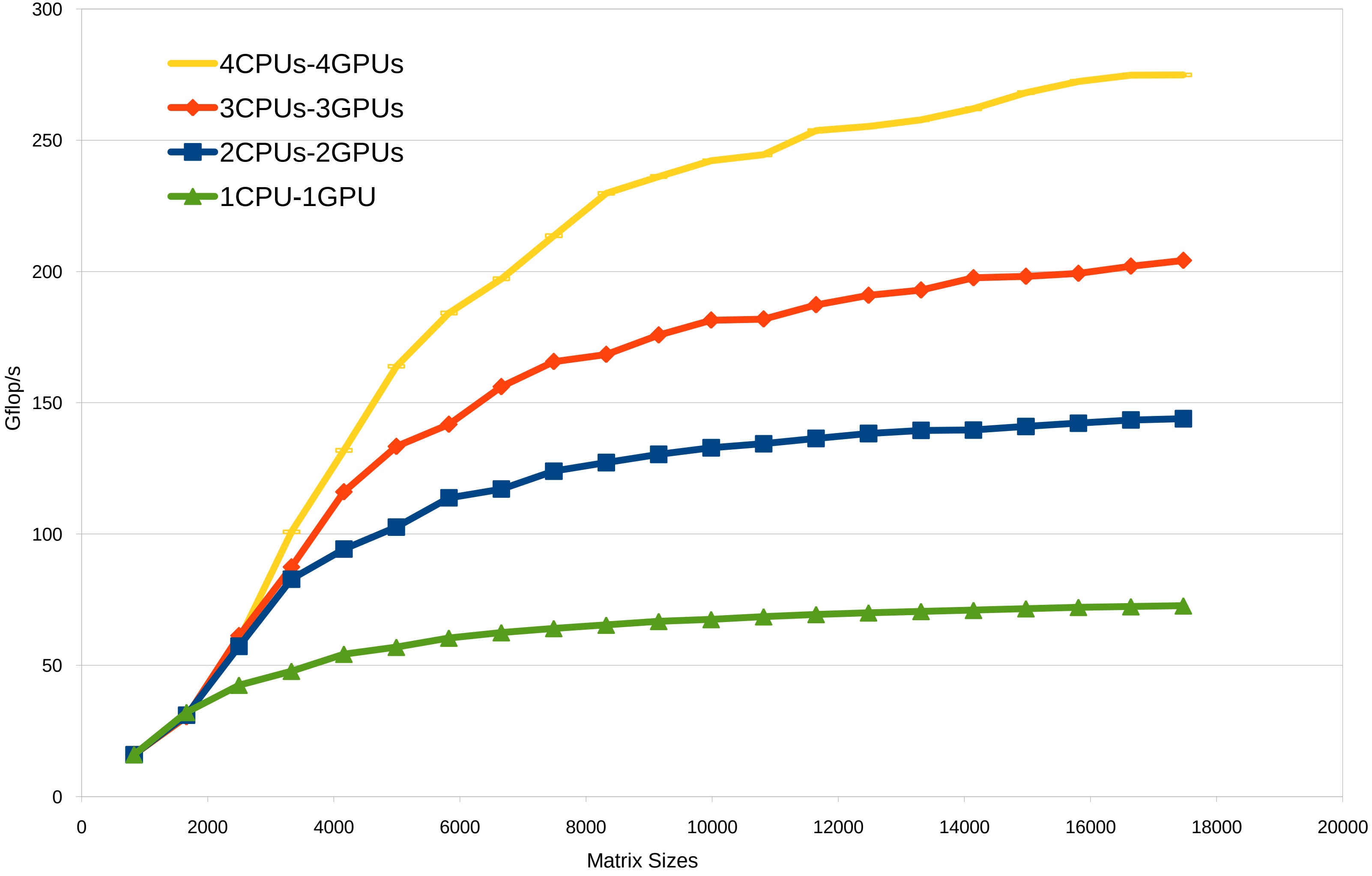
- **Hardware**
  - •HOST: Two socket - dual core AMD Opteron 1.8GHz, 2GB memory
  - •DEVICE:
    - –4 GPU TESLA C1070 1.44GHz
    - –240 computing cores per GPU
    - –4GB memory per GPU
    - –Single precision floating point performance (NVIDIA PEAK): 4.14 Tflop/s
    - –Memory bandwidth: 408 GB/s
    - –System interface: PCIexpress
- **Preliminary results on single precision**
- **RAM limitations on HOST prevented runs on larger sizes**

Single Precision - Cholesky on Multicore + Multi GPUs



DP Cholesky with Multiple GPUs

Parallel Performance of the hybrid DPOTRF (4 Opteron 1.8GHz and 4 GPU TESLA C1060 1.44GHz)



Performance of Single Precision on Conventional and GPU's

- Realized have the similar situation on our commodity processors.
  - That is, SP is 2X as fast as DP on many systems
- The Intel Pentium and AMD Opteron have SSE2
  - 2 flops/cycle DP
  - 4 flops/cycle SP
- IBM PowerPC has AltiVec
  - 8 flops/cycle SP
  - 4 flops/cycle DP
    - No DP on AltiVec

NVIDIA Tesla

Best case reality: 240 mul-adds per clock

Just able to do the mul-add so 2/3 or 624 Gflop/s of theoretical peak

All this is single precision

Double precision is 78 Gflop/s peak (Factor of 8 from SP; exploit mixed prec)

Single precision is faster because:
- Operations are faster
- Reduced data motion
- Larger blocks gives higher locality in cache

Idea Goes Something Like This…

- **Exploit 32 bit floating point as much as possible.**
  - **Especially for the bulk of the computation**
- **Correct or update the solution with selective use of 64 bit floating point to provide a refined results**
- **Intuitively:**
  - **Compute a 32 bit result,**
  - **Calculate a correction to 32 bit result using selected higher precision and,**
  - **Perform the update of the 32 bit results with the correction using high precision.**

Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, *Ax* = *b*, can work this way.

```
L U = lu(A)                                  $O(n^3)$
x = L\(U\b)                                  $O(n^2)$
r = b – Ax                                   $O(n^2)$
WHILE || r || not small enough
        z = L\(U\r)                          $O(n^2)$
        x = x + z                            $O(n^1)$
        r = b – Ax                           $O(n^2)$
END
```

  - Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.

Mixed-Precision Iterative Refinement

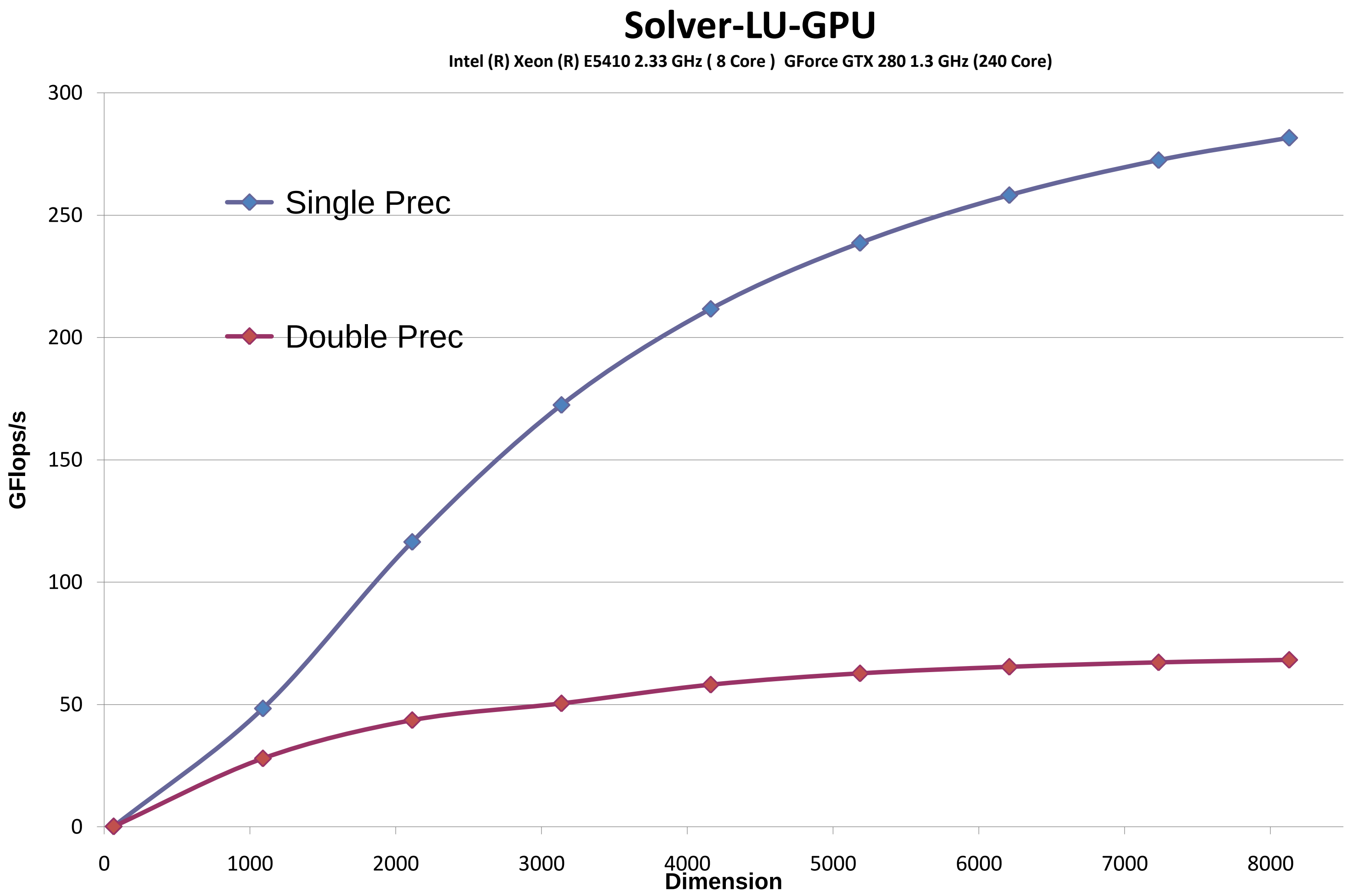
- Iterative refinement for dense systems, *Ax = b*, can work this way.

```
L U = lu(A)                          SINGLE      $O(n^3)$
x = L\(U\b)                          SINGLE      $O(n^2)$
r = b – Ax                           DOUBLE      $O(n^2)$
WHILE || r || not small enough
        z = L\(U\r)                  SINGLE      $O(n^2)$
        x = x + z                    DOUBLE      $O(n^1)$
        r = b – Ax                   DOUBLE      $O(n^2)$
END
```

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

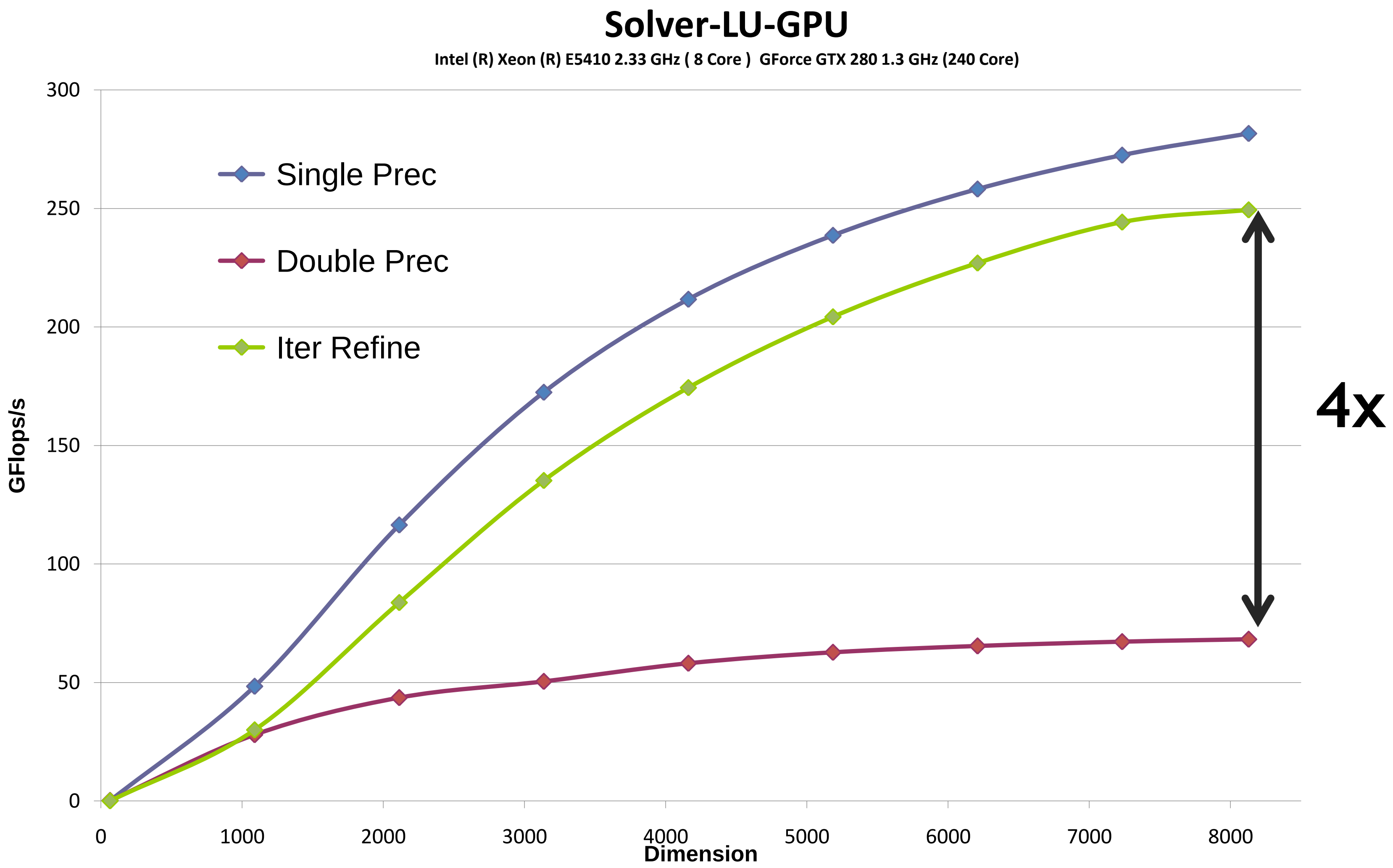
- Requires extra storage, total is 1.5 times normal;
- $O(n^3)$ work is done in lower precision
- $O(n^2)$ work is done in high precision
- Problems if the matrix is ill-conditioned in sp; $O(10^8)$

General (LU) Solver



GPU : NVIDIA GeForce GTX 280 **GPU BLAS** : CUBLAS 2.2, s/d gemm peak: 375 / 75 GFlop/s
CPU : Intel Xeon dual socket quad-core @2.33 GHz **CPU BLAS** : MKL 10.0 , s/d gemm peak: 17.5 / 8.6 GFlop/s

General (LU) Solver



GPU : NVIDIA GeForce GTX 280

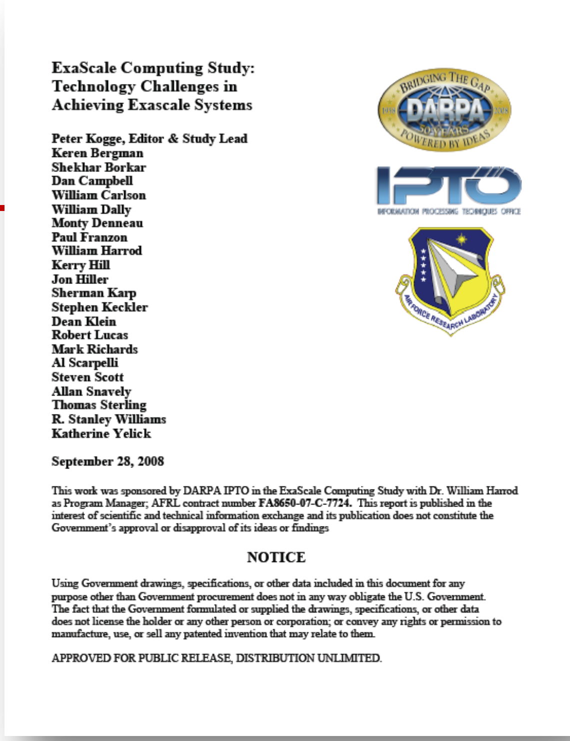
CPU : Intel Xeon dual socket quad-core @2.33 GHz

GPU BLAS : CUBLAS 2.2, s/d gemm peak: 375 / 75 GFlop/s

CPU BLAS : MKL 10.0 , s/d gemm peak: 17.5 / 8.6 GFlop/s

Exascale Computing

- Exascale systems are likely feasible by 2017±2
- 10-100 Million processing elements (cores or mini-cores) with chips perhaps as dense as 1,000 cores per socket, clock rates will grow more slowly
- 3D packaging likely
- Large-scale optics based interconnects
- 10-100 PB of aggregate memory
- Hardware and software based fault management
- Heterogeneous cores
- Performance per watt — stretch goal 100 GF/watt of sustained performance ⇒ >> 10 - 100 MW Exascale system
- Power, area and capital costs will be significantly higher than for today's fastest systems



ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley Williams
Katherine Yelick

September 28, 2008

This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724.** This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

Google: exascale computing study

Major Changes to Software

- Must rethink the design of our software
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software

Conclusions

- For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware.
- This strategy needs to be rebalanced - barriers to progress are increasingly on the software side.
- Moreover, the return on investment is more favorable to software.
  - Hardware has a half-life measured in years, while software has a half-life measured in decades.
- High Performance Ecosystem out of balance
  - Hardware, OS, Compilers, Software, Algorithms, Applications
    - No Moore's Law for software, algorithms and applications

Collaborators / Support

Employment opportunities for post-docs in the PLASMA/MAGMA projects

PLASMA Parallel Linear Algebra Software for Multicore Architectures

http://icl.cs.utk.edu/plasma/

MAGMA Matrix Algebra on GPU and Multicore Architectures

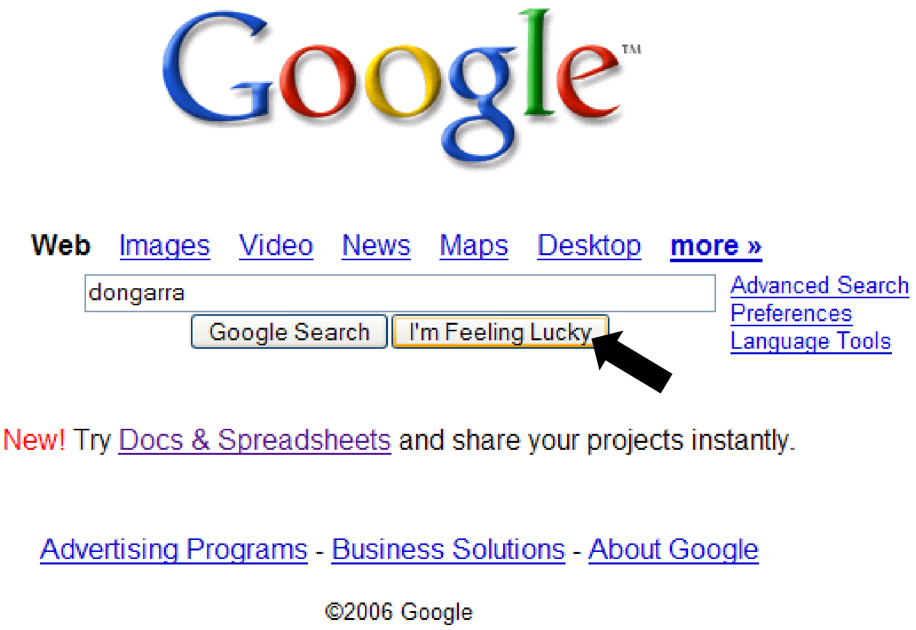
http://icl.cs.utk.edu/magma/

Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julie & Julien Langou, Hatem Ltaief, Piotr Luszczek, Stan Tomov

NVIDIA

The MathWorks

Microsoft



dongarra

New! Try Docs & Spreadsheets and share your projects instantly.

©2006 Google

From LAPACK (multithreaded)

```
// Matrix Initialization
~~DLARNV( &IONE, ISEED, &N, D );~~
DLAGSY( &N, &Nminus1, D, A,& N, ISEED, WORK, &INFO );

// Make A definite positive
For (int I = 0; I < N; I++)
      A( I, I ) = A( I, I ) + N;

// Cholesky Factorization
dpotrf(UPLO, &N, A, &LDA, &INFO);
```

To PLASMA (Multicores)

```
// Matrix Initialization
~~DLARNV( &IONE, ISEED, &N, D );~~
DLAGSY( &N, &Nminus1, D, A,& N, ISEED, WORK, &INFO );

// Make A definite positive
For (int I = 0; I < N; I++)
      A( I, I ) = A( I, I ) + N;

// Initialize PLASMA
INFO = PLASMA_Init( CORES);

// Cholesky Factorization
INFO = PLASMA_dpotrf(UPLO, N, A, LDA);

// Finalize PLASMA
INFO = PLASMA_Finalize( );
```

To MAGMA (1CPU-1GPU)

```
// CUDA Initialization
CublasInit();

// Matrix Allocation on the HOST
cudaMallocHost( (void**)&h_A,  N*N*sizeof(double) );
// Matrix Allocation on the DEVICE
cublasAlloc(N*N, sizeof(double), (void**)&d_A);

// Matrix Initialization
DLARNV( &IONE, ISEED, &N, D );
DLAGSY( &N, &Nminus1, D, h_A,& N, ISEED, WORK, &INFO );

// Make h_A definite positive
For (int I = 0; I < N; I++)
      h_A( I, I ) = h_A( I, I ) + N;

// Cholesky Factorization
magma_dpotrf(UPLO, &N, h_A, &LDA, d_A, INFO);
```

To Multicores-MultiGPUs

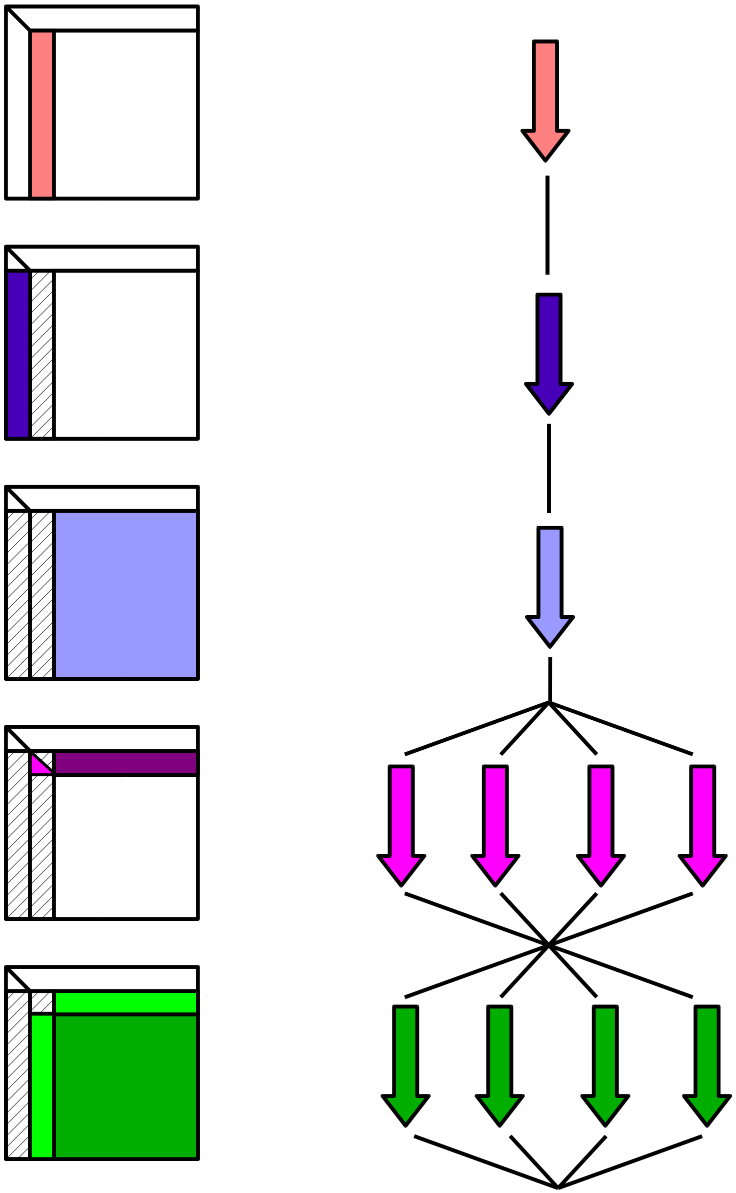
```
// CUDA Initialization
CublasInit();

// Matrix Allocation on the HOST
cudaMallocHost( (void**)&h_A,  N*N*sizeof(double) );
// Matrix Allocation on the DEVICE
cublasAlloc(N*N, sizeof(double), (void**)&d_A);

// Matrix Initialization
DLARNV( &IONE, ISEED, &N, D );
DLAGSY( &N, &Nminus1, D, h_A,& N, ISEED, WORK, &INFO );

// Make h_A definite positive
For (int I = 0; I < N; I++)
      h_A( I, I ) = h_A( I, I ) + N;

// Attach a  DEVICE to a core
cudaSetDevice(my_core_id);

// Cholesky Factorization
PLASMA_dpotrf(UPLO, &N, h_A, &LDA, d_A, INFO);
```

Coding for an Abstract Multicore

Parallel software for multicores should have two characteristics:

- Fine granularity:
  - High level of parallelism is needed
  - Cores will probably be associated with relatively small local memories. This requires splitting an operation into tasks that operate on small portions of data in order to reduce bus traffic and improve data locality.
- Asynchronicity:
  - As the degree of thread level parallelism grows and granularity of the operations becomes smaller, the presence of synchronization points in a parallel execution seriously affects the efficiency of an algorithm.

Steps in the LAPACK LU



DGETF2
(Factor a panel)
LAPACK

DLSWP
(Backward swap)
LAPACK

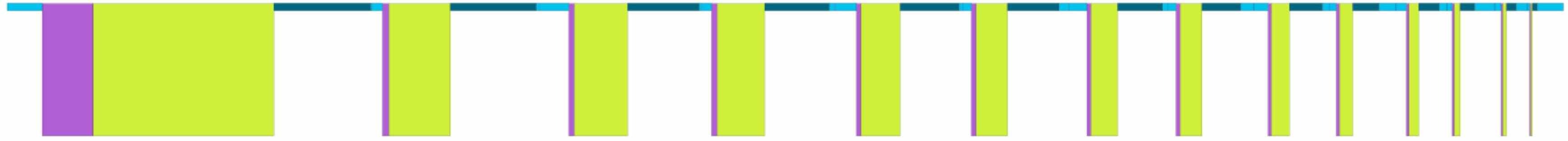
DLSWP
(Forward swap)
LAPACK

DTRSM
(Triangular solve)
BLAS

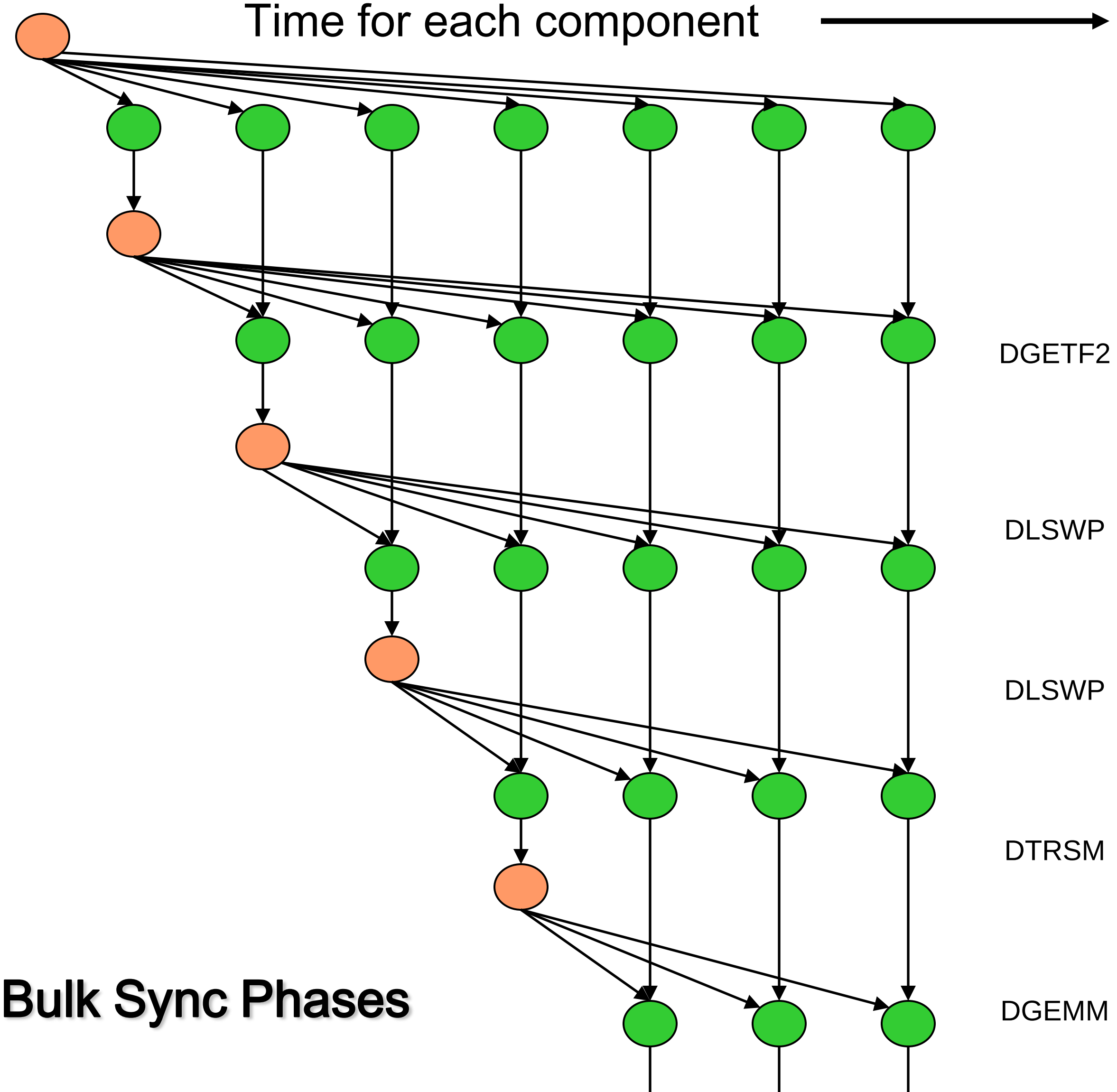
DGEMM
(Matrix multiply)
BLAS

LU Timing Profile (16 core system)

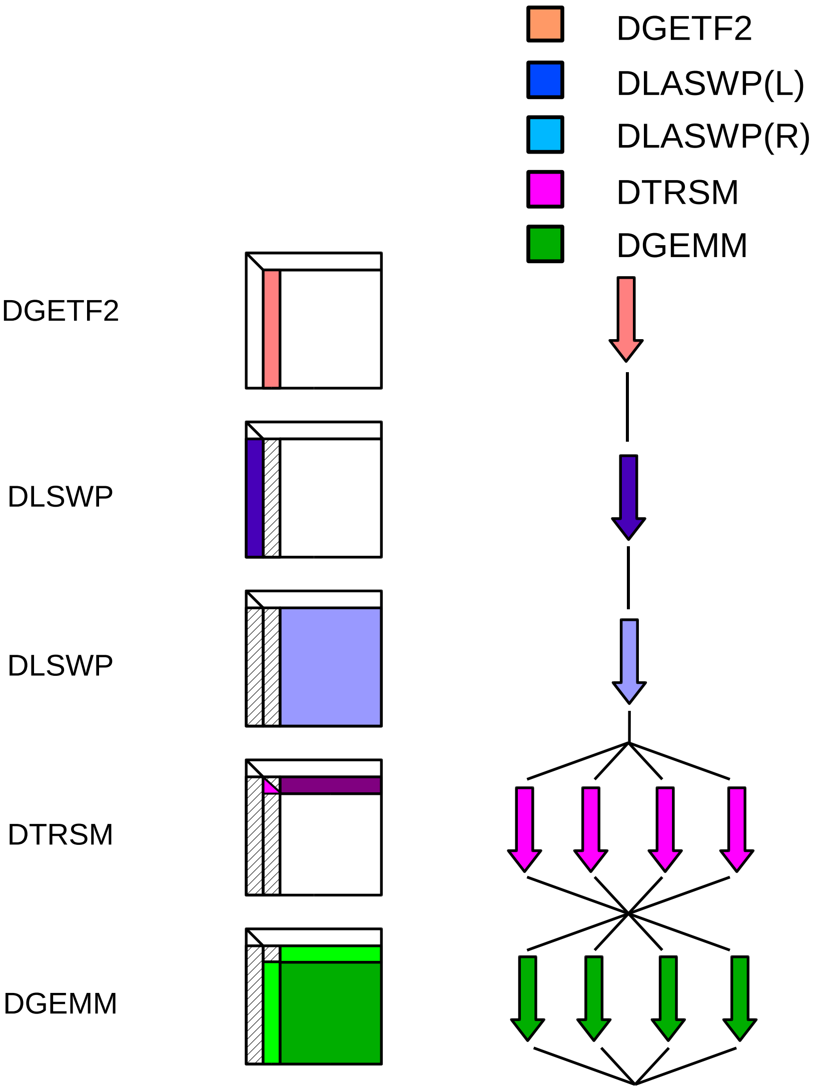
Threads – no lookahead



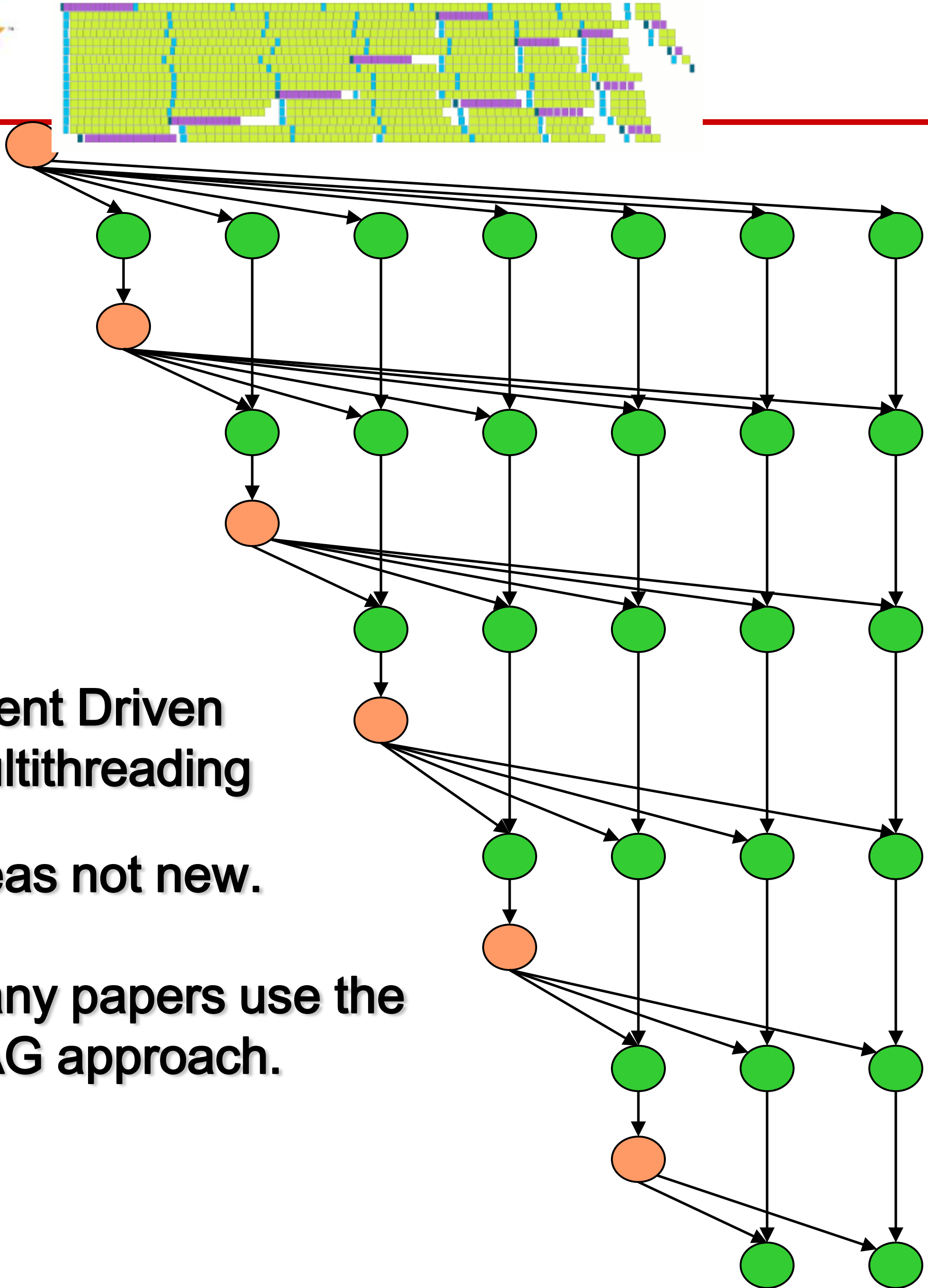
Time for each component



Bulk Sync Phases



Adaptive Lookahead - Dynamic



Event Driven Multithreading

Ideas not new.

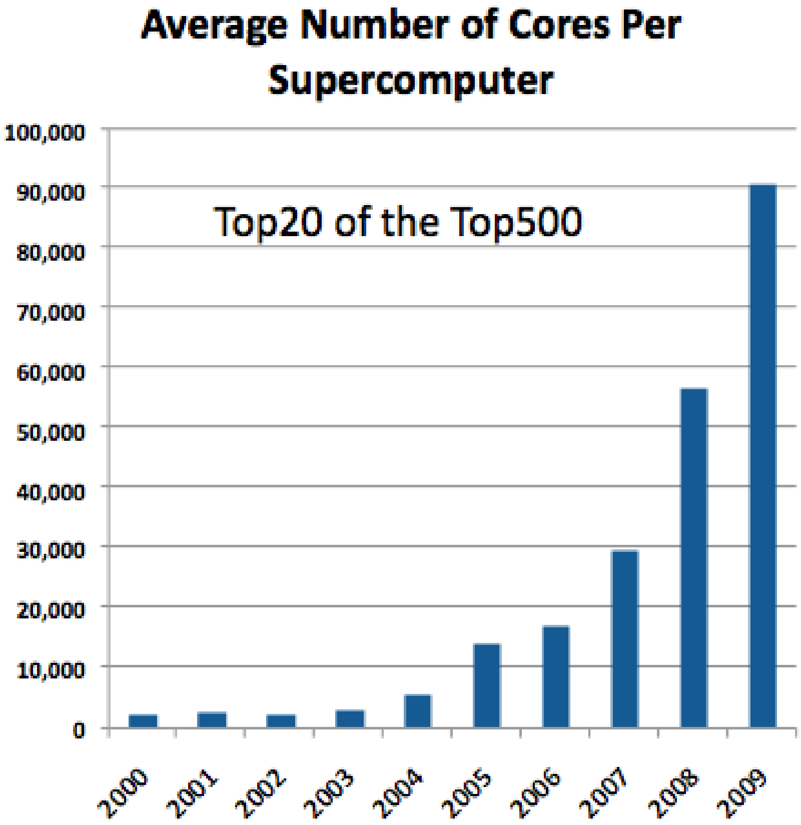
Many papers use the DAG approach.

```
while(1)
    fetch_task();
    switch(task.type) {
        case PANEL:
            dgetf2();
            update_progress();
        case COLUMN:
            dlaswp();
            dtrsm();
            dgemm();
            update_progress();
        case END:
            for()
                dlaswp();
            return;
    }
}
```

Reorganizing algorithms to use this approach

Moore's Law Reinterpreted

- Number of cores per chip doubles every 2 year, while clock speed decreases (not increases).
  - Need to deal with systems with millions of concurrent threads
    - Future generation will have billions of threads!
  - Need to be able to easily replace inter-chip parallelism with intro-chip parallelism
- Number of threads of execution doubles every 2 year


Average Number of Cores Per Supercomputer
Top20 of the Top500