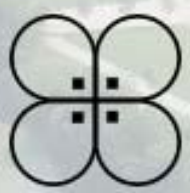


International ACM Symposium on
High Performance Distributed Computing

HPDC 2009, Munich, Germany, June 11-13, 2009

An Overview of High Performance Computing and Challenges for the Future

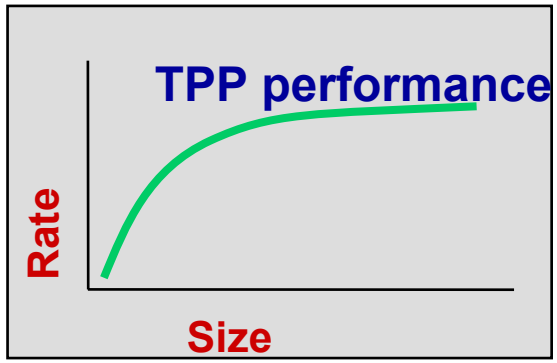
Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

H. Meuer, H. Simon, E. Strohmaier, & JD

- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP

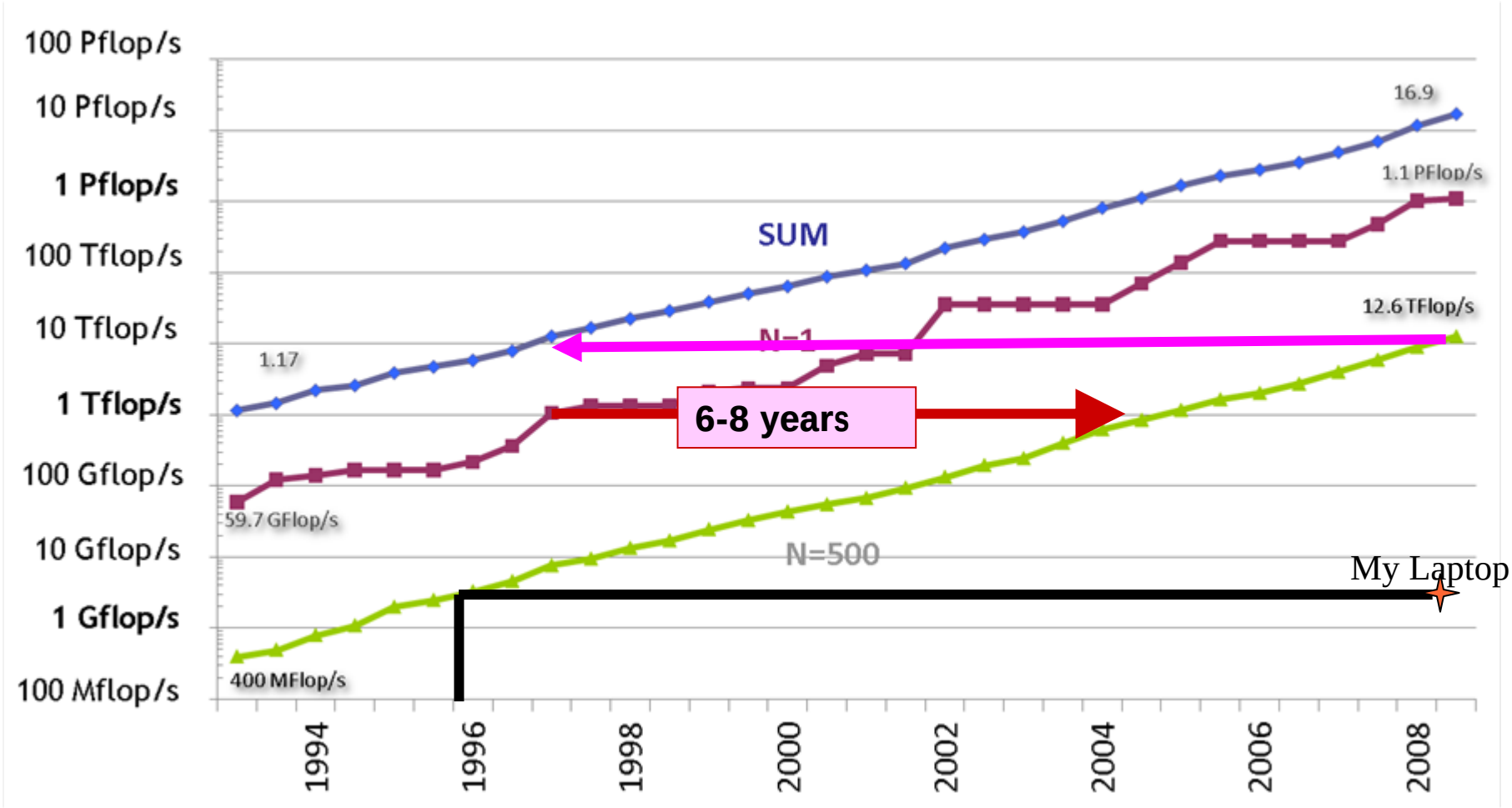
  $Ax=b$, *dense problem*



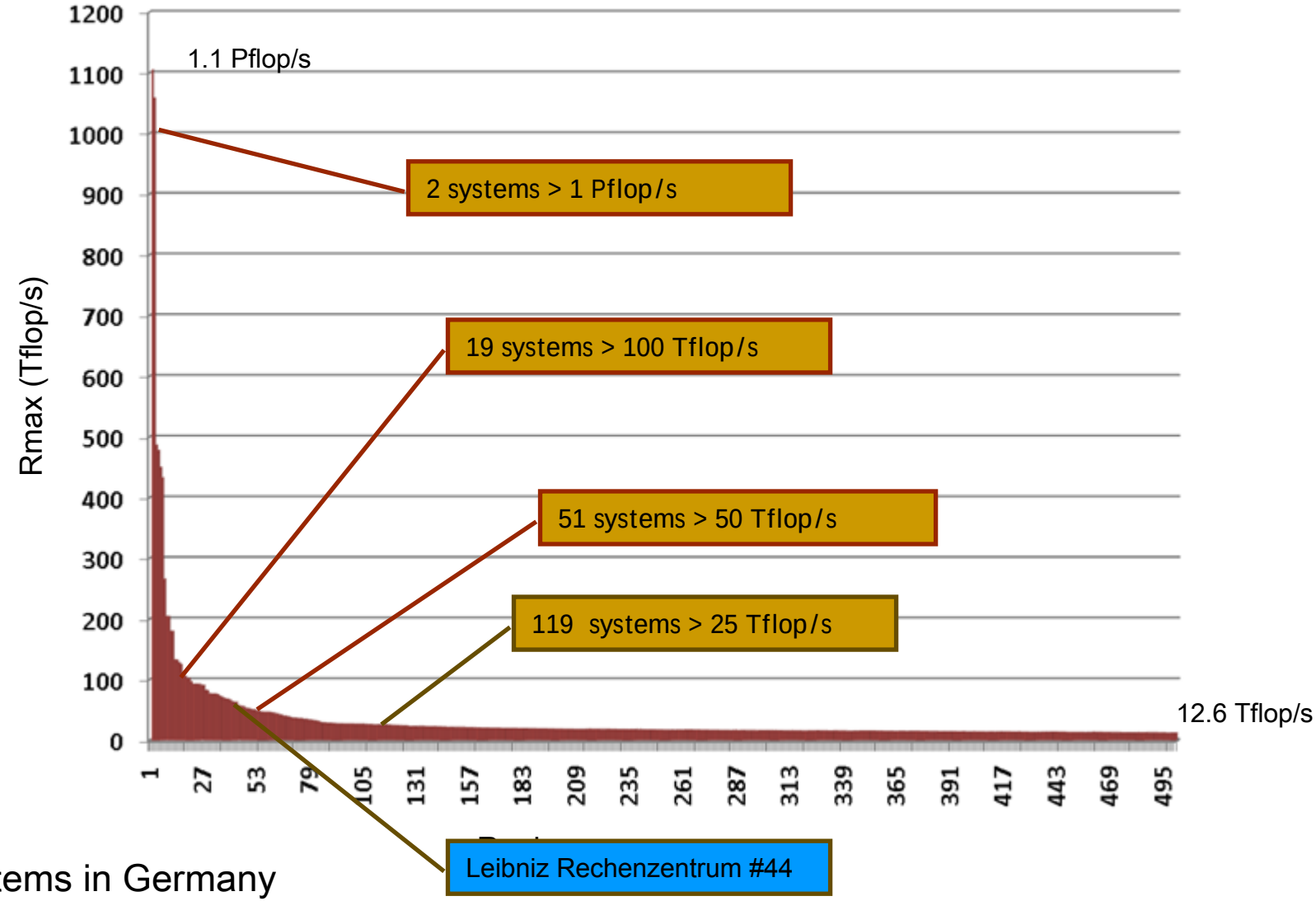
- Updated twice a year
  SC‘xy in the States in November
  Meeting in Germany in June

- All data available from **www.top500.org**

Performance Development



Distribution of the Top500



24 systems in Germany
267 systems replaced last time

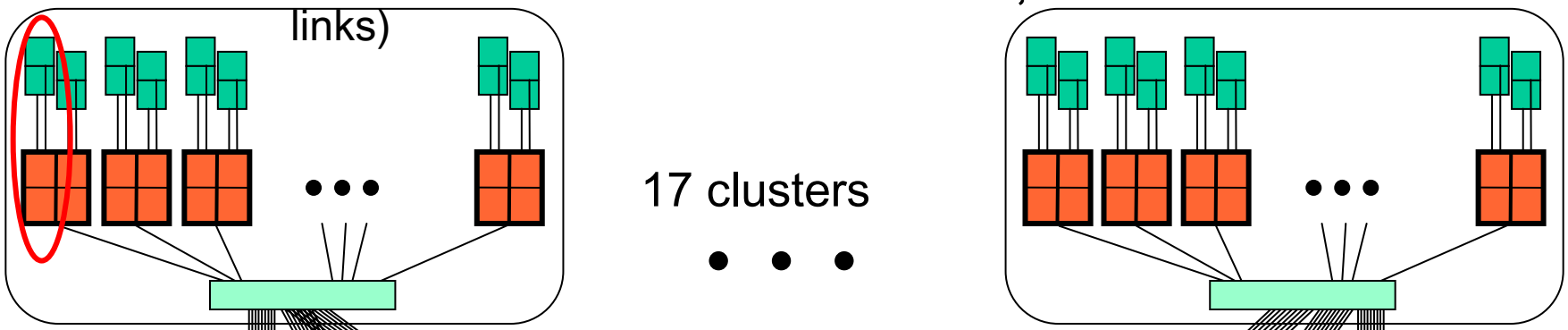
32nd List: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Tflops]	Rmax/ Rpeak	Power [MW]	MF/W
1	DOE/NNSA Los Alamos Nat Lab	IBM / Roadrunner - BladeCenter QS22/LS21	USA	129600	1105.0	76%	2.48	445
2	DOE/OS Oak Ridge Nat Lab	Cray / Jaguar - Cray XT5 QC 2.3 GHz	USA	150152	1059.0	77%	6.95	152
3	NASA/Ames Research Center/NAS	SGI / Pleiades - SGI Altix ICE 8200EX	USA	51200	487.0	80%	2.09	233
4	DOE/NNSA Lawrence Livermore NL	IBM / eServer Blue Gene Solution	USA	212992	478.2	80%	2.32	205
5	DOE/OS Argonne Nat Lab	IBM / Blue Gene/P Solution	USA	163840	450.3	81%	1.26	357
6	NSF TACC/ Univ. of Texas	Sun / Ranger - SunBlade x6420	USA	62976	433.2	75%	2.0	217
7	DOE/OS/NERSC/Lawrence Berkeley NL	Cray / Franklin - Cray XT4	USA	38642	266.3	75%	1.15	232
8	DOE/OS Oak Ridge Nat Lab	Cray / Jaguar - Cray XT4	USA	30976	205.0	79%	1.58	130
9	DOE/NNSA Sandia Nat Lab	Cray / Red Storm - XT3/4	USA	38208	204.2	72%	2.5	81
10	Shanghai Supercomputer Center	Dawning 5000A, Windows HPC 2008	China	30720	180.6	77%	.85	212

LANL Roadrunner
A Petascale System in 2008

"Connected Unit" cluster
192 Opteron nodes
(180 w/ 2 dual-Cell blades
connected w/ 4 PCIe x8
links)

≈ 13,000 Cell HPC chips
≈ 1.33 PetaFlop/s (from Cell)
≈ 7,000 dual-core Opterons
≈ 122,000 cores

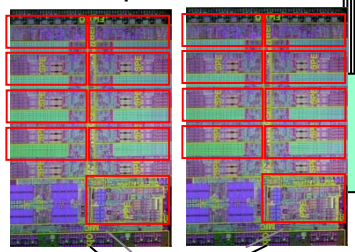


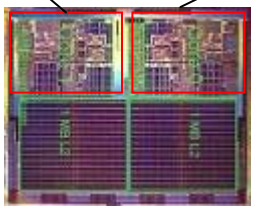
17 clusters

2nd stage InfiniBand 4x DDR interconnect
(18 sets of 12 links to 8 switches)

2nd stage InfiniBand interconnect (8 switches)

Cell chip for each core





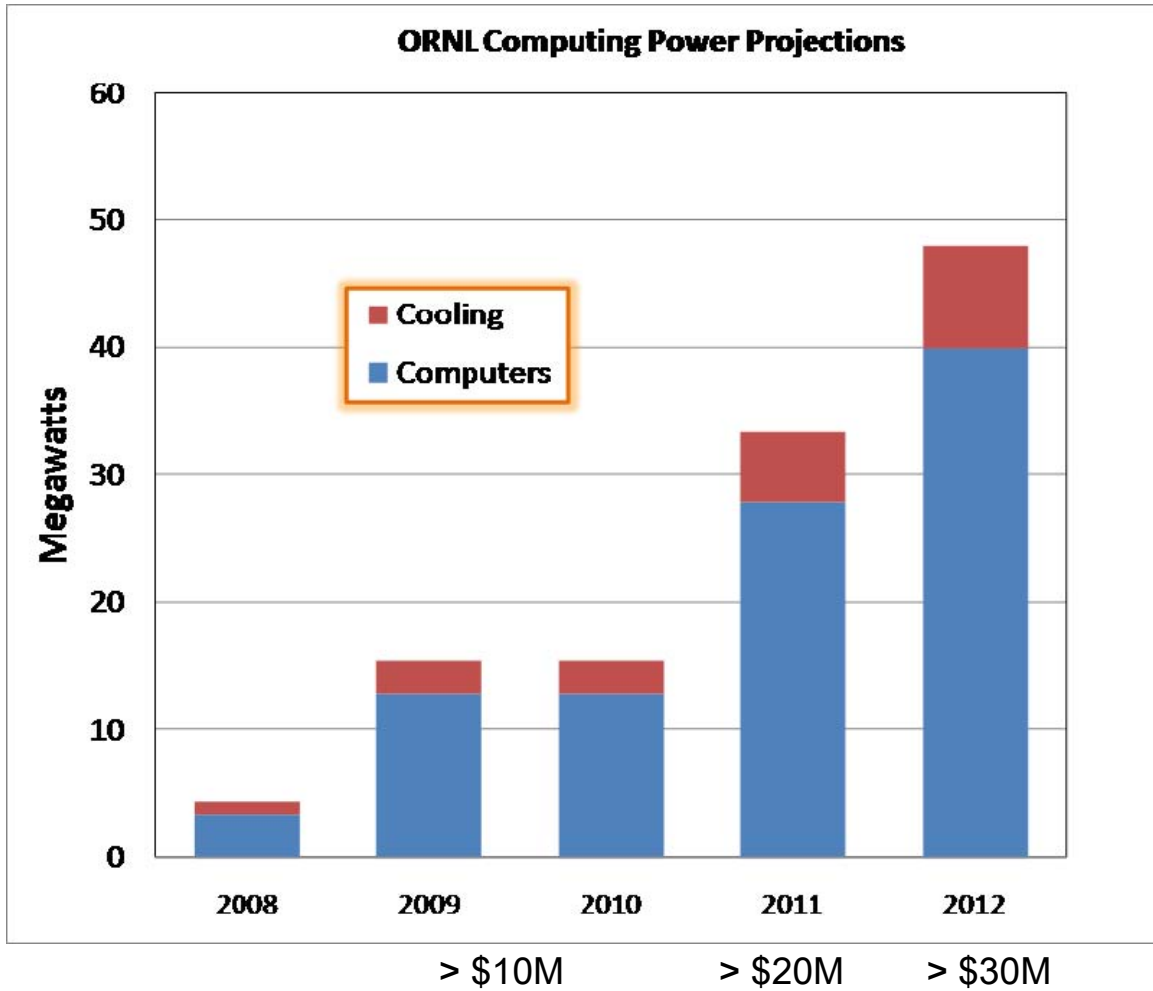
Dual Core Opteron Chip

Based on the 100 Gflop/s (DP) Cell chip

Hybrid Design (2 kinds of chips & 3 kinds of cores)
Programming required at 3 levels.

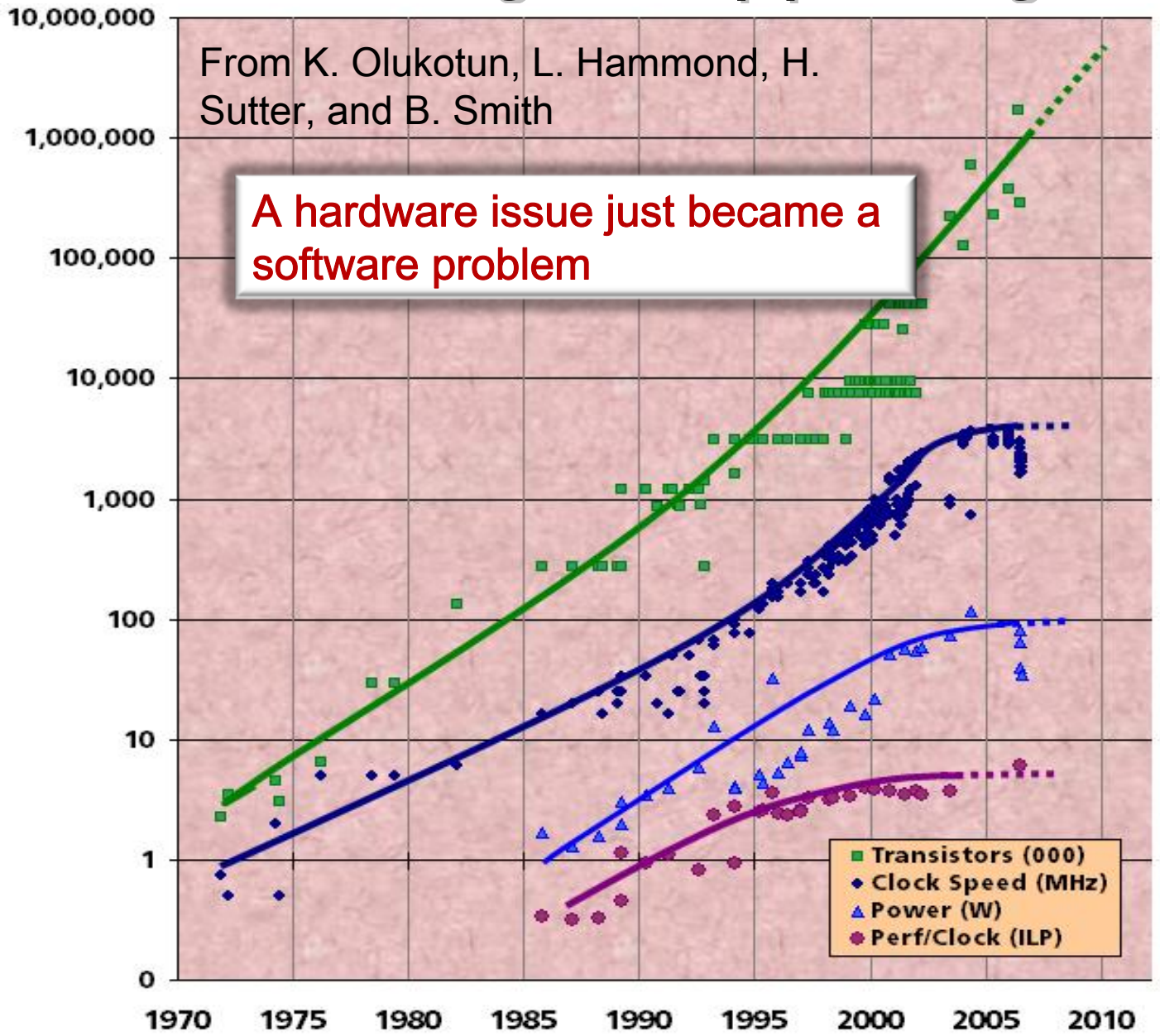
ORNL/UTK Computer Power Cost Projections 2008-2012

- Over the next 5 years ORNL/UTK will deploy 2 large Petascale systems
- Using 15 MW today
- By 2012 close to 50MW!!
- Power costs greater than $10M today.
- Cost estimates based on $0.07 per KwH



Power becomes the architectural driver for future large systems

Something's Happening Here...

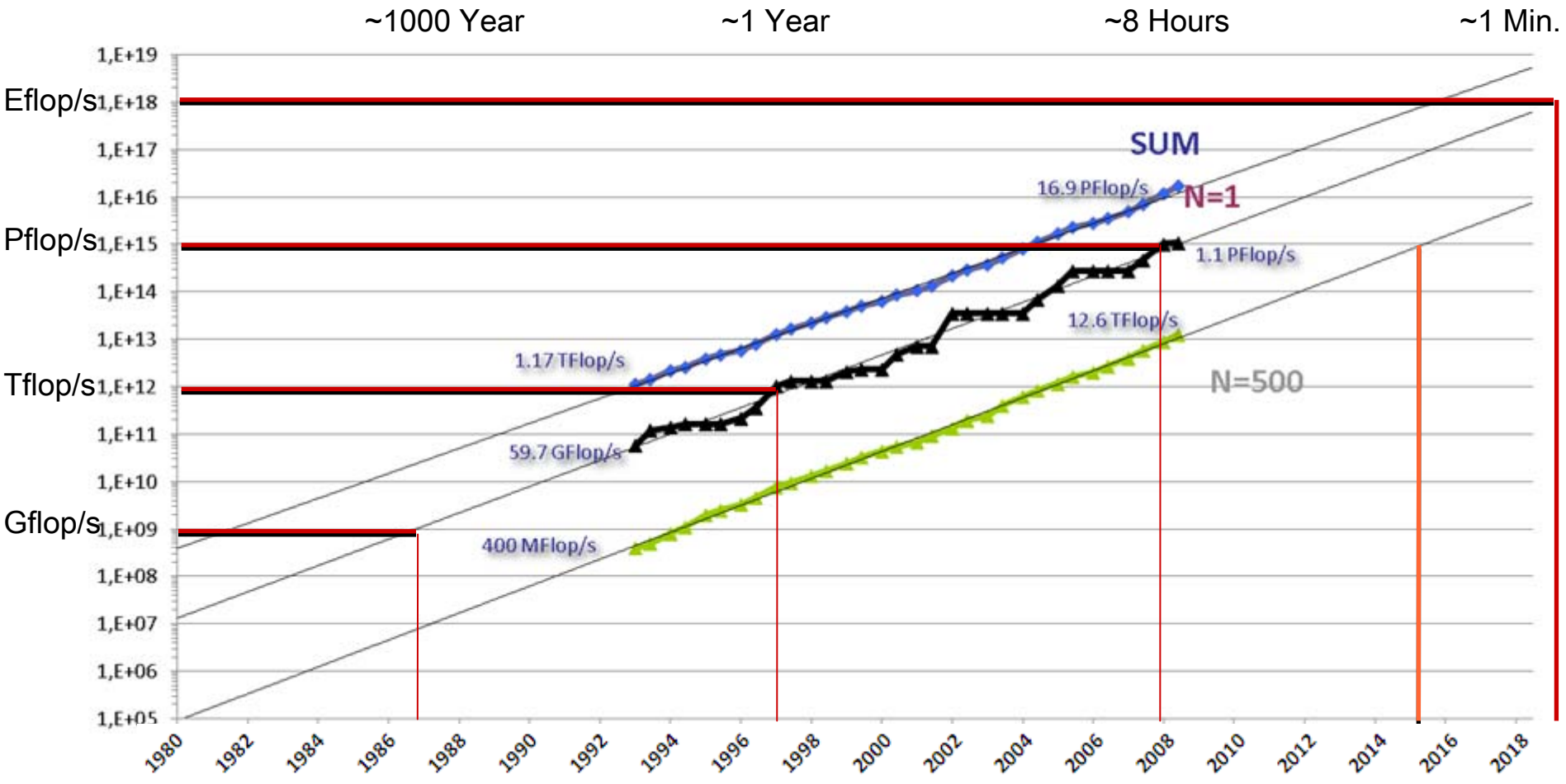


- **In the "old days" it was: each year processors would become faster**
- **Today the clock speed is fixed or getting slower**
- **Things are still doubling every 18 -24 months**
- **Moore's Law reinterpretated.**
  - **Number of cores double every 18-24 months**

Moore's Law Reinterpreted

- **Number of cores per chip doubles every 2 year, while clock speed remains fixed or decreases**
- **Need to deal with systems with millions of concurrent threads**
  - **Future generation will have billions of threads!**
- **Number of threads of execution doubles every 2 year**

Performance Development and Projections



Cray 2
1 Gflop/s
O(1) Thread

ASCI Red
1 Tflop/s
$O(10^3)$ Threads

RoadRunner
1.1 Pflop/s
$O(10^6)$ Threads

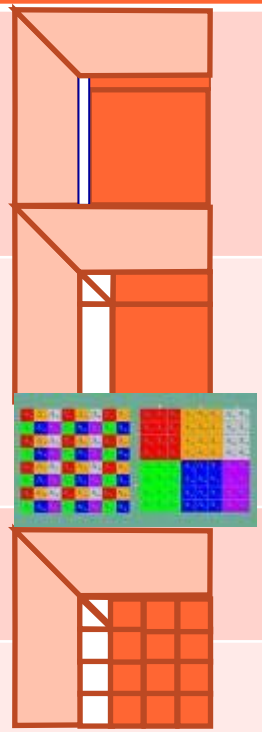
1 Eflop/s
$O(10^9)$ Threads

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**

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.

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

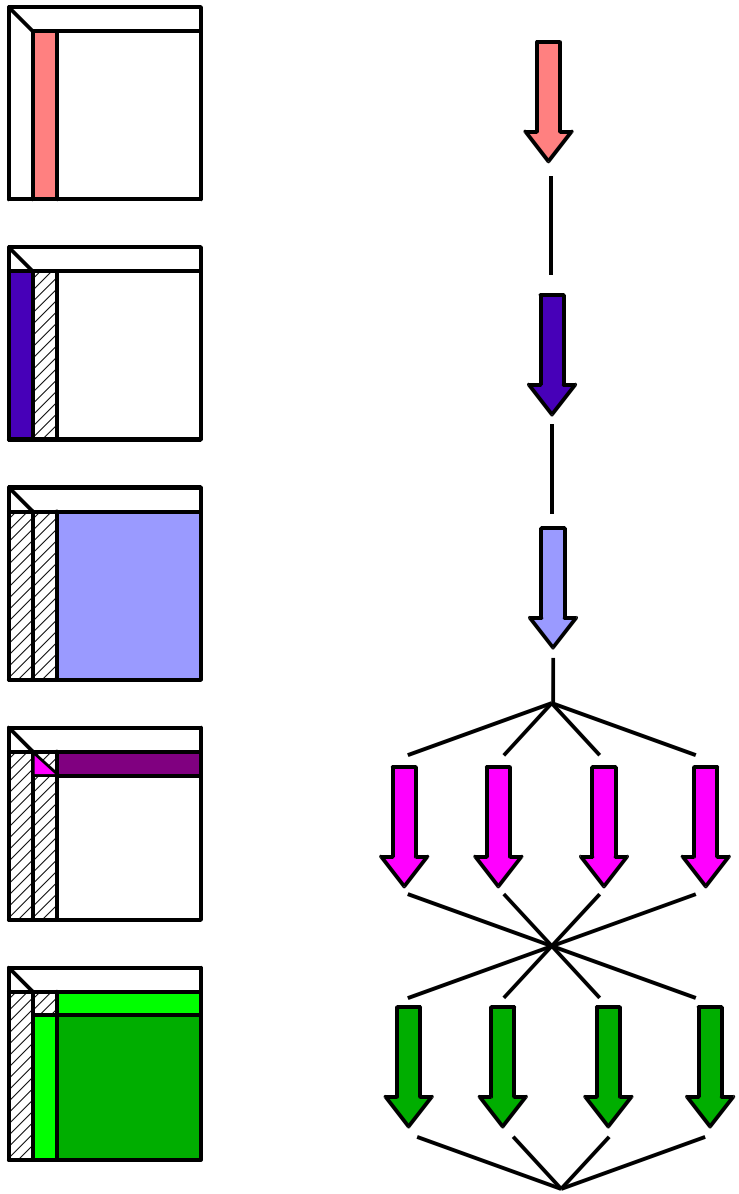
DGETF2
(Factor a panel)

DLSWP
(Backward swap)

DLSWP
(Forward swap)

DTRSM
(Triangular solve)

DGEMM
(Matrix multiply)



LAPACK

LAPACK

LAPACK

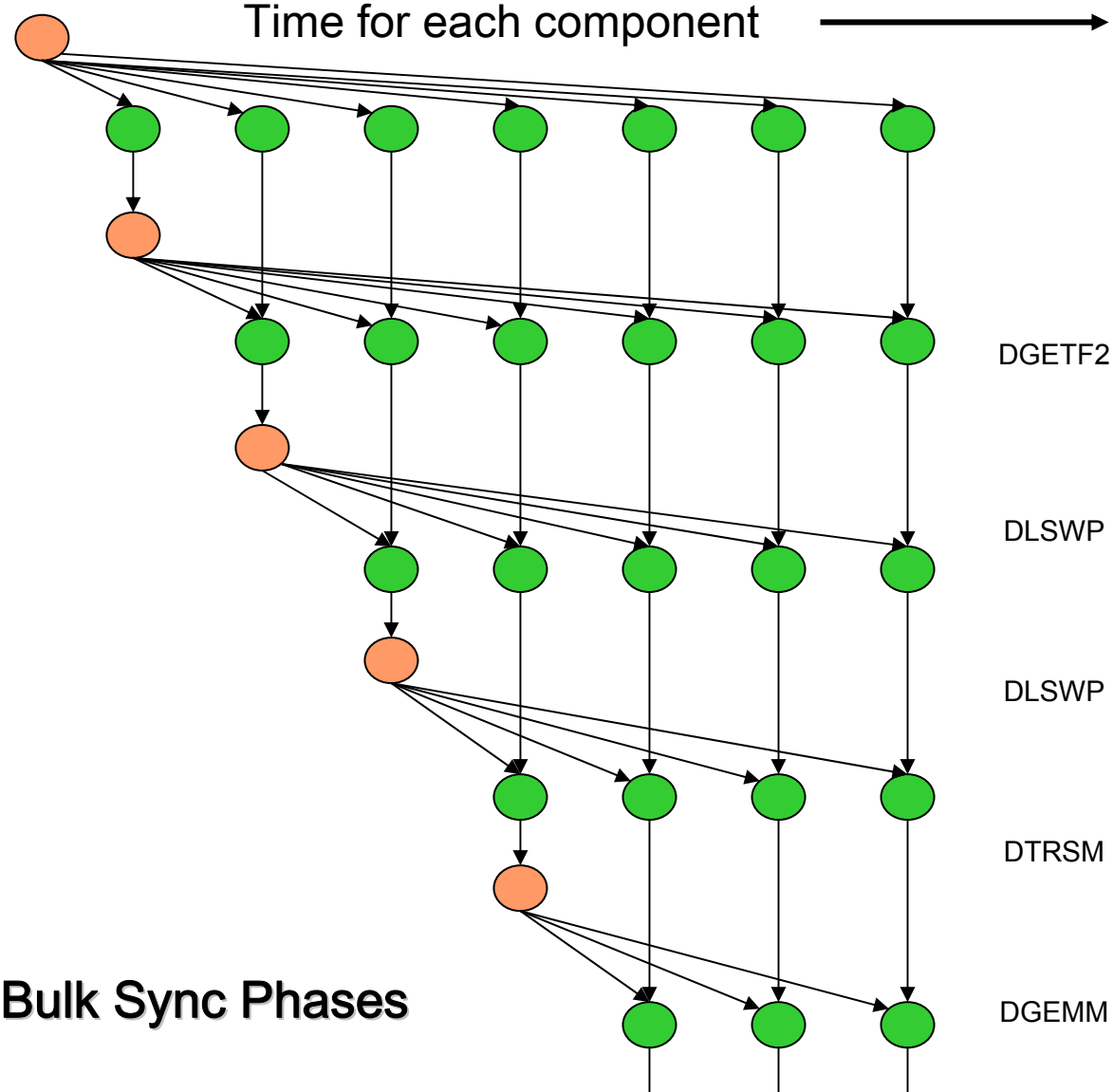
BLAS

BLAS

LU Timing Profile (16 core system)

Threads – no lookahead

Time for each component



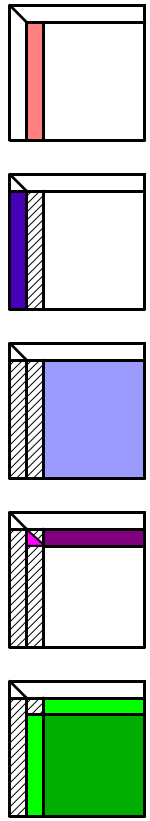
DGETF2

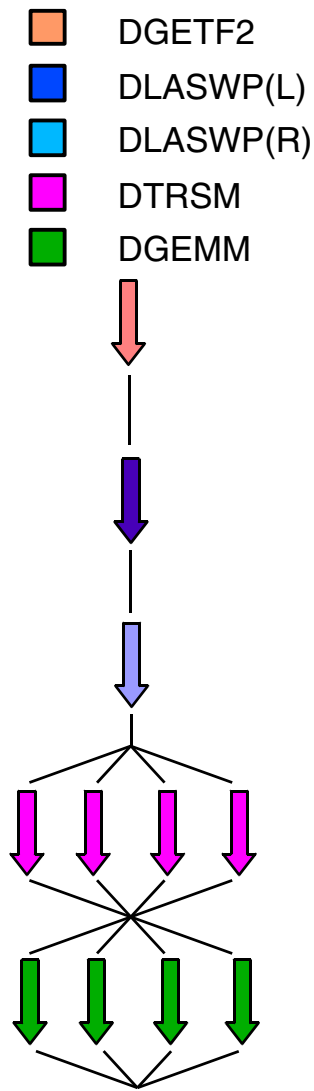
DLSWP

DLSWP

DTRSM

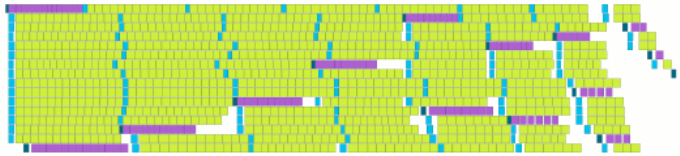
DGEMM

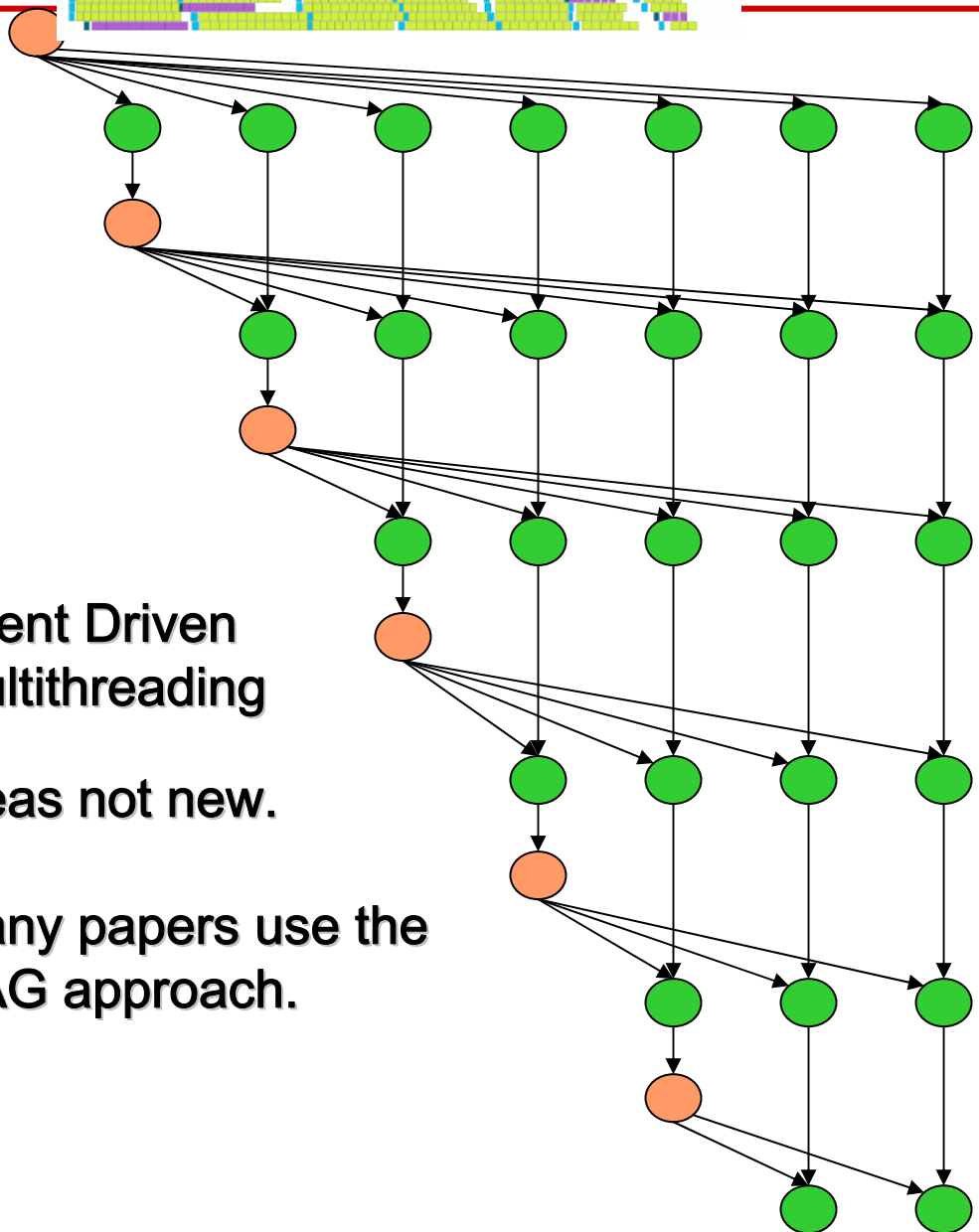




Bulk Sync Phases

Adaptive Lookahead - Dynamic





Event Driven Multithreading

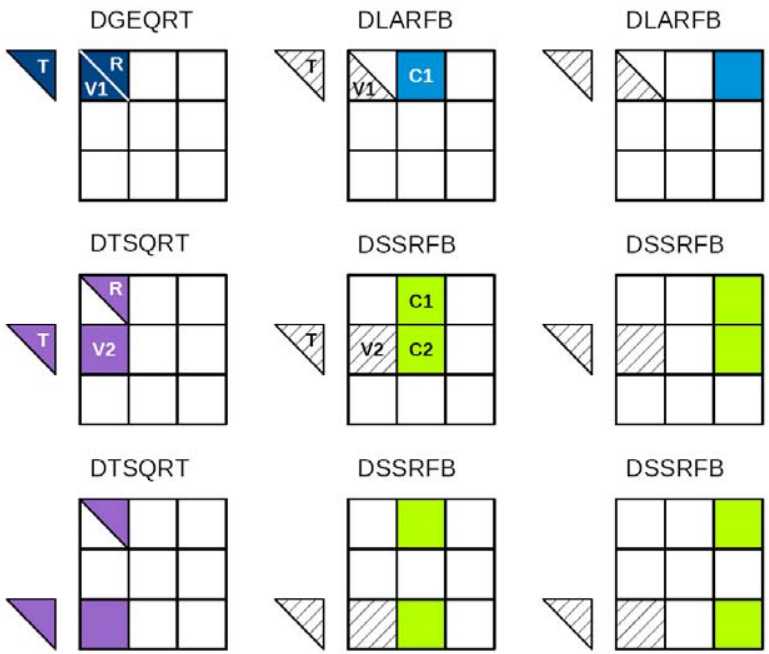
Ideas not new.

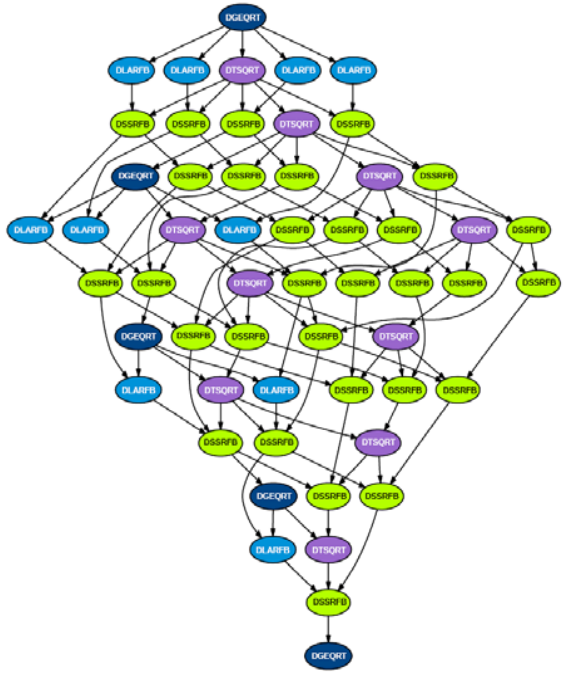
Many papers use the DAG approach.

```
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

Tile QR (&LU) Algorithms





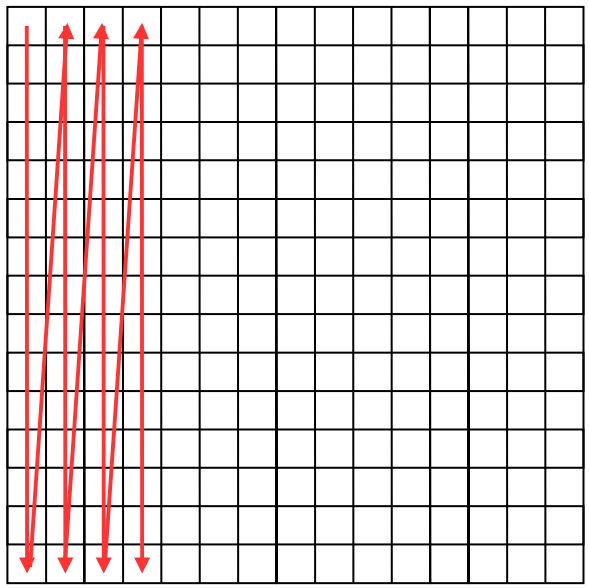
```
FOR k = 0..TILES-1
   A[k][k], T[k][k] ← DGRQRT(A[k][k])
   FOR m = k+1..TILES-1
      A[k][k], A[m][k], T[m][k] ← DTSQRT(A[k][k], A[m][k], T[m][k])
   FOR n = k+1..TILES-1
      A[k][n] ← DLARFB(A[k][k], T[k][k], A[k][n])
      FOR m = k+1..TILES-1
         A[k][n], A[m][n] ← DSSRFB(A[m][k], T[m][k], A[k][n], A[m][n])
```

- input matrix stored and processed by square tiles
- complex DAG

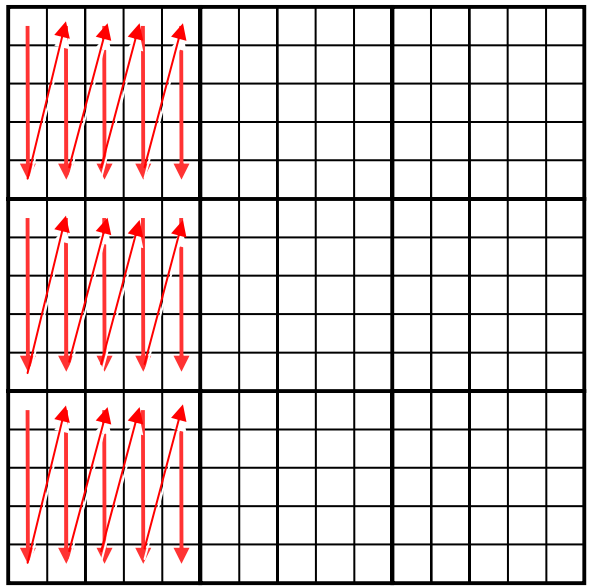
Achieving Fine Granularity

Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

Column-Major



Blocked



PLASMA (Redesign LAPACK/ScaLAPACK)

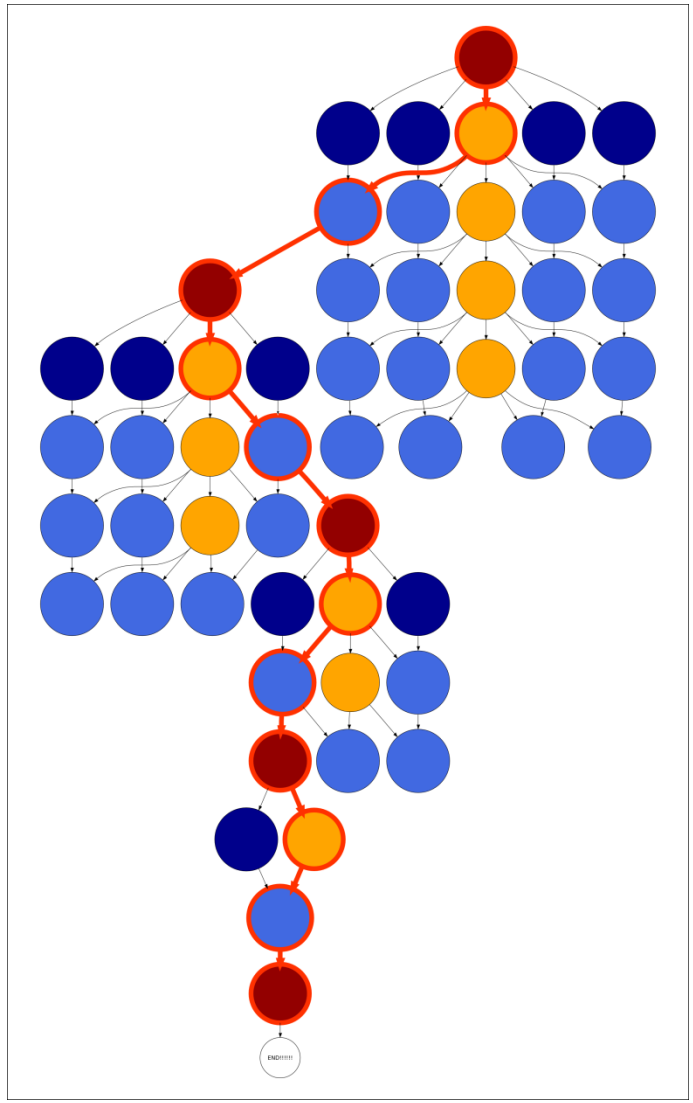
Parallel Linear Algebra Software for Multicore Architectures

- **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

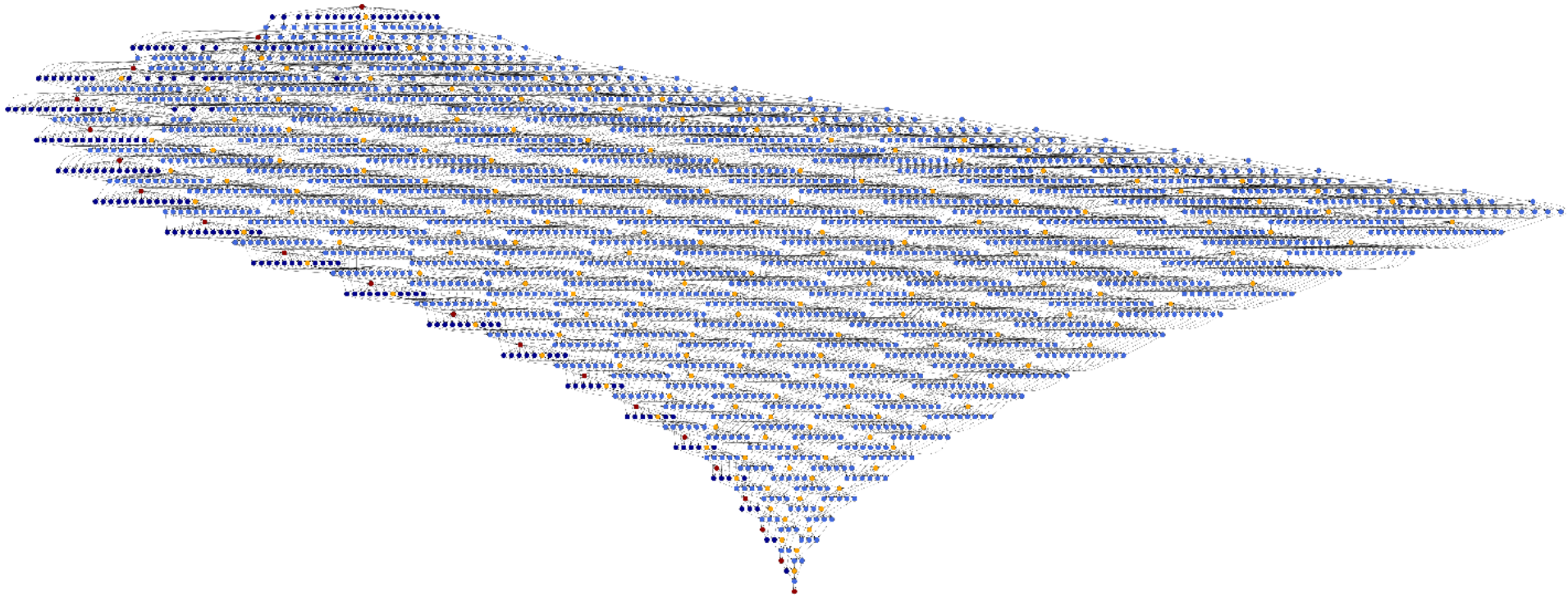
If We Had A Small Matrix Problem

- We would generate the DAG, find the critical path and execute it.
- DAG too large to generate ahead of time
  - Not explicitly generate
  - Dynamically generate the DAG as we go
- Machines will have large number of cores in a distributed fashion
  - Will have to engage in message passing
  - Distributed management
  - Locally have a run time system



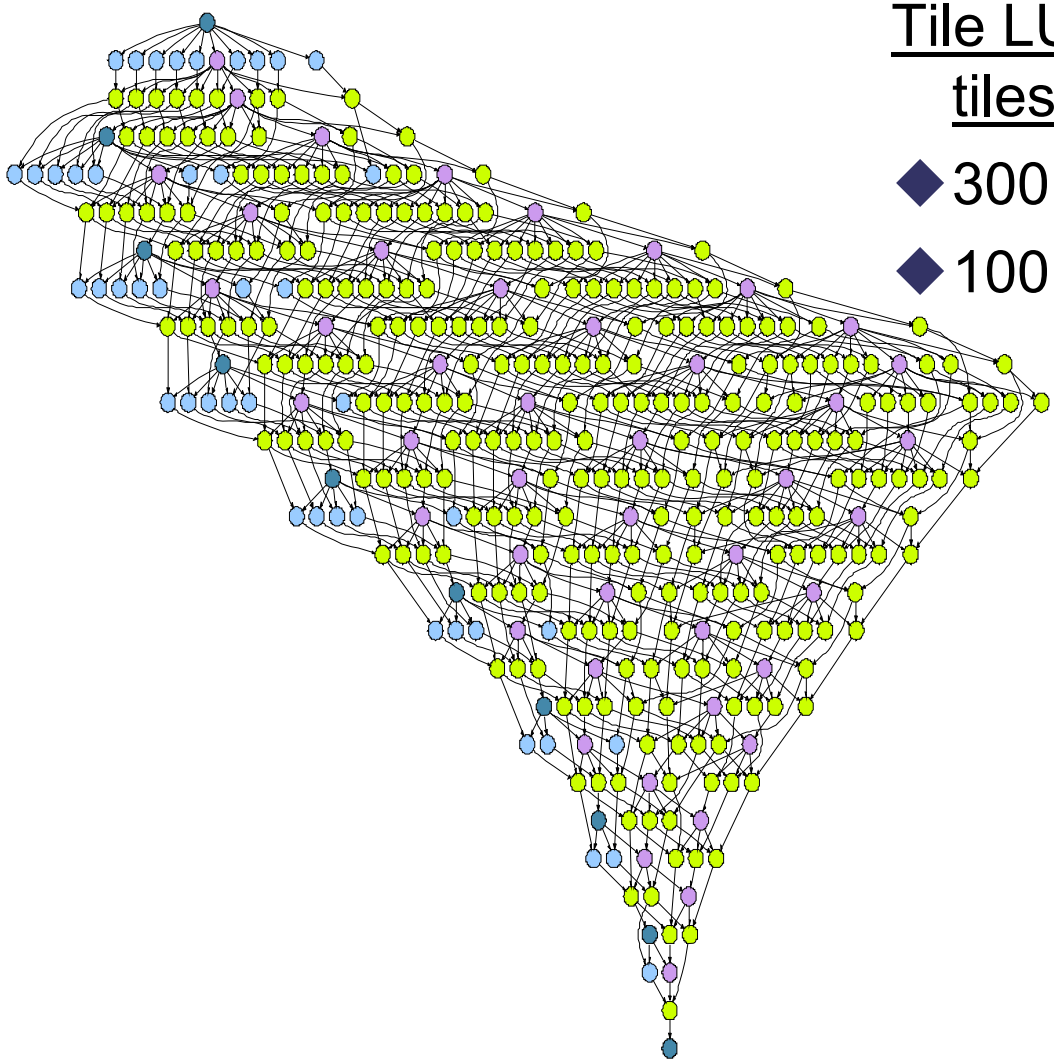
The DAGs are Large

- **Here is the DAG for a factorization on a 20 x 20 matrix**



- **For a large matrix say O(10^6) the DAG is huge**
- **Many challenges for the software**

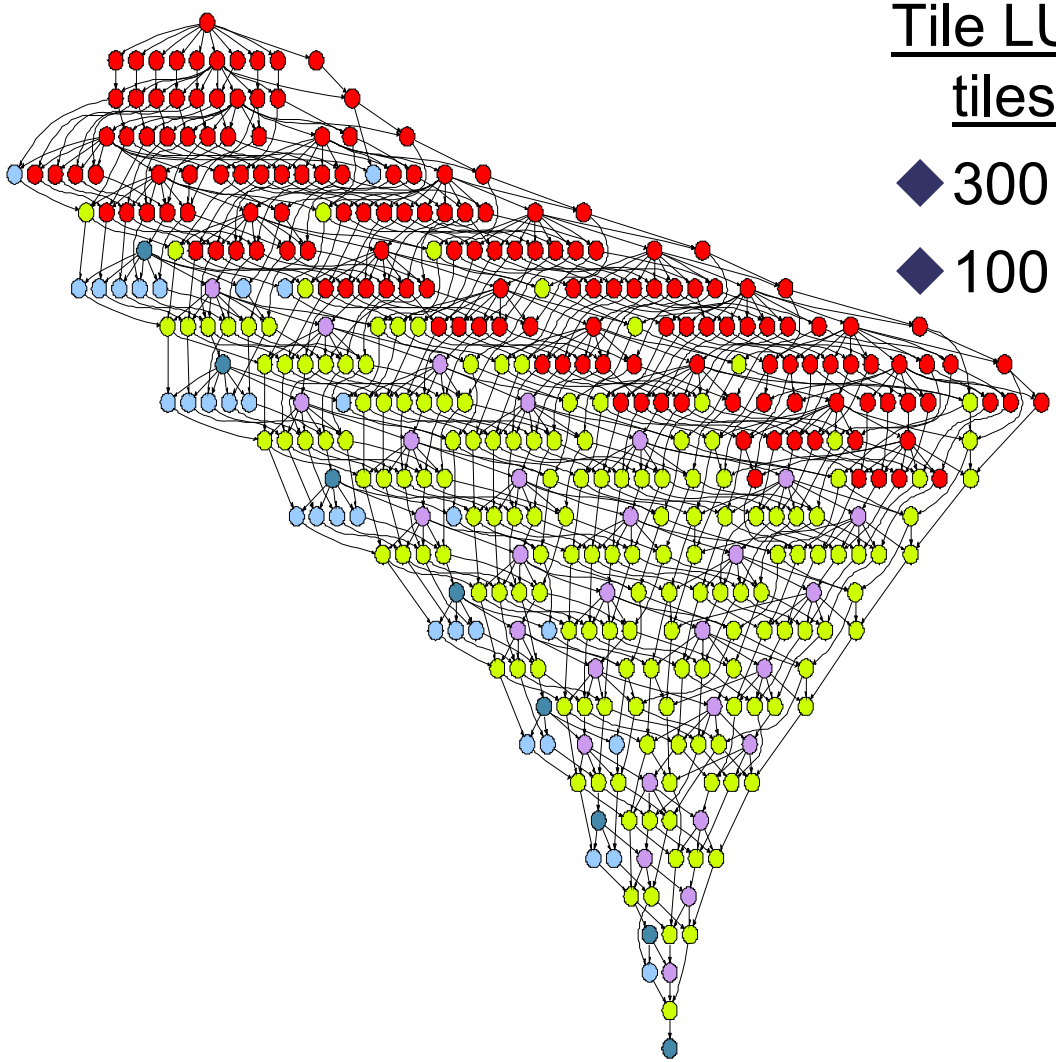
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

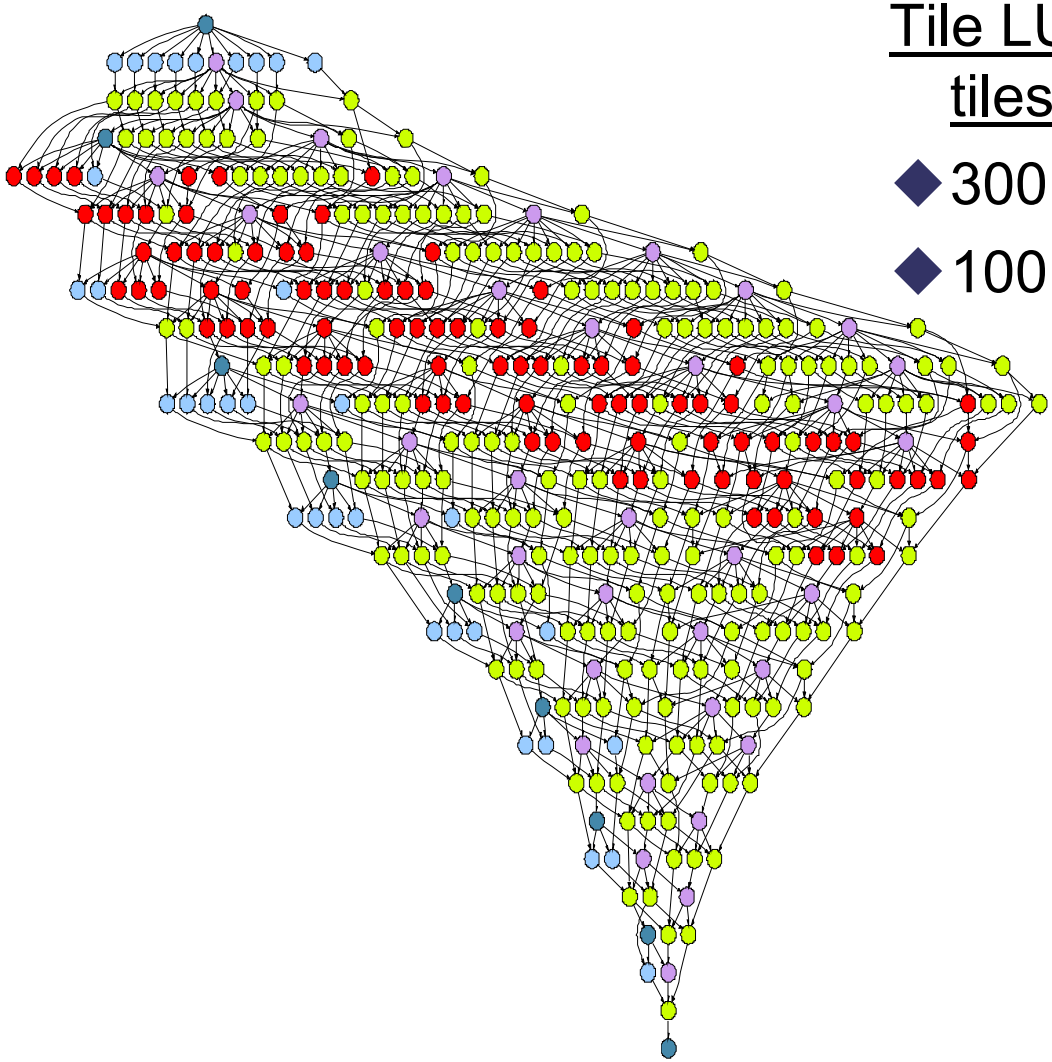
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

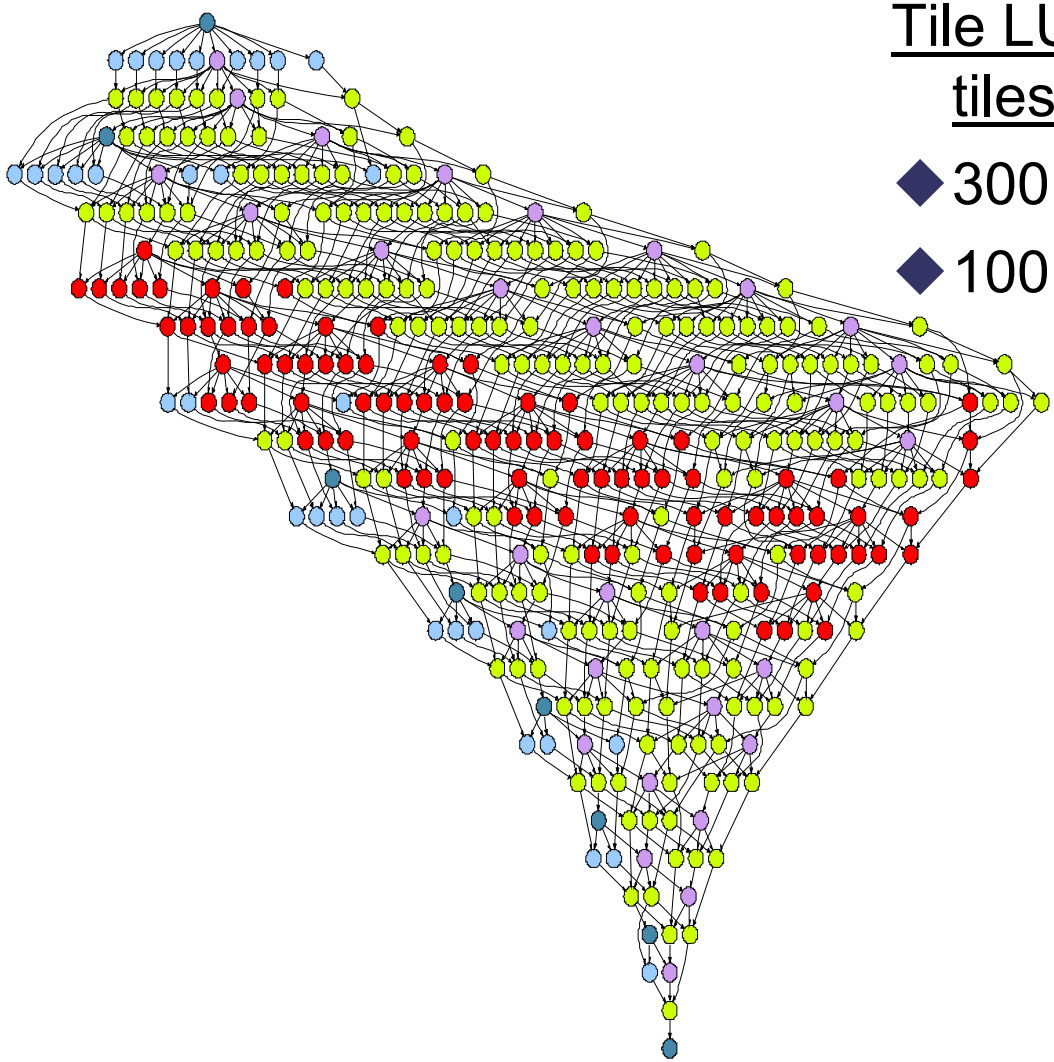
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

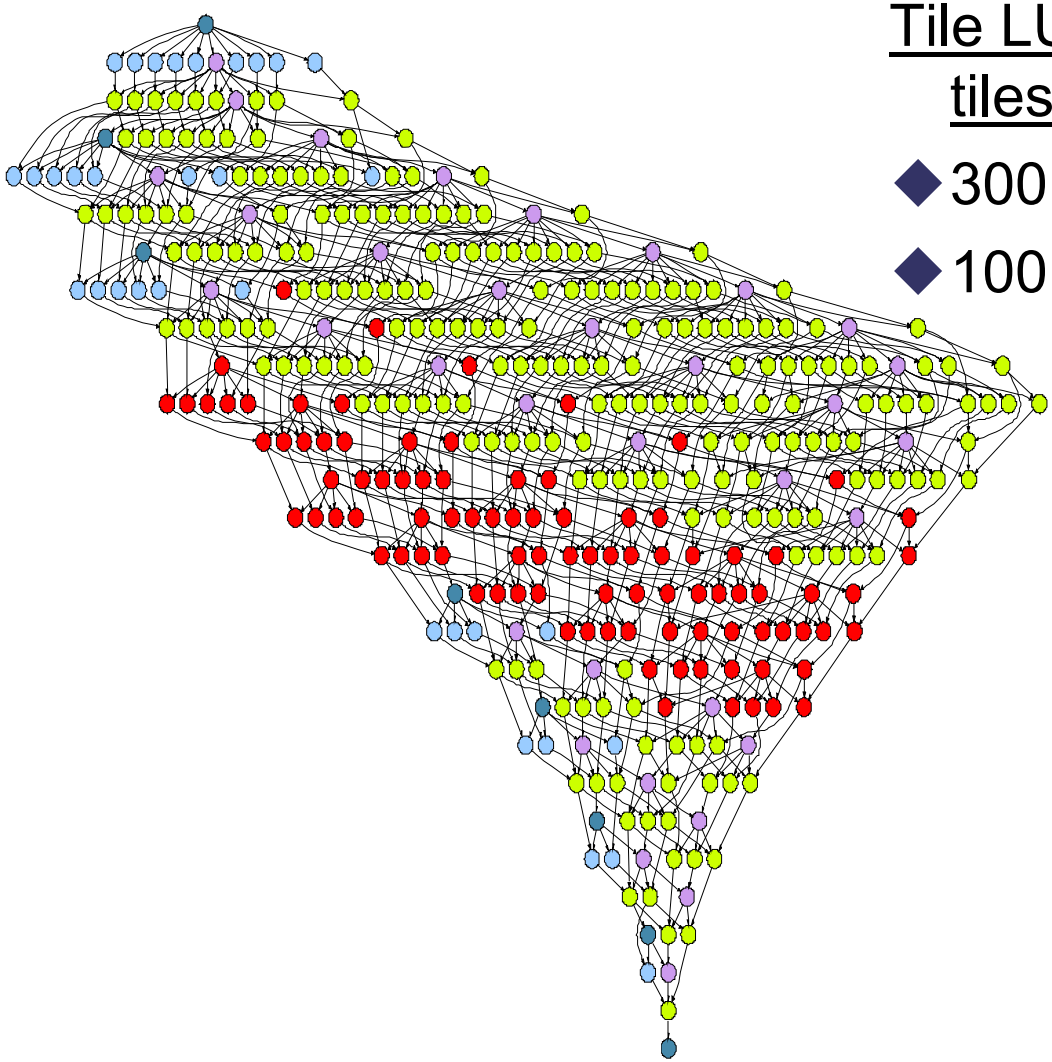
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

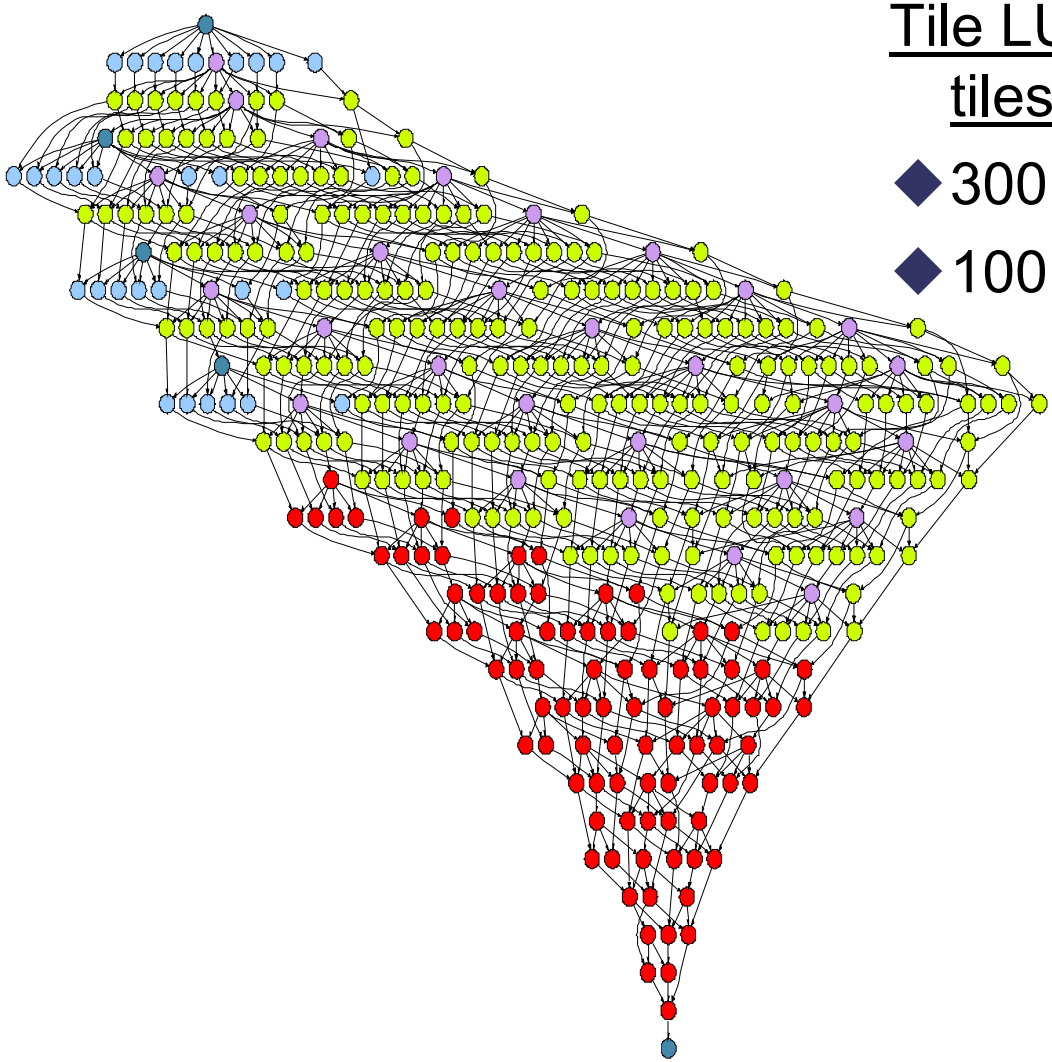
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

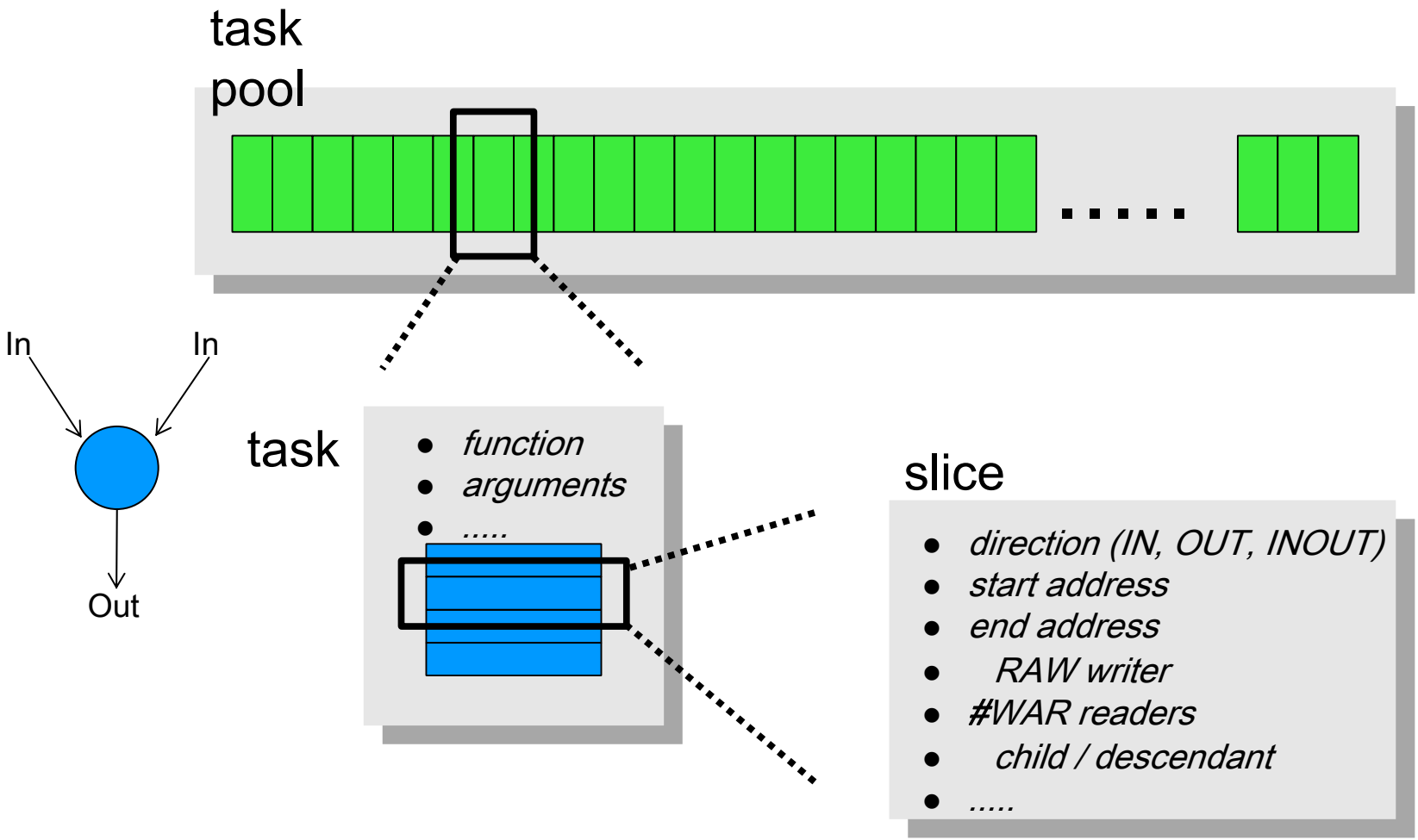
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

PLASMA Dynamic Task Scheduler



- task – a unit of scheduling (quantum of work)
- slice – a unit of dependency resolution (quantum of data)
- Current version uses one core to manage the task pool

How to Deal with Complexity?

- Many parameters in the code needs to be optimized.
- Software adaptivity is the key for applications to effectively use available resources whose complexity is exponentially increasing
- Goal:
  - Automatically bridge the gap between the application and computers that are rapidly changing and getting more and more complex
- Non obvious interactions between HW/SW can effect outcome

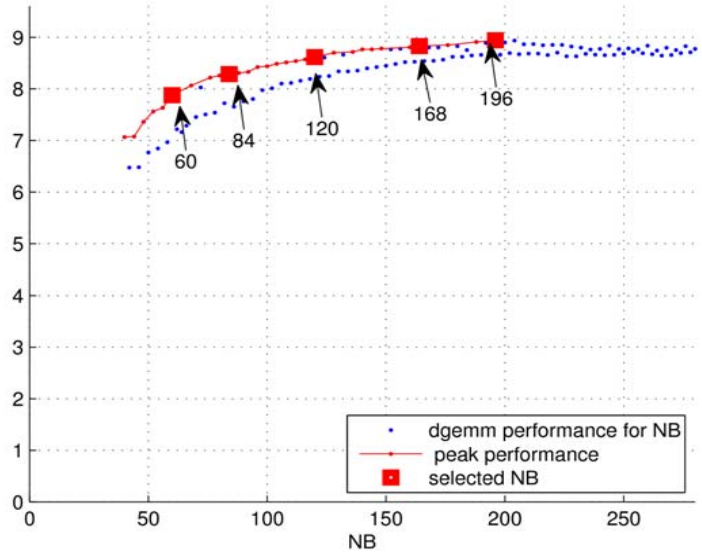
Auto-Tuning

- **Best algorithm implementation can depend strongly on the problem, computer architecture, compiler,…**
- **There are 2 main approaches**
  - **Model-driven optimization**
    [Analytical models for various parameters;
    Heavily used in the compilers community;
    May not give optimal results]
  - **Empirical optimization**
    [Generate large number of code versions and runs them on a given platform to determine the best performing one;
    Effectiveness depends on the chosen parameters to optimize and the search heuristics used]
- **Natural approach is to combine them in a hybrid approach**

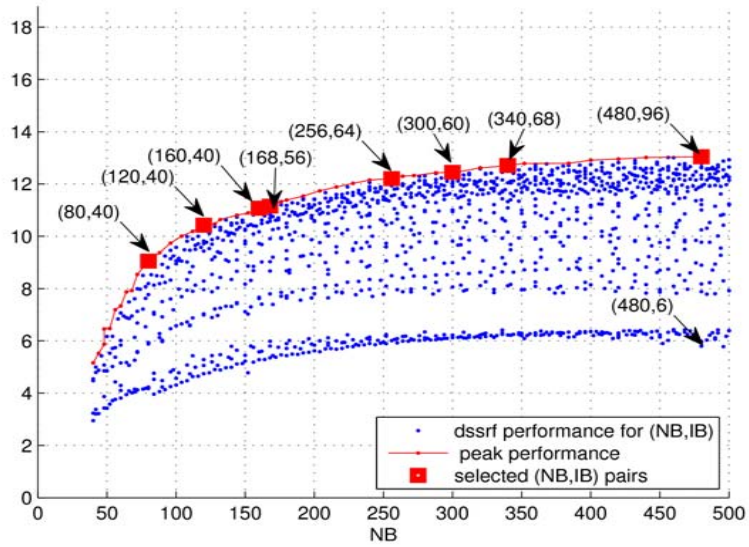
[1st model-driven to limit the search space for a 2nd empirical part]
[Another aspect is adaptivity – to treat cases where tuning can not be restricted to optimizations at design, installation, or compile time]

Pruning the Search Space

- **Time serial core kernels (dgemm, dssrfb, dssssm).**



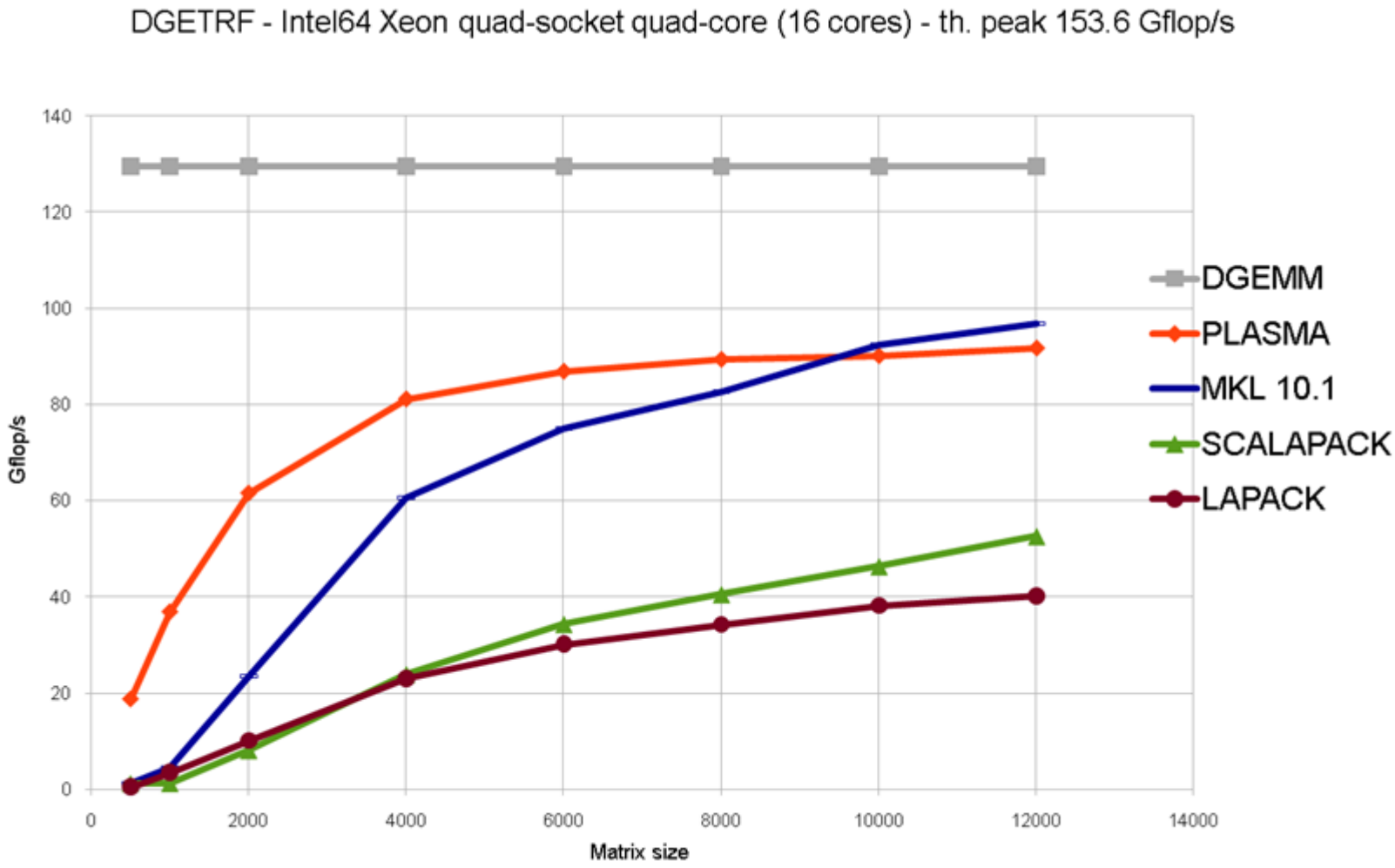
Intel 64 – dgemm



Power 6 – dssrfb

- **Pick up the 'best' NB/IB samples (pruning);**
- **Select one per matrix size and number of cores.**

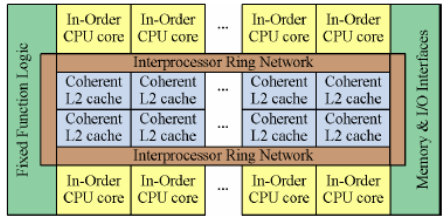
DGETRF - Intel64 - 16 cores



Future Computer Systems

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

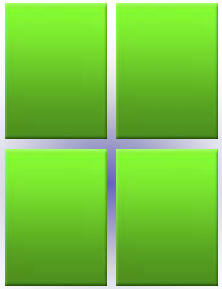
- Next generation more integrated
- Intel's Larrabee in 2010
  - 8,16,32,or 64 x86 cores
- AMD's Fusion in 2011
  - Multicore with embedded graphics ATI
- Nvidia's plans?



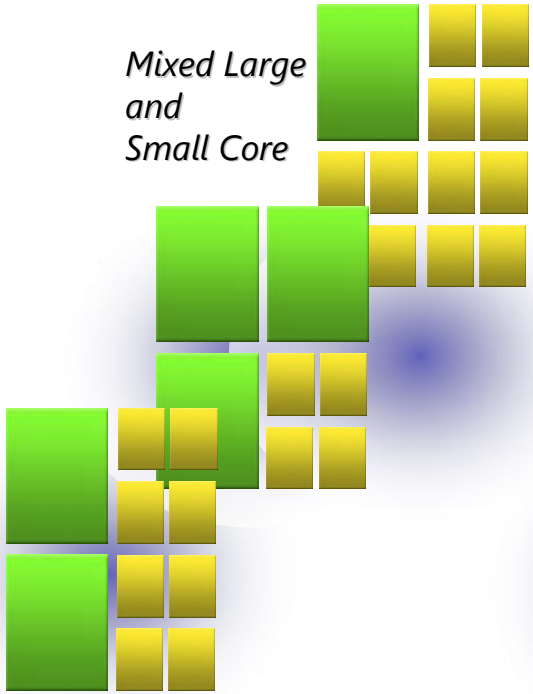
Intel Larrabee

What's Next?

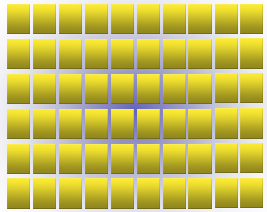
All Large Core



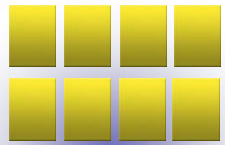
Mixed Large and Small Core



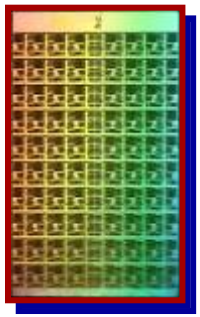
Many Small Cores



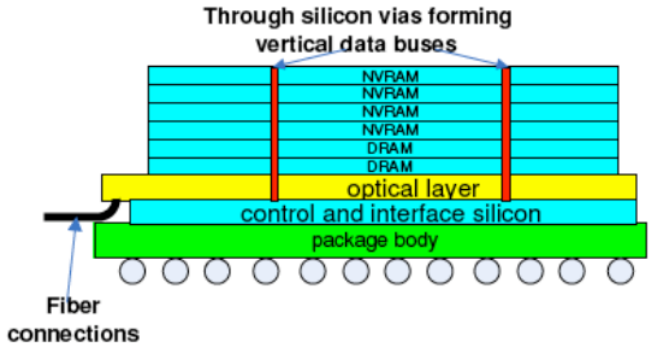
All Small Core



Many Floating-Point Cores



+ 3D Stacked Memory



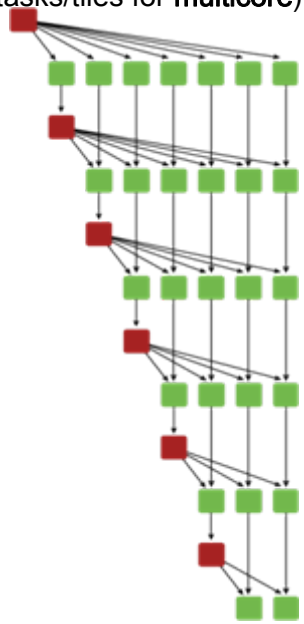
Different Classes of Chips

- Home
- Games / Graphics
- Business
- Scientific

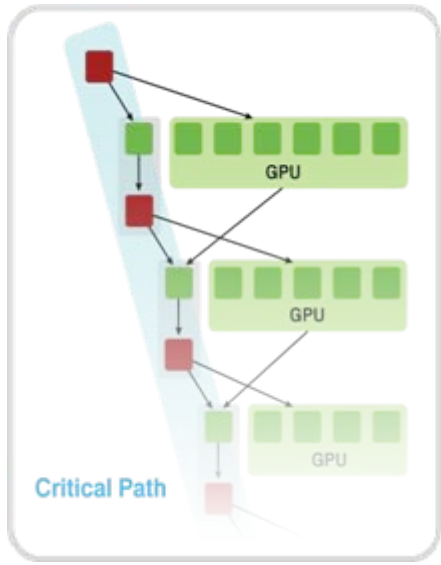
Hybrid Computing

- **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**)



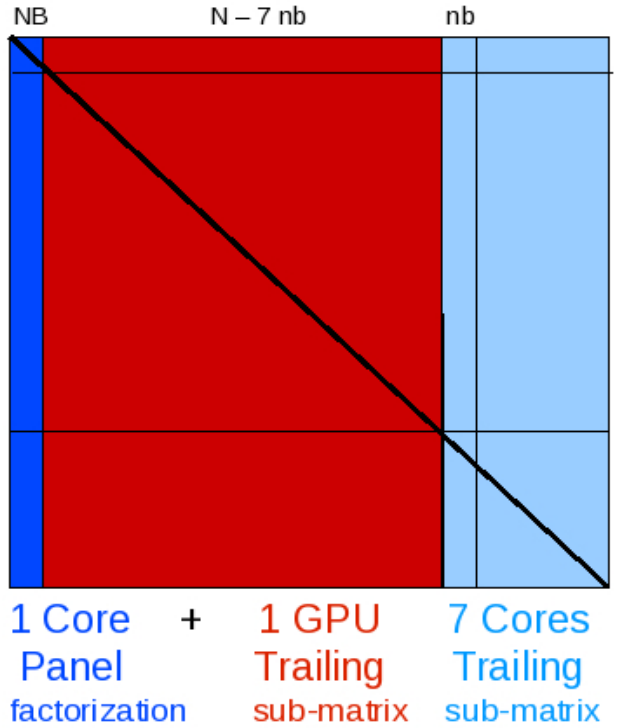
Current hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)



- **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**

Current Work: MAGMA

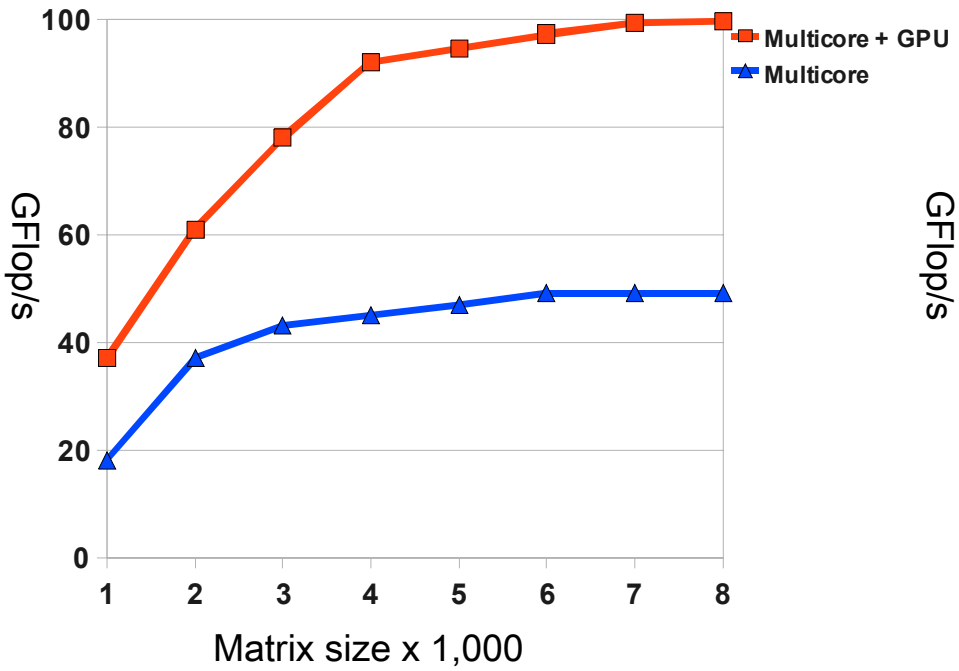
- **Algorithms (in particular LU) for Multicore + GPU systems**
- **Challenges**
  - **How to split the computation**
  - **Software development**
  - **Tuning**



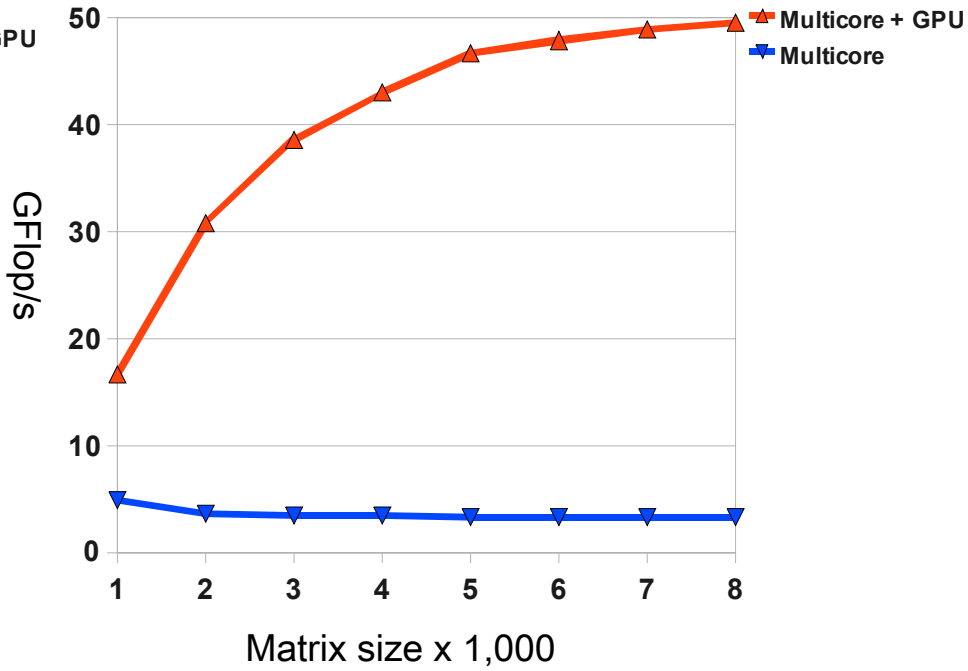
Work splitting
(for single GPU + 8 cores host)

Performance [in double precision]

One-sided (LU)



Two-sided (Hessenberg)



- **Needed tuned parameters and tuned DGEMM for "rectangular" matrices**

GPU : GeForce GTX 280
(240 Cores @ 1.30 GHz)
Multicore : Intel Xeon
(2x4 Cores @ 2.33 GHz)

Exascale Computing

Google: exascale computing study

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

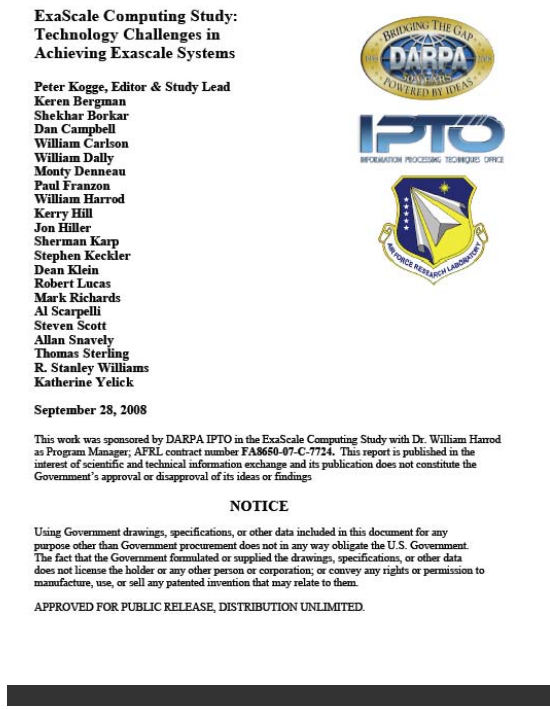
September 28, 2008

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

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

Google: exascale computing study

Five Important Features to Consider When Computing at Scale

- Effective Use of Many-Core and Hybrid architectures
  - Dynamic Data Driven Execution
  - Block Data Layout
- Exploiting Mixed Precision in the Algorithms
  - Single Precision is 2X faster than Double Precision
  - With GP-GPUs 10x
- Self Adapting / Auto Tuning of Software
  - Too hard to do by hand
- Fault Tolerant Algorithms
  - With 1,000,000's of cores things will fail
- Communication Avoiding Algorithms
  - For dense computations from O(n *log* p) to O(*log* p) communications
  - GMRES s-step compute (x, Ax, A^2x, … A^sx)

Conclusions

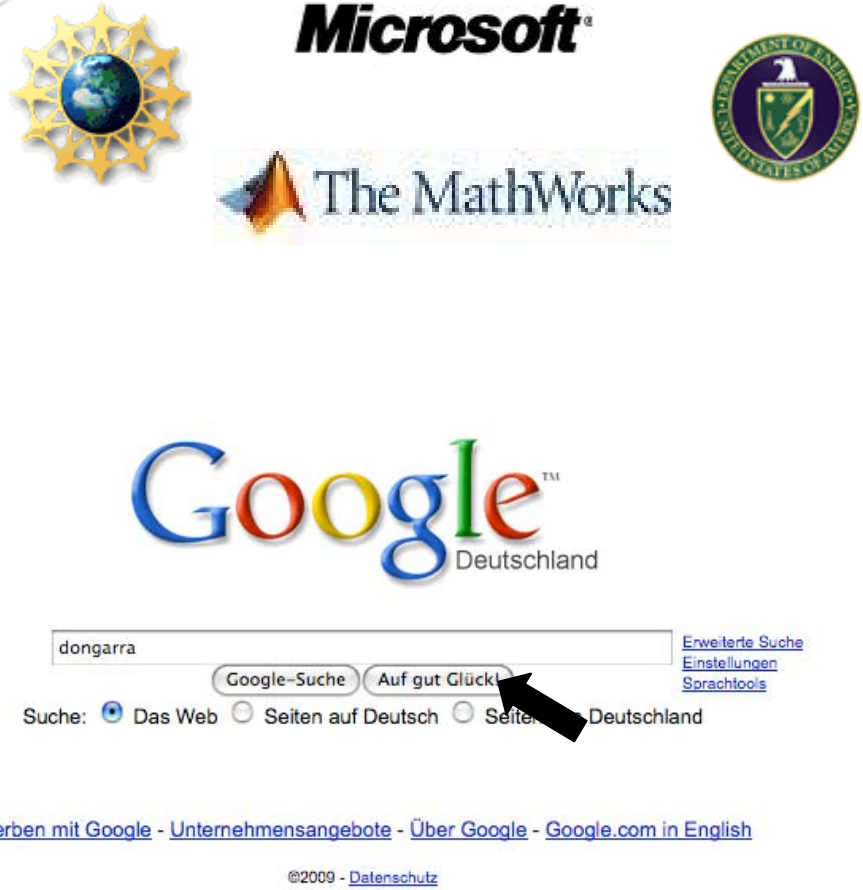
- For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware.
- This strategy needs to be rebalanced - barriers to progress are increasingly on the software side.
- Moreover, the return on investment is more favorable to software.
  - Hardware has 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 ICL group at Tennessee

PLASMA Parallel Linear Algebra Software for Multicore Architectures

MAGMA Matrix Algebra on GPU and Multicore Architectures



Contact Jack Dongarra

If you are wondering what's beyond ExaFlops

Mega, Giga, Tera, Peta, Exa, Zetta …

10^{3}	kilo
10^{6}	mega
10^{9}	giga
10^{12}	tera
10^{15}	peta
10^{18}	exa
10^{21}	zetta

10^{24}	yotta
10^{27}	xona
10^{30}	weka
10^{33}	vunda
10^{36}	uda
10^{39}	treda
10^{42}	sorta
10^{45}	rinta
10^{48}	quexa
10^{51}	pepta
10^{54}	ocha
10^{57}	nenaN
10^{60}	minga
10^{63}	luma