PLASMA and Scheduling Dense Linear Algebra on Multicore Chips

Jack Dongarra

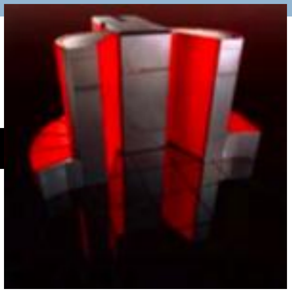
University of Tennessee
Oak Ridge National Laboratory
University of Manchester

Argonne ~1976 (IFIPS WG 2.5)

Gatlinburg Meeting 1981, Oxford

Looking at the Gordon Bell Prize

(Recognize outstanding achievement in high-performance computing applic
and encourage development of parallel processing)

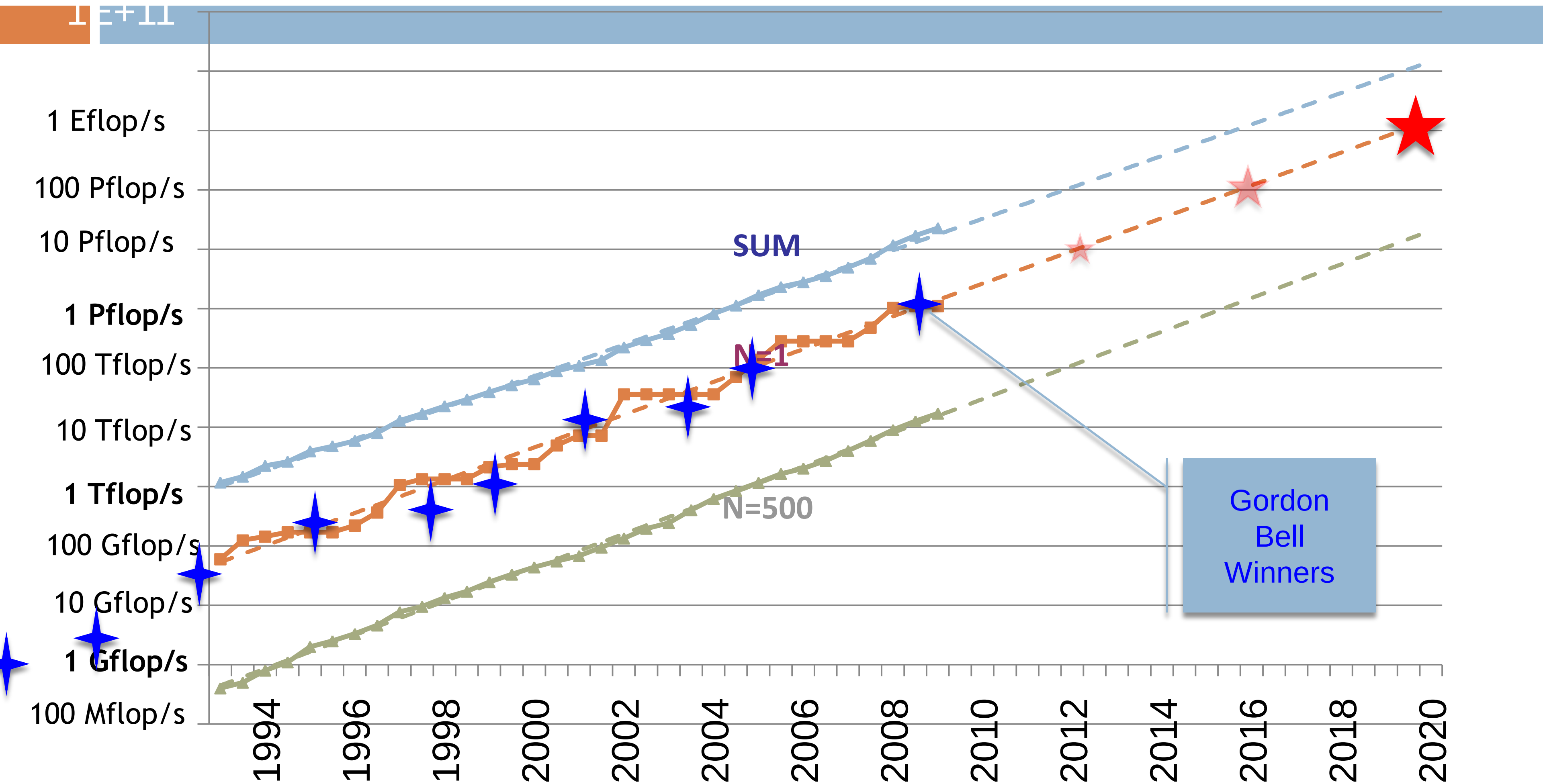


- 1 GFlop/s; 1988; Cray Y-MP; 8 Processor
  - Static finite element analysis
- 1 TFlop/s; 1998; Cray T3E; 1024 Processors
  - Modeling of metallic magnet atoms, using a variation of the locally self-consistent multip scattering method.

- 1 PFlop/s; 2008; Cray XT5; 1.5x10^5 Processors
  - Superconductive materials

- 1 EFlop/s; ~2018; ?; 1x10^7 Processors (10^9 threads)

Performance Development in Top500



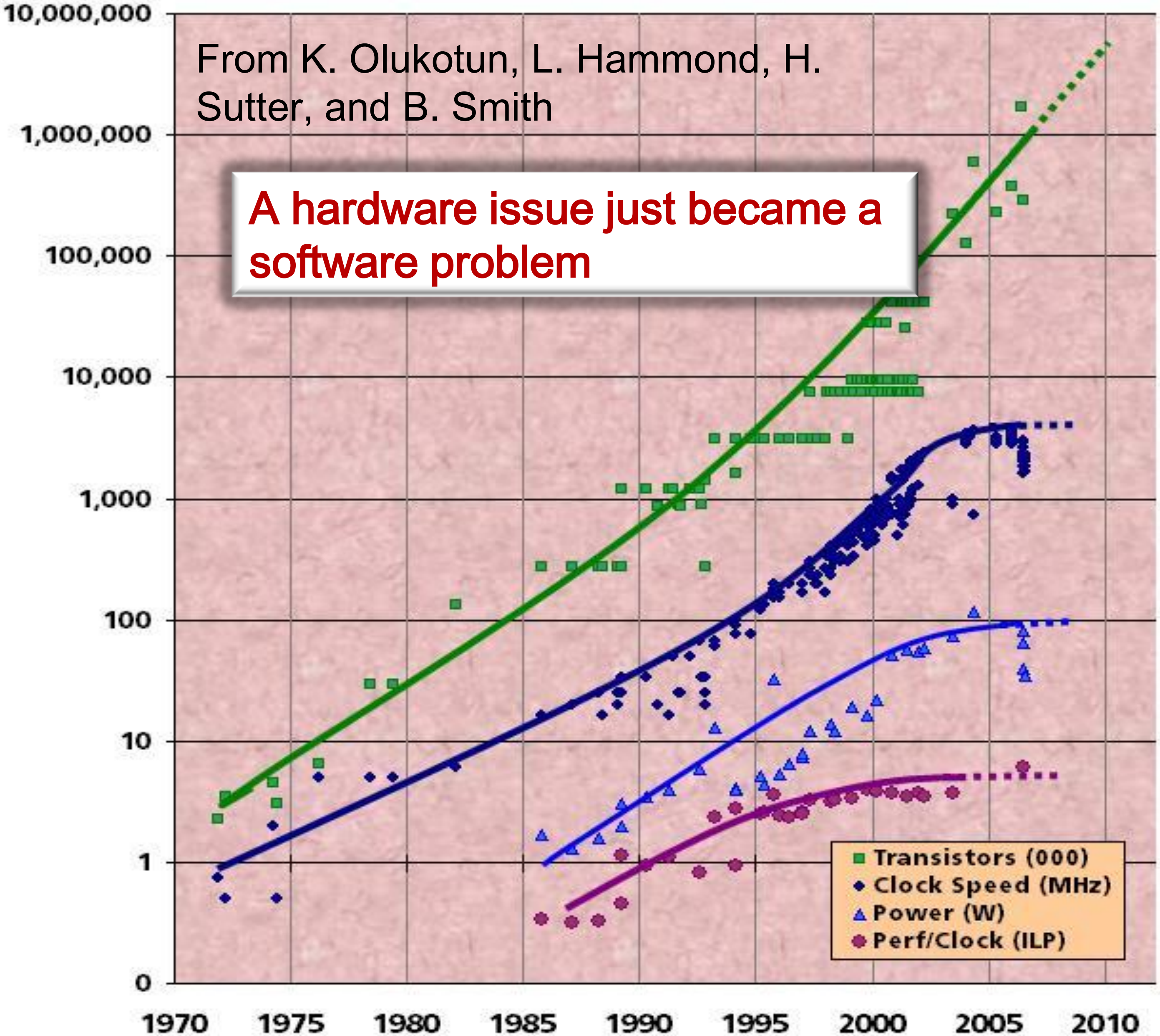
33rd List: The TOP10 (core overloaded term)

Rank	Site	Computer	Country	Procs (Cores)	Rmax [Tflops]	% of Peak
1	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenter QS22/LS21	USA	129,600	1,105	76
2	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5 QC 2.3 GHz	USA	150,152	1,059	77
3	Forschungszentrum Juelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	825	82
4	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	51,200	480	79
5	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServer Blue Gene Solution	USA	212,992	478	80
6	NSF NICS/U of Tennessee	Kraken / Cray Cray XT5 QC 2.3 GHz	USA	66,000	463	76
7	DOE / OS Argonne Nat Lab	Intrepid / IBM Blue Gene/P Solution	USA	163,840	458	82
8	NSF TACC/U. of Texas	Ranger / Sun SunBlade x6420	USA	62,976	433	75
9	DOE / NNSA Lawrence Livermore NL	Dawn / IBM Blue Gene/P Solution	USA	147,456	415	83
10	Forschungszentrum Juelich (FZJ)	JUROPA /Sun - Bull SA NovaScale /Sun Blade	Germany	26,304	274	89

Numerical Linear Algebra Library

- Interested in developing numerical library for the fastest, largest computer platforms for scientific computing.
- Today we have machines with 100K of processors (cores) going to 1M in the next generation
- A number of important issues must be addressed in the design of algorithms and software.

Something's Happening Here…



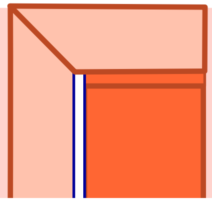
- In the "old days" it was: each year processors would become faster
- Today the clock speed is fixed or getting slower
- Things are still doubling every 18 -24 months
- Moore's Law reinterpretated.
  - Number of cores double every 18-24 months

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

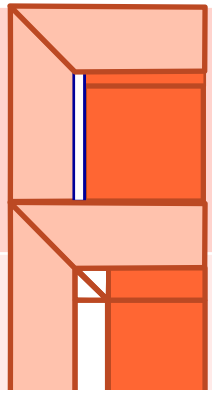
A New Generation of Software:

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

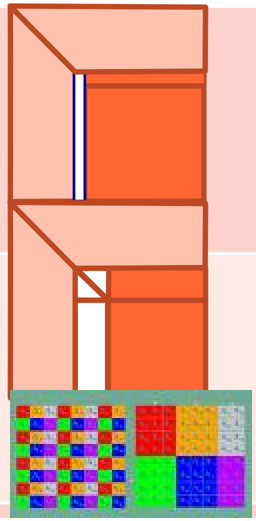
A New Generation of Software:

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		Rely on - Level-3 BLAS

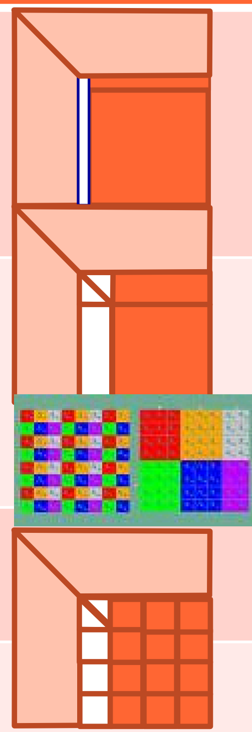
A New Generation of Software:

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

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

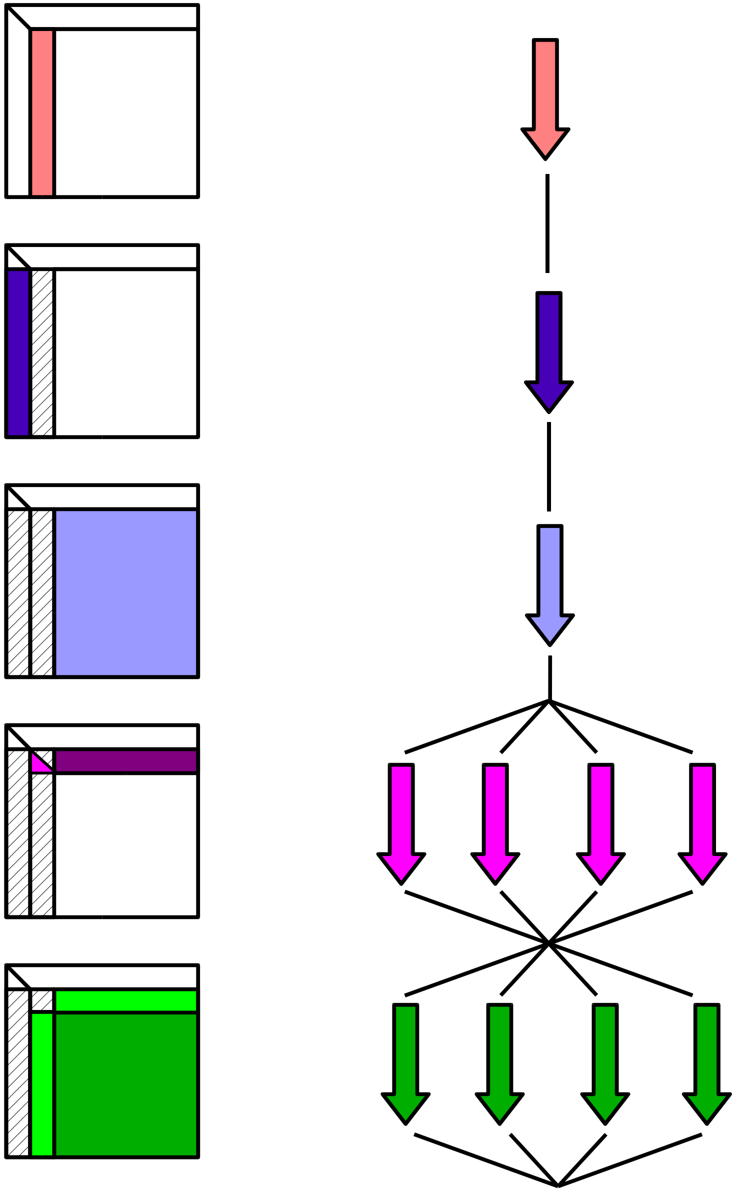
DGETF2 (Factor a panel) — LAPACK

DLSWP (Backward swap) — LAPACK

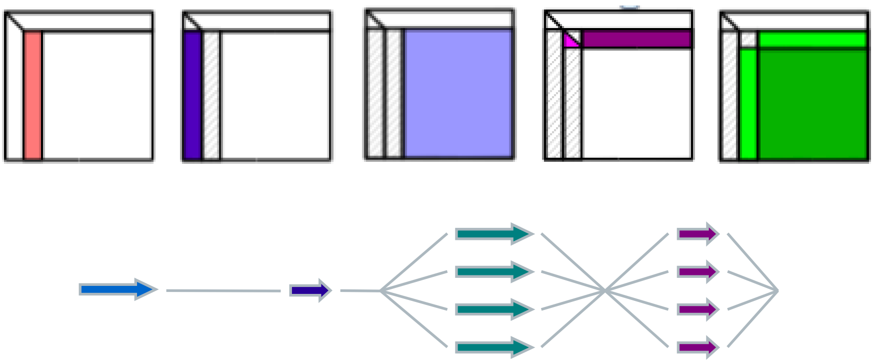
DLSWP (Forward swap) — LAPACK

DTRSM (Triangular solve) — **BLAS**

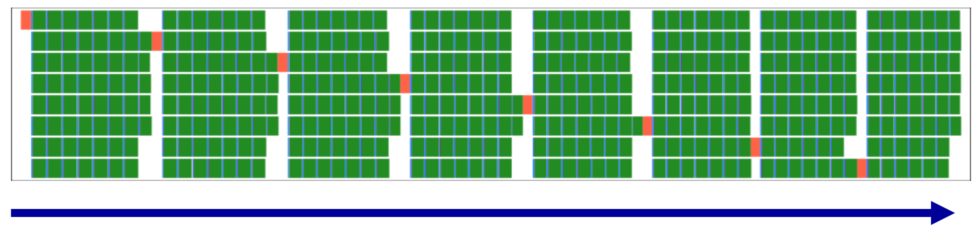
DGEMM (Matrix multiply) — **BLAS**



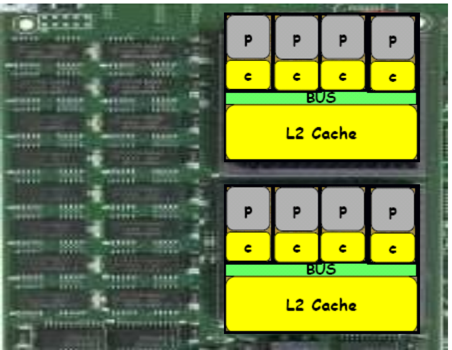
Fork-Join vs. Dynamic Execution



Fork-Join – parallel BLAS

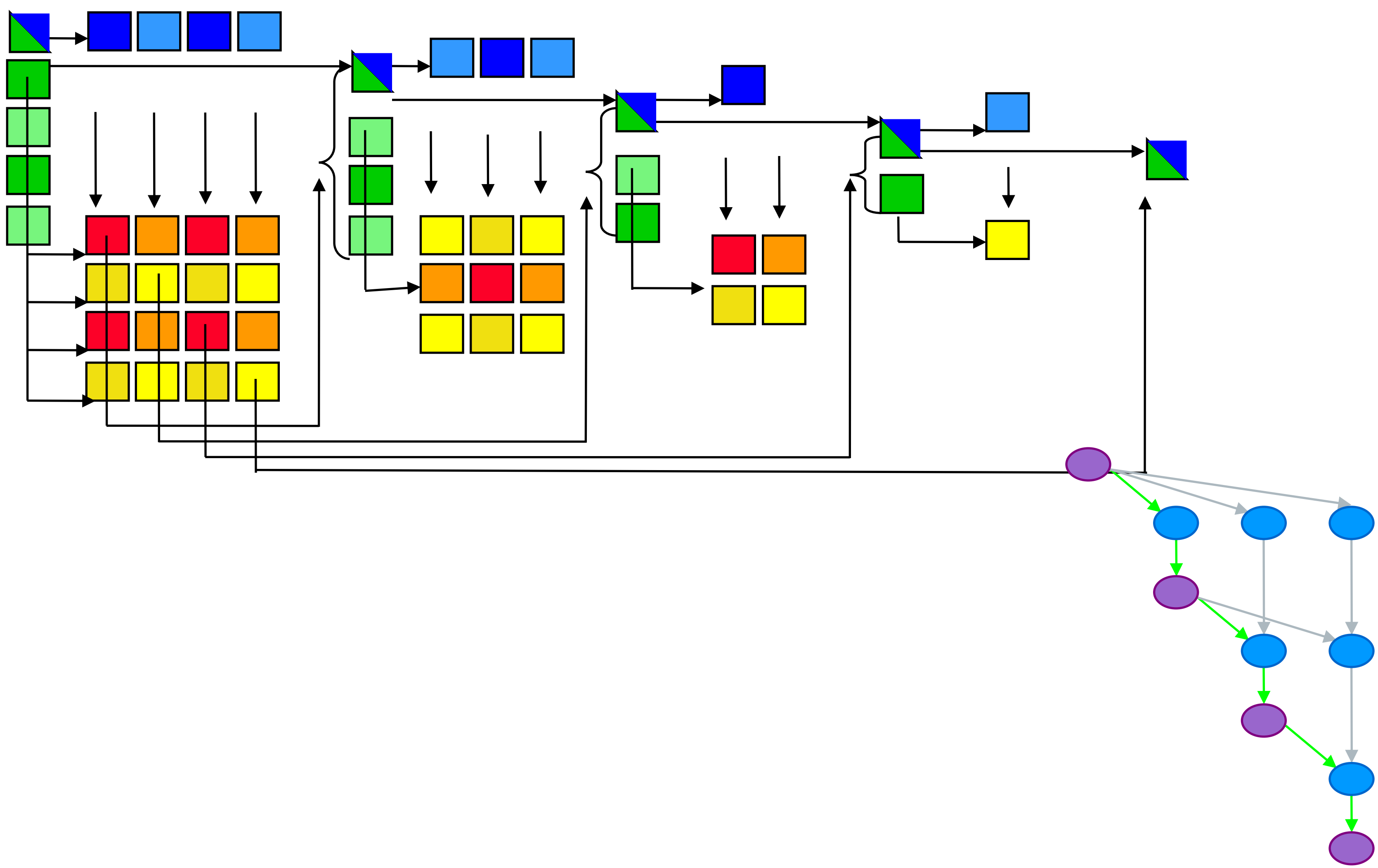


Time

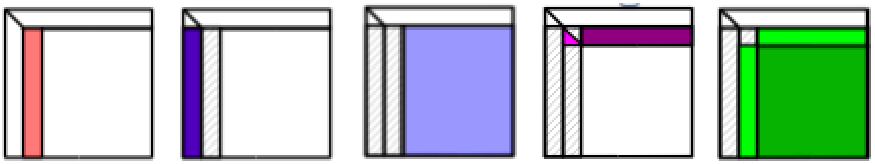


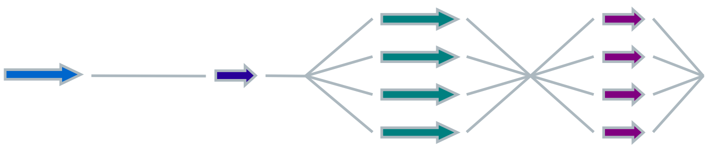
Experiments on
Intel's Quad Core Clovertown
with 2 Sockets w/ 8 Treads

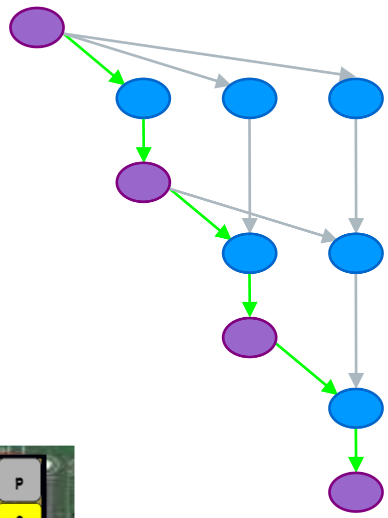
Parallel Tasks in LU



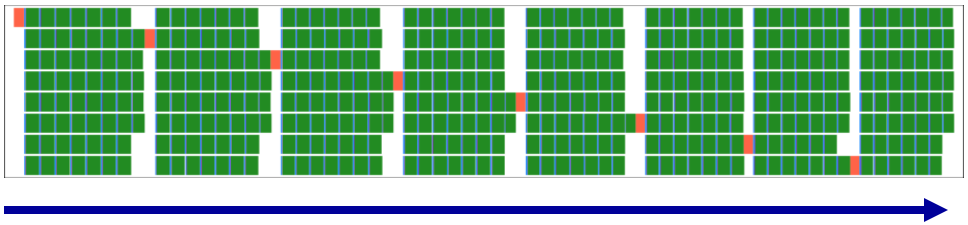
Fork-Join vs. Dynamic Execution





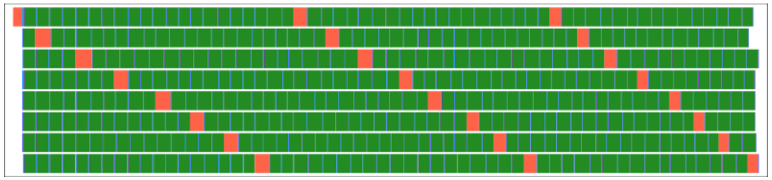


Fork-Join – parallel BLAS

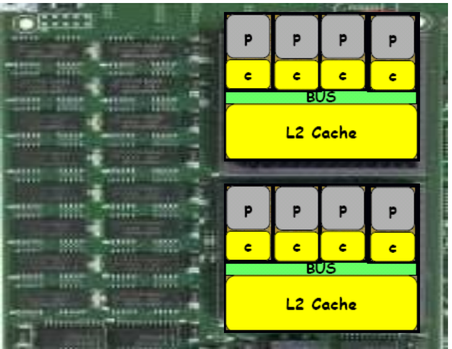


Time

DAG-based – dynamic scheduling

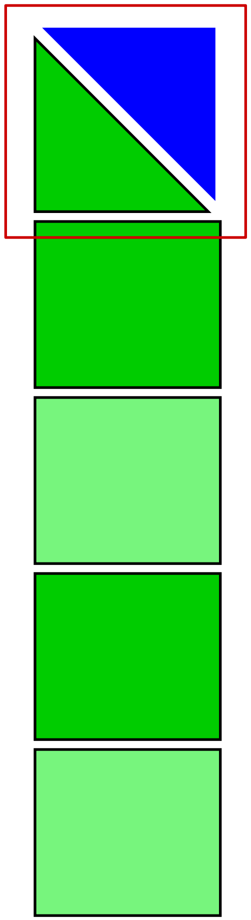


Time saved



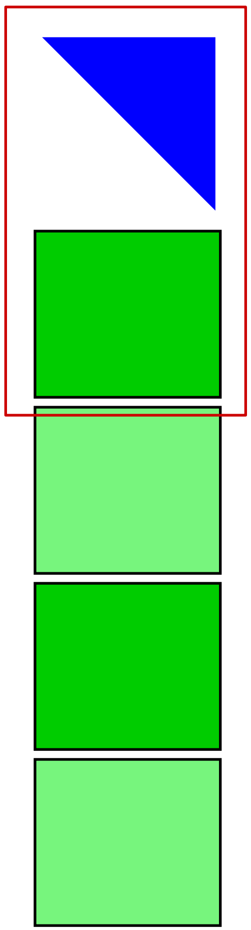
Experiments on
Intel's Quad Core Clovertown
with 2 Sockets w/ 8 Treads

Parallel Tasks in LU



Step 1: LU of block 1,1 (w/partial pivoting)

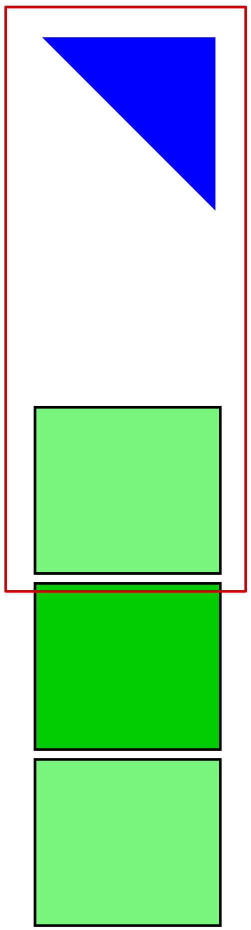
Parallel Tasks in LU



Step 1: LU of block 1,1 (w/partial pivoting)

Step 2: Use $U_{1,1}$ to zero $A_{1,2}$ (w/partial pivoting

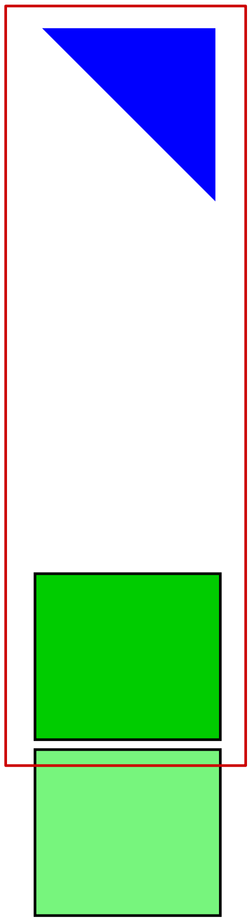
Parallel Tasks in LU



Step 1: LU of block 1,1 (w/partial pivoting)

Step 2: Use $U_{1,1}$ to zero $A_{1,2}$ (w/partial pivoting

Parallel Tasks in LU



Step 1: LU of block 1,1 (w/partial pivoting)

Step 2: Use $U_{1,1}$ to zero $A_{1,2}$ (w/partial pivoting

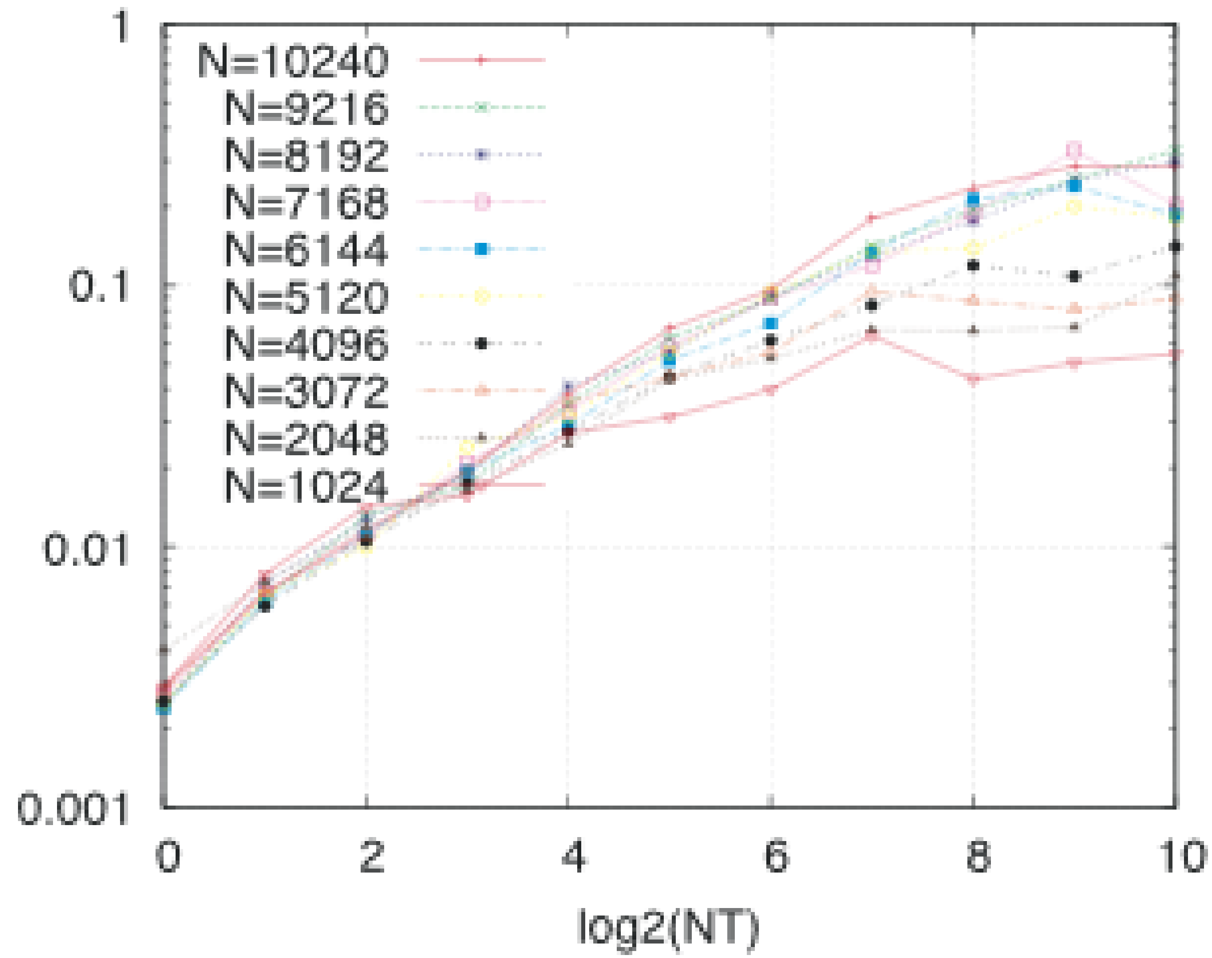
Step3: Use $U_{1,1}$ to zero $A_{1,3}$ (w/partial pivoting

.

.

.

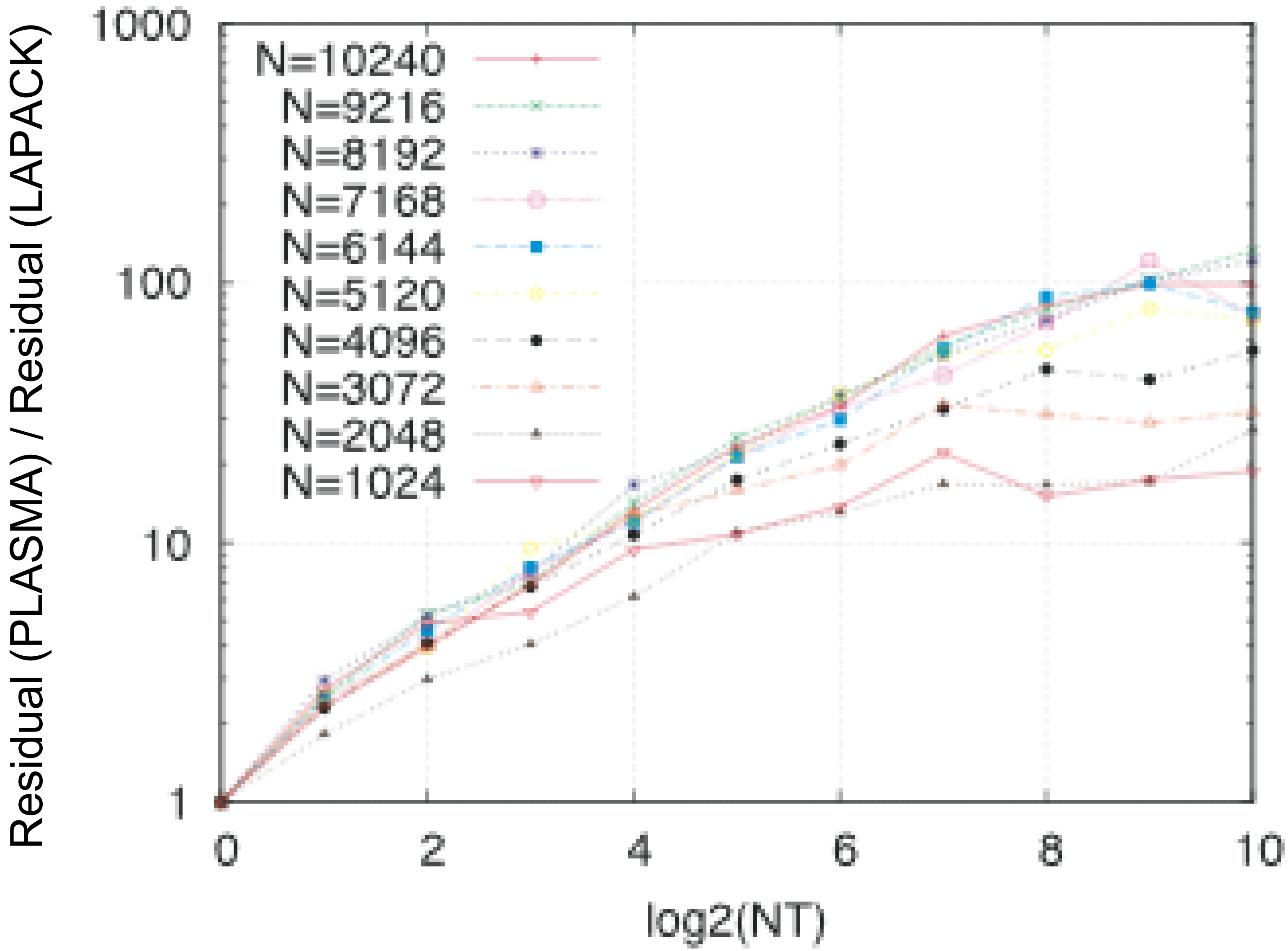
Residual from PLASMA's Tiled LU



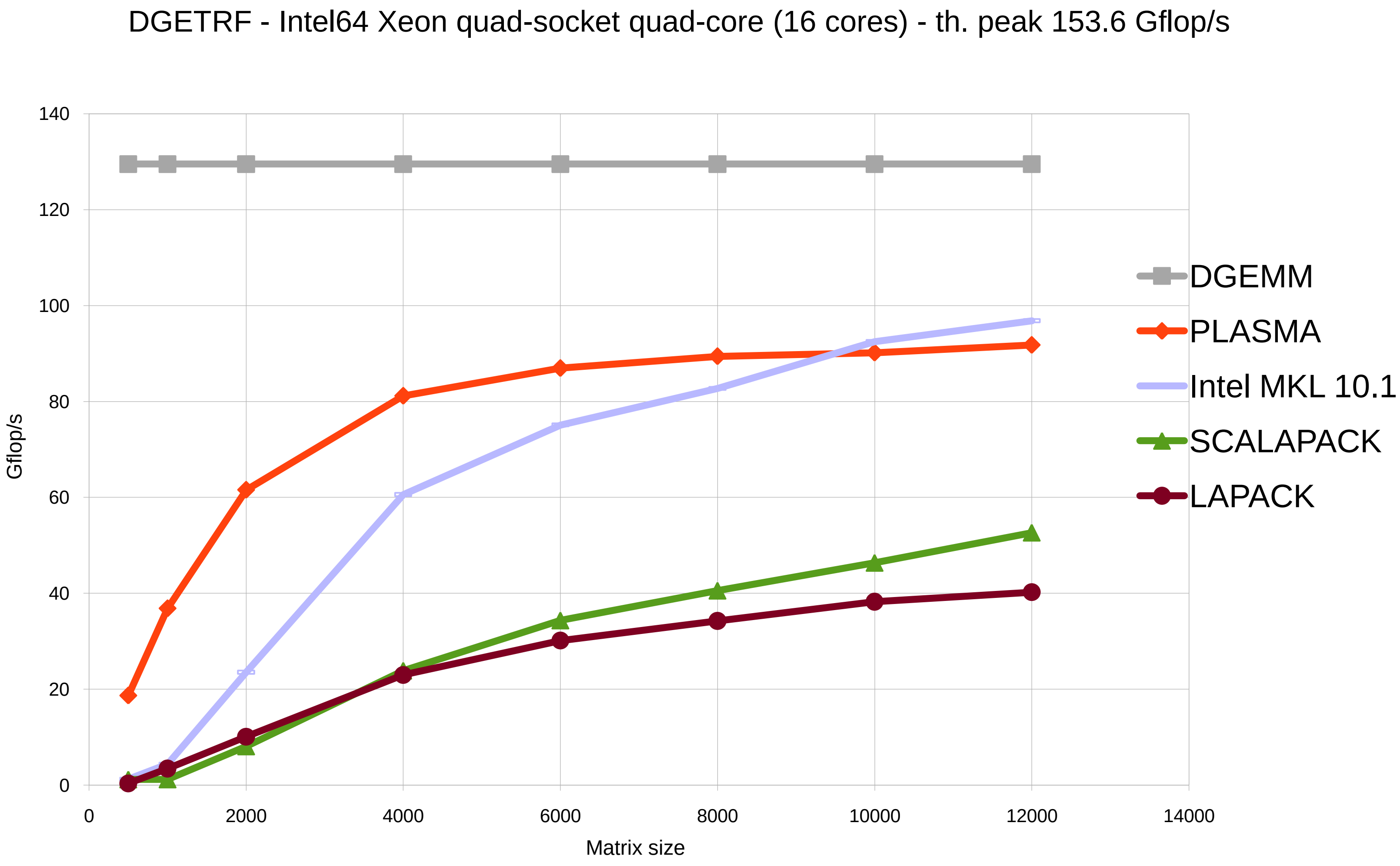
Random Matrices

NT (Number of Tiles)

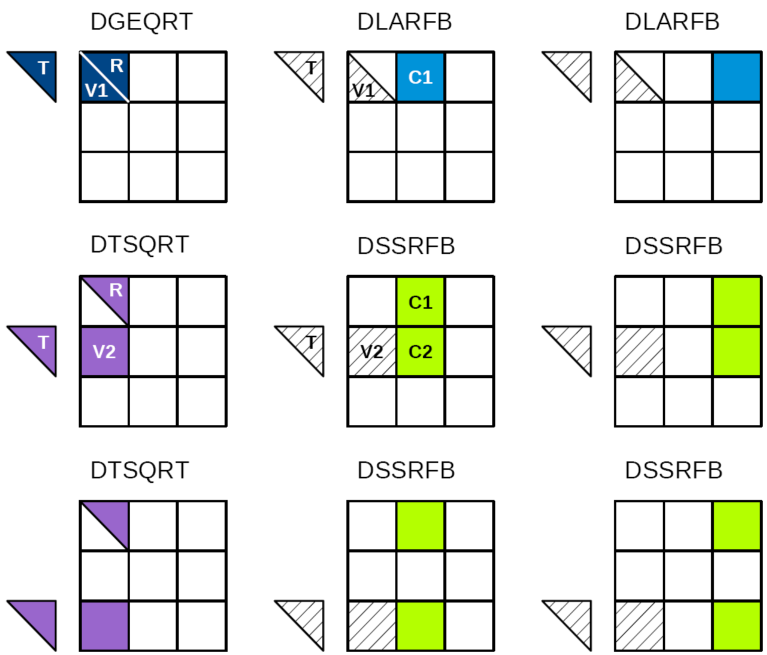
Residual Comparison with LAPACK

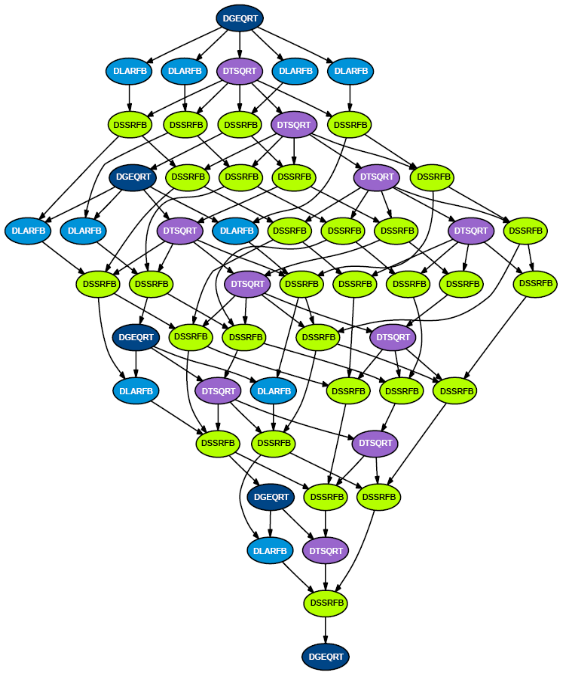


DGETRF - Intel64 - 16 cores



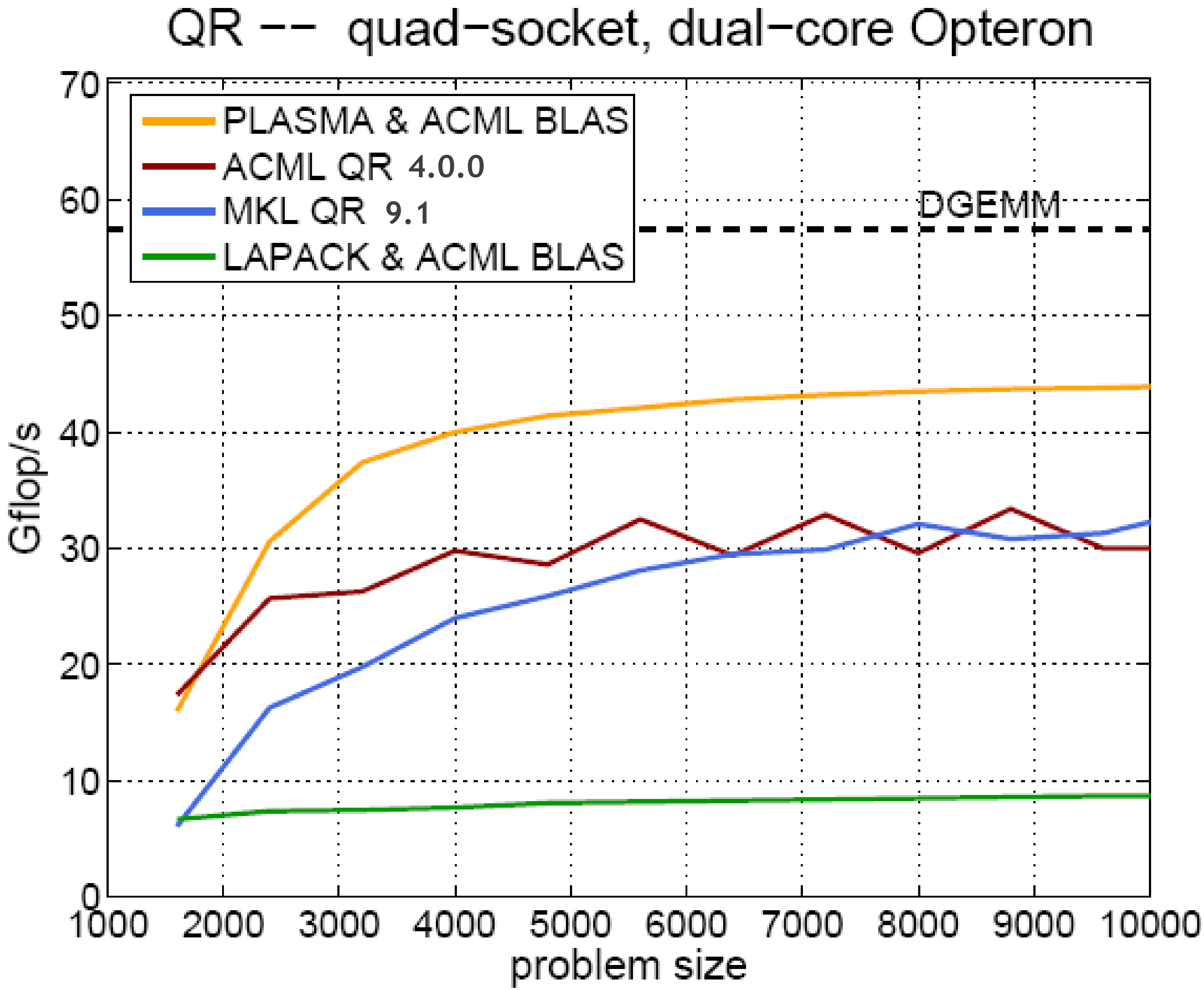
Tile QR Algorithms





```
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
- DAG organization


QR -- quad-socket, dual-core Opteron
PLASMA & ACML BLAS
ACML QR 4.0.0
MKL QR 9.1
LAPACK & ACML BLAS
DGEMM
Gflop/s
0
10
20
30
40
50
60
70
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
problem size

PLASMA (Redesign LAPACK/ScaLAPACK)

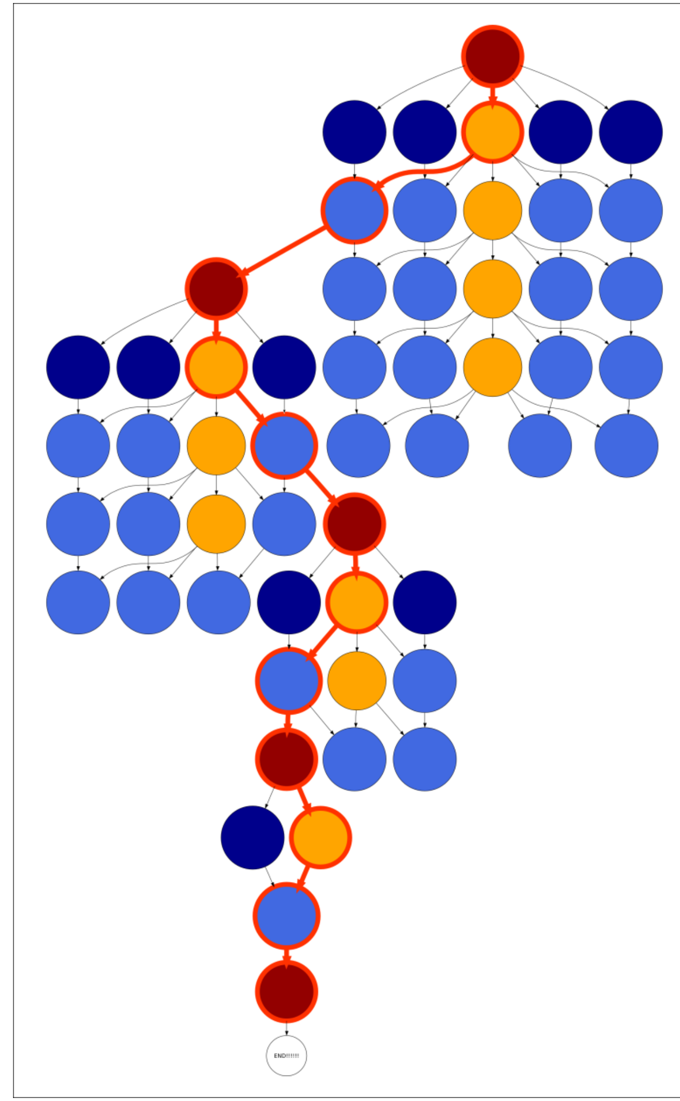
Parallel Linear Algebra Software for Multicore Architectures

- 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

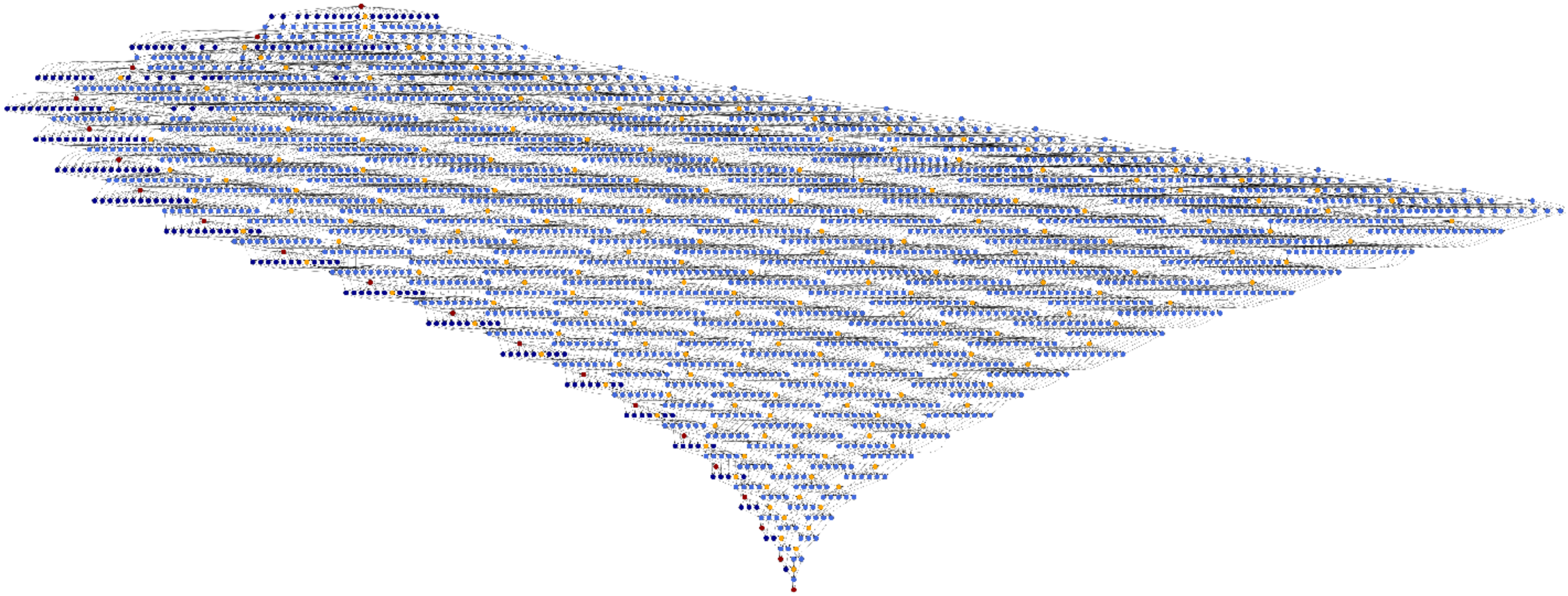
If We Had A Small Matrix Problem

- 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



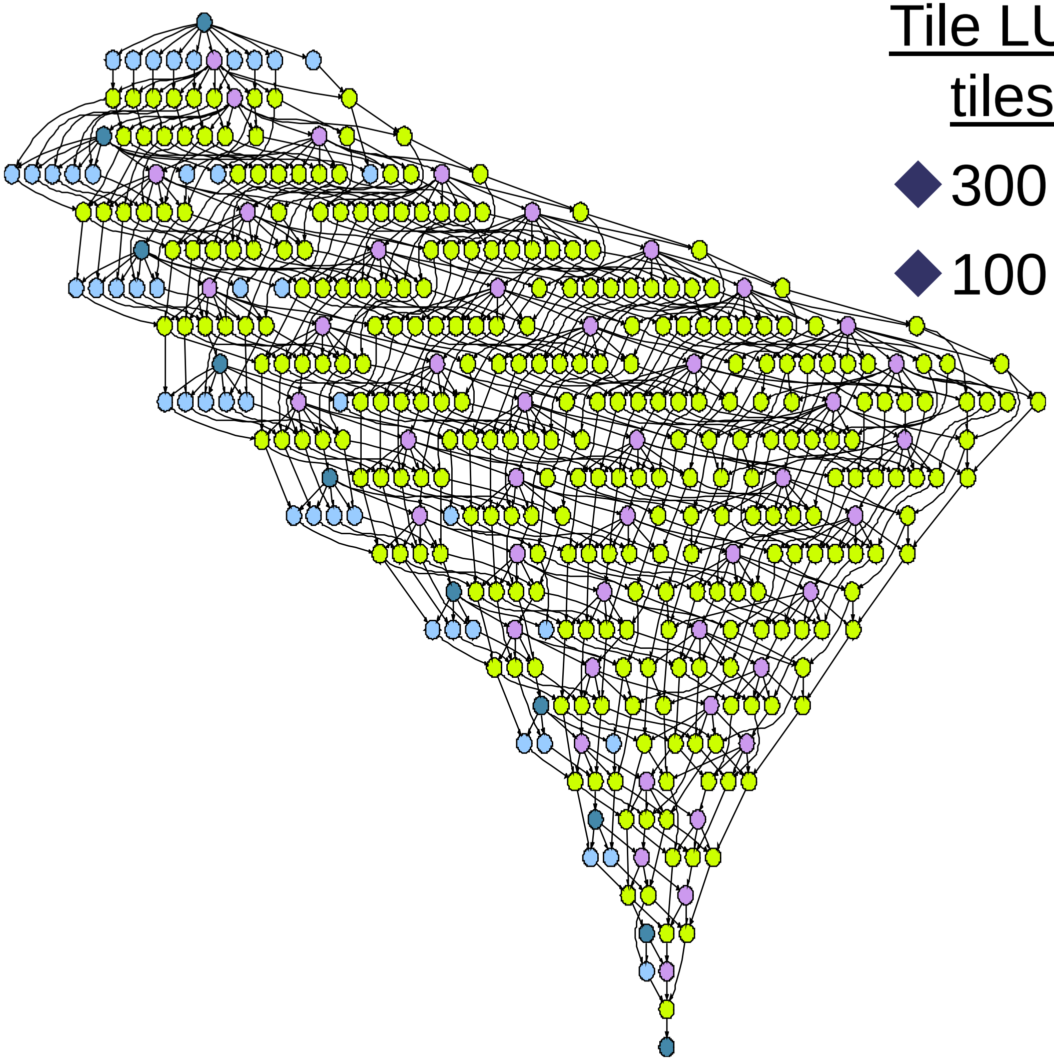
The DAGs are Large

- **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**

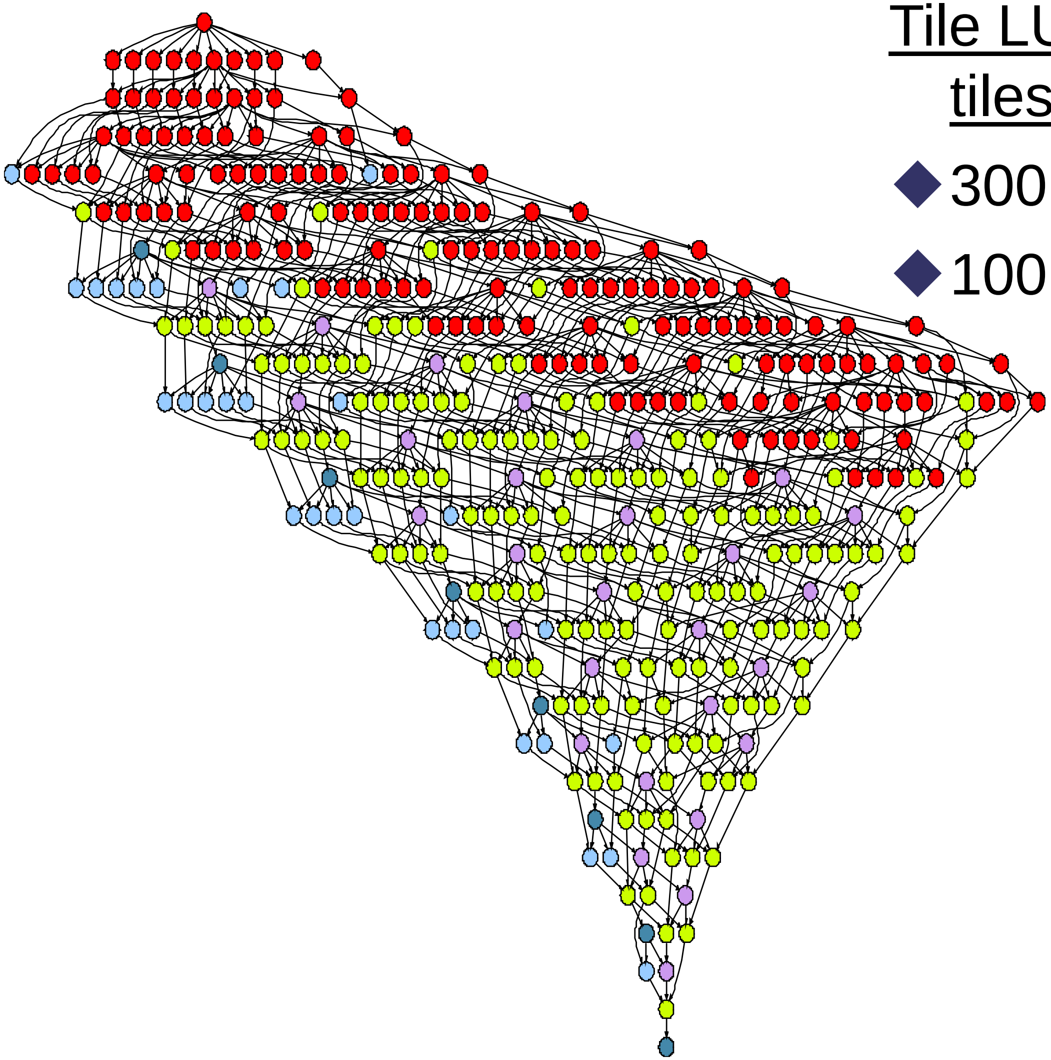
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

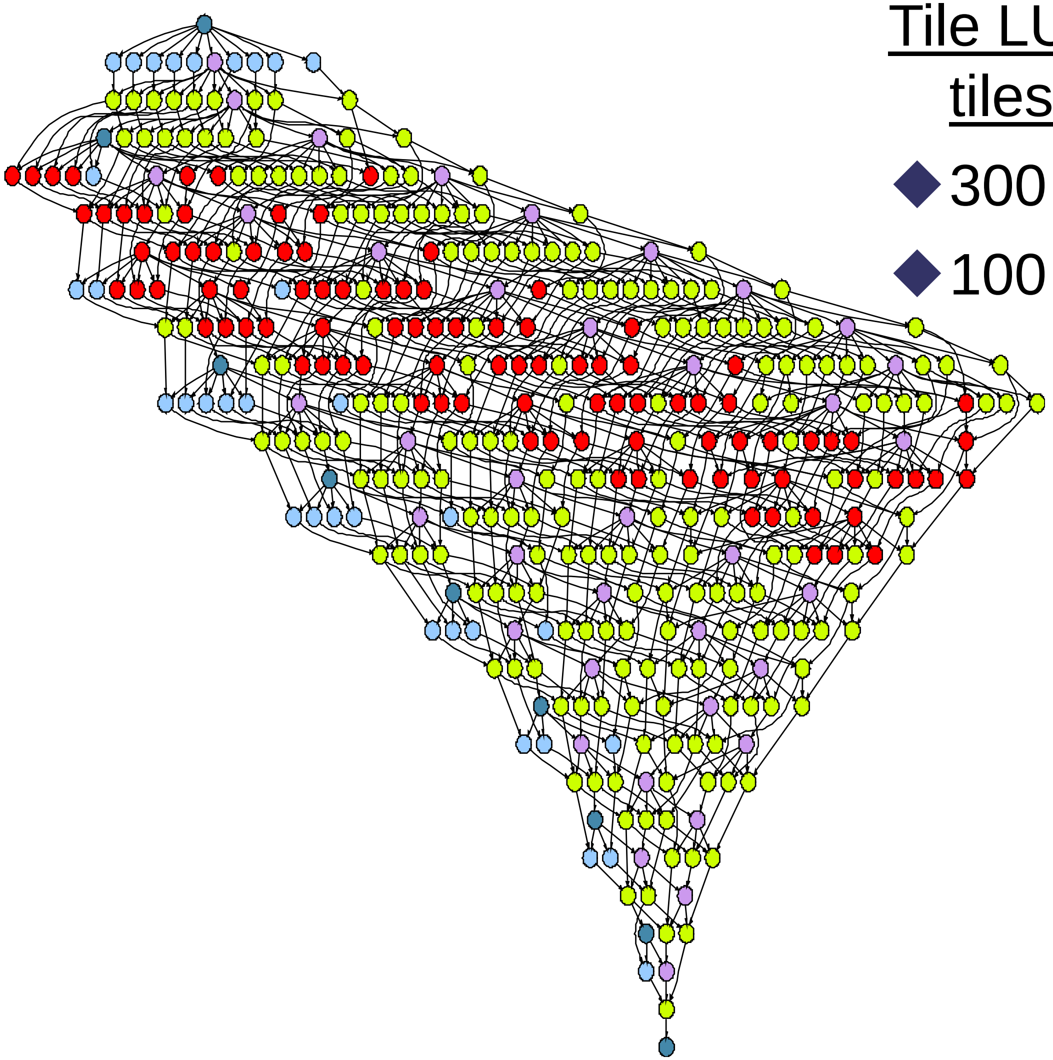
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

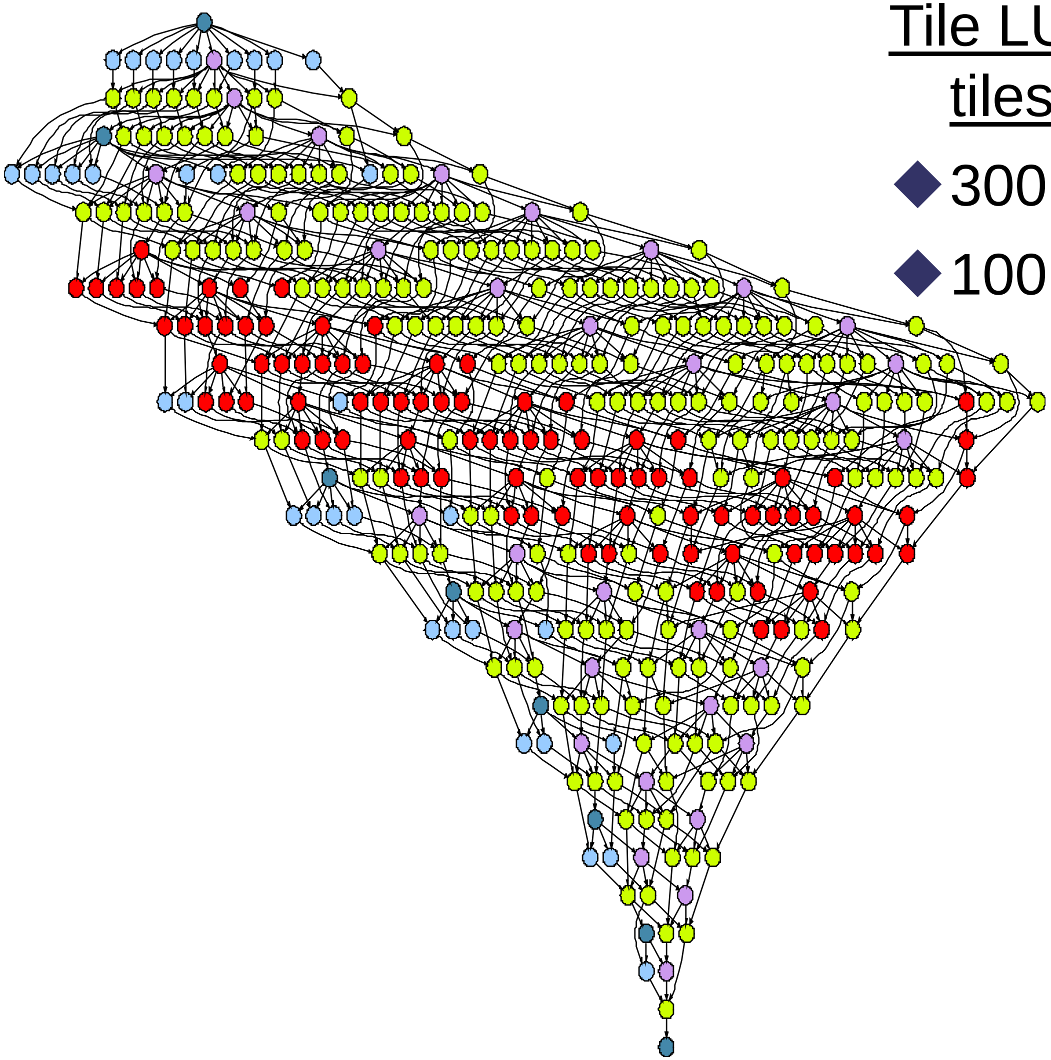
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

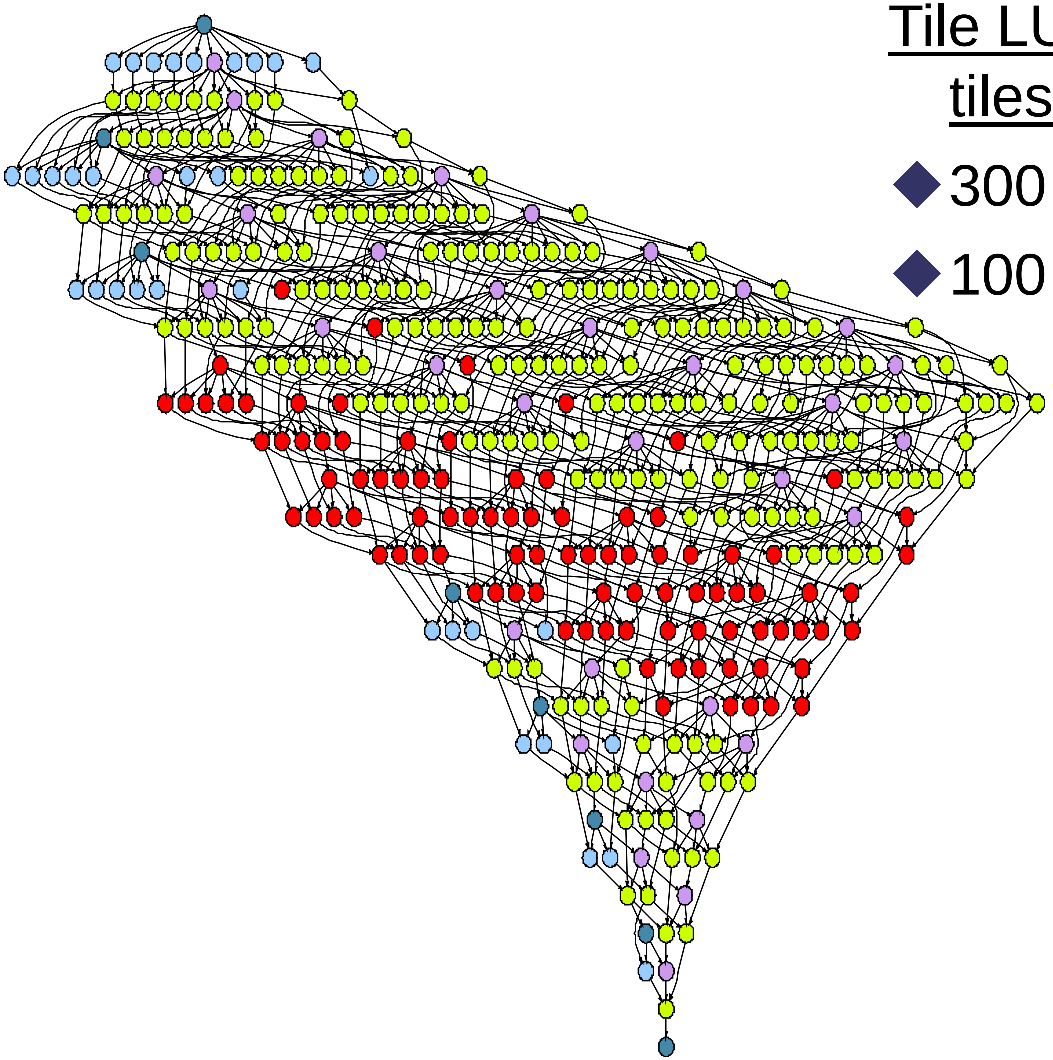
Execution of the DAG by a Sliding Window



<u>Tile LU factorization 10x10 tiles</u>

- 300 tasks total
- 100 task window

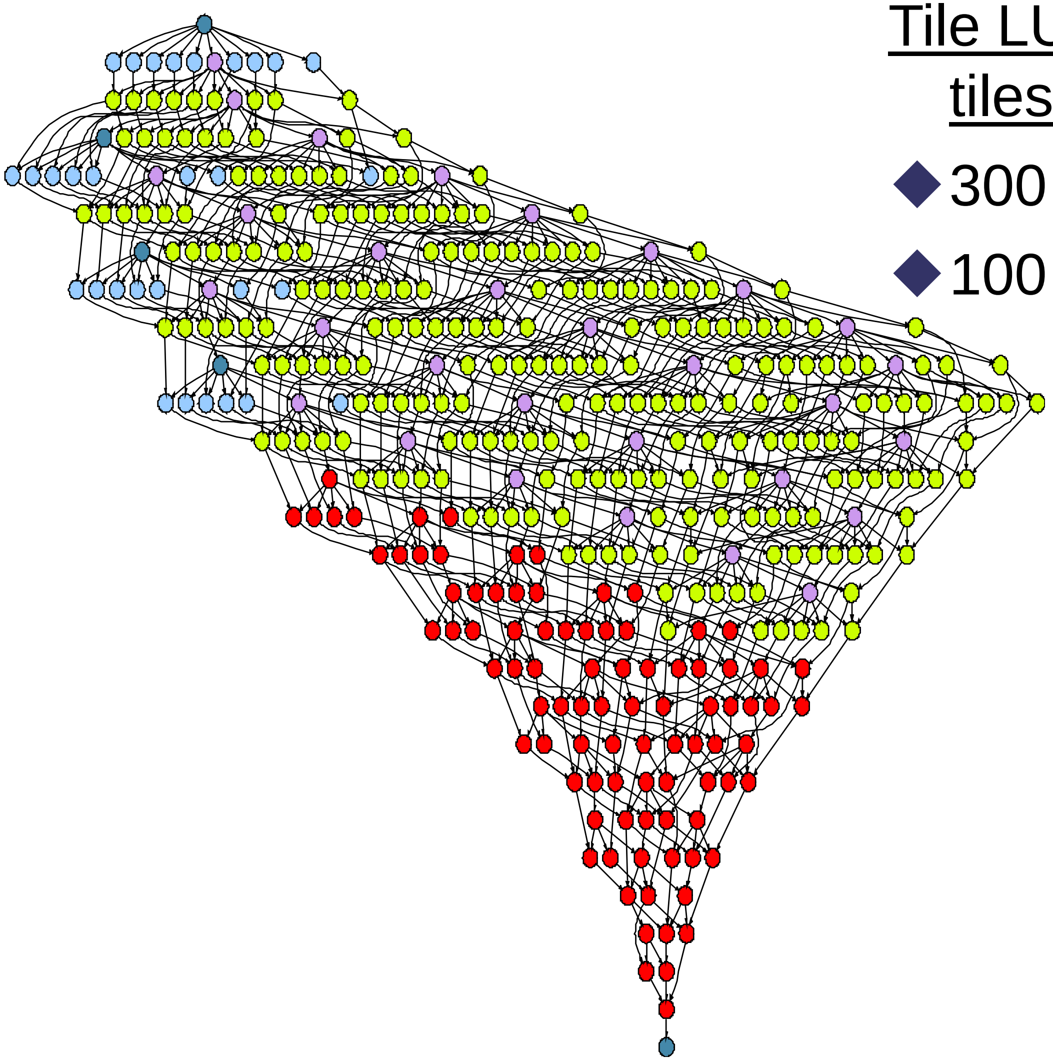
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

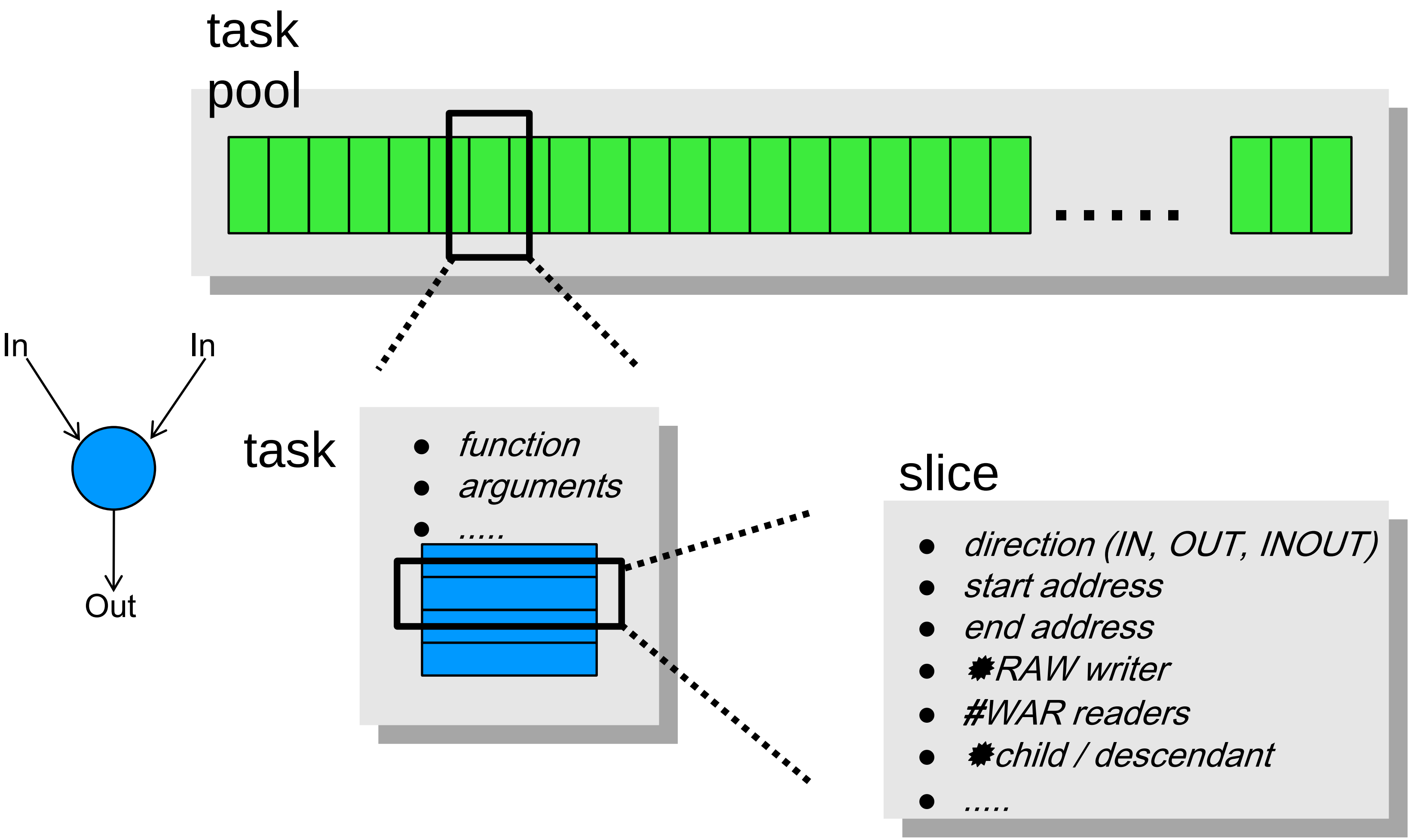
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

PLASMA Dynamic Task Scheduler



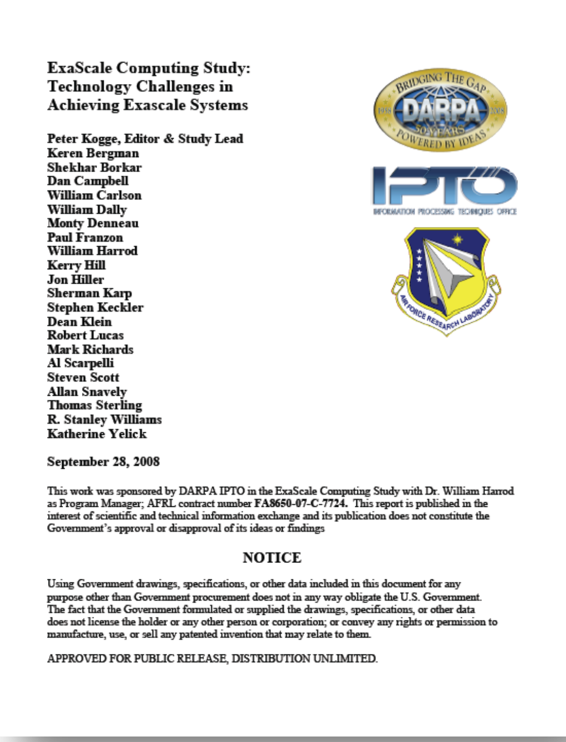
- task – a unit of scheduling (quantum of work)
- slice – a unit of dependency resolution (quantum of data)
- Current version uses one core to manage the task pool

PLASMA Today

- http://icl.cs.utk.edu/plasma/
  - **Linear general system – LU – *tile pairwise* pivoting**
  - **Linear SPD system – Cholesky factorization**
  - **Overdetermined system – QR factorization**
  - **Underdetermined system – LQ factorization**
  - **Iterative refinement for s/d and c/z**
  - **Single, double, complex, and double complex arithmetic.**
  - **LAPACK look and feel, testing, timing, examples**
    - **Shared memory**
  - **Next two sided factorization, accelerators, message passing**

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

Five Important Features to Consider When Developing Software 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)

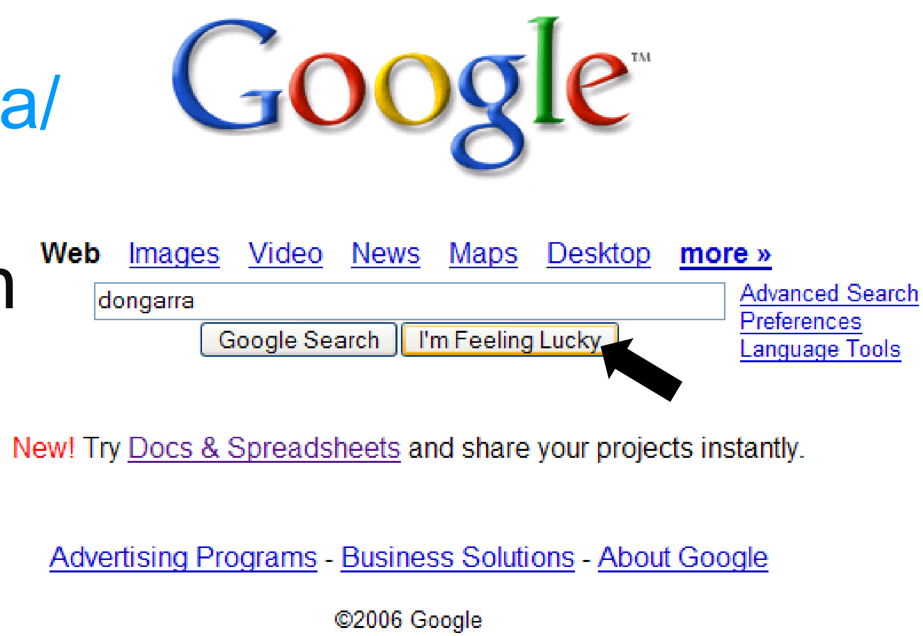
Collaborators / Support

Joint work with Jim Demmels' group at Berkeley

PLASMA Parallel Linear Algebra Software for Multicore Architectures

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

MAGMA Matrix Algebra on GPU and Multicore Architectures



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