18th Annual Meeting
Saturday 12th September 2009, 10:00-17:00

Linear Algebra Libraries for Peta and Exascale Systems

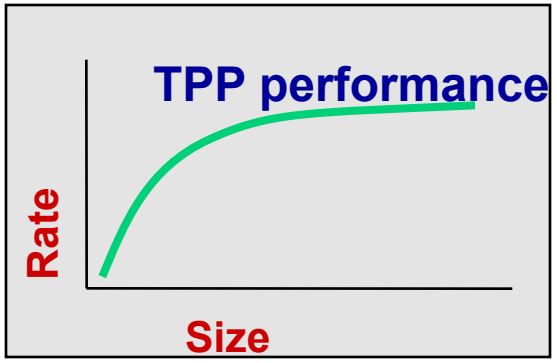
Jack Dongarra

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

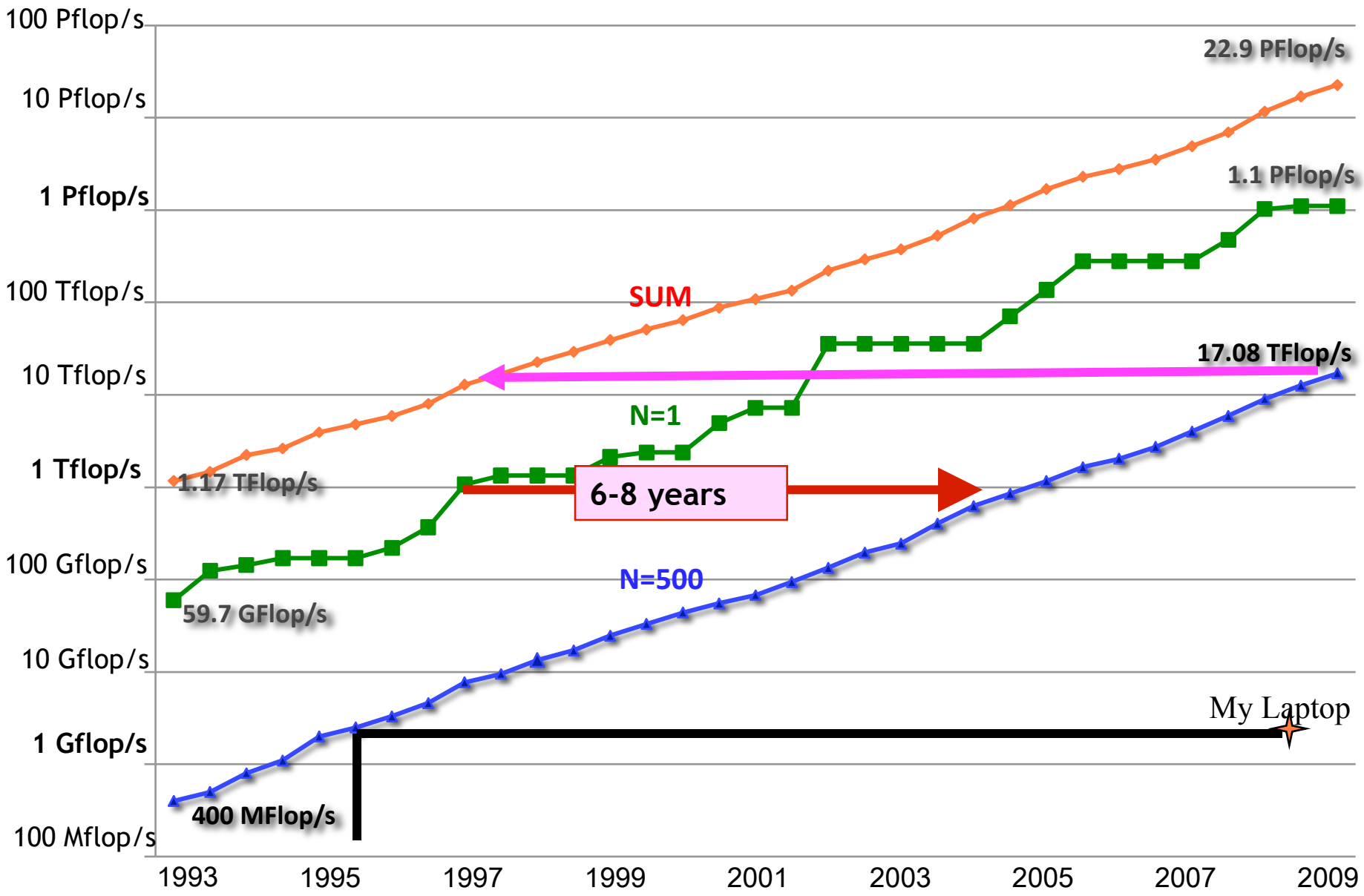
  $Ax=b$, *dense problem*



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

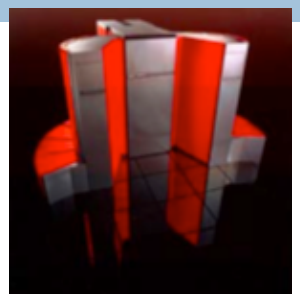
- All data available from **www.top500.org**

Performance Development



Looking at the Gordon Bell Prize

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

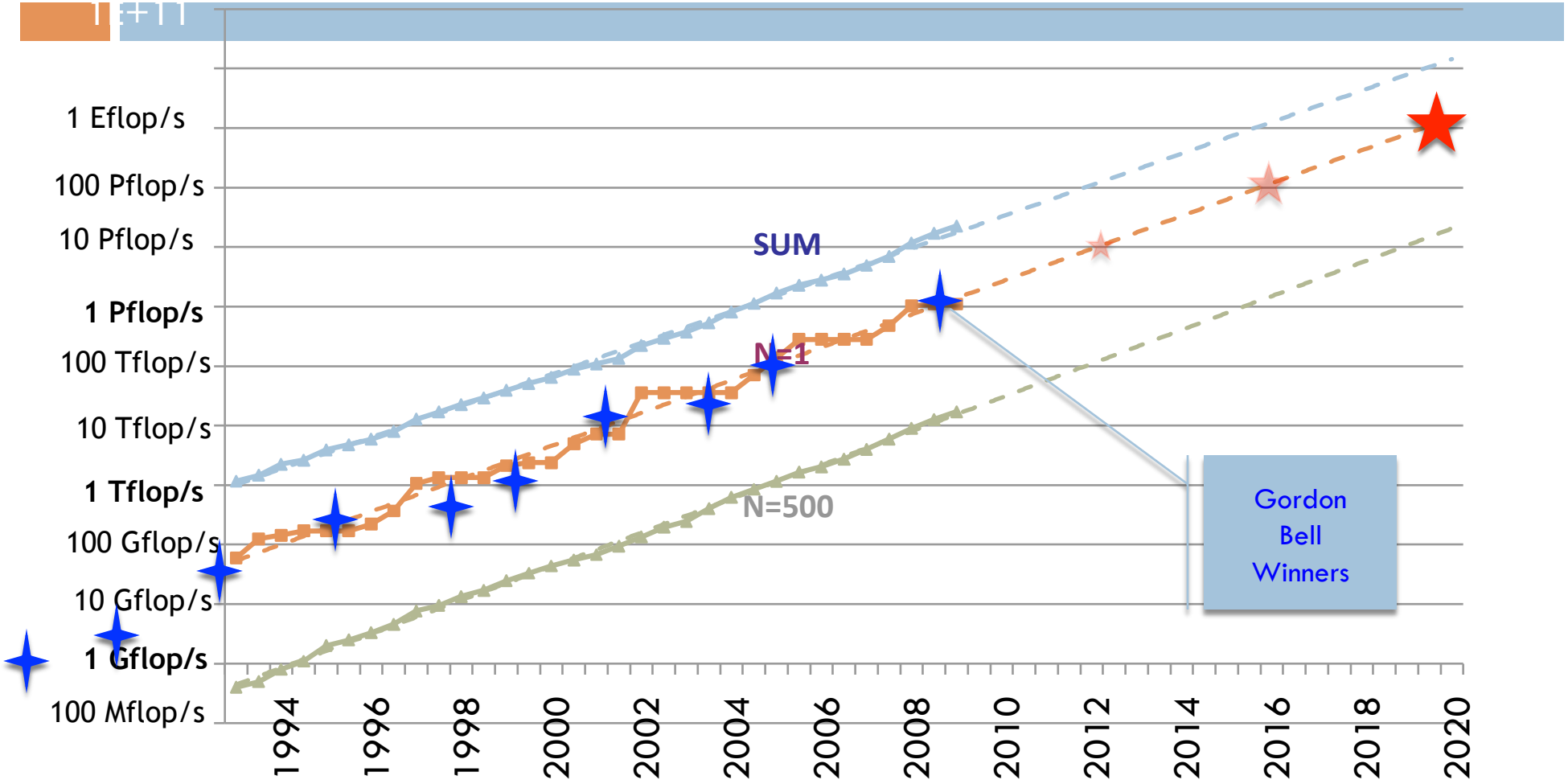


- 1 GFlop/s; 1988; Cray Y-MP; 8 Processors
  - Static finite element analysis

- 1 TFlop/s; 1998; Cray T3E; 1024 Processors
  - Modeling of metallic magnet atoms, using a variation of the locally self-consistent multiple scattering method.
- 1 PFlop/s; 2008; Cray XT5; 1.5×10^5 Processors
  - Superconductive materials

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

Performance Development in Top500



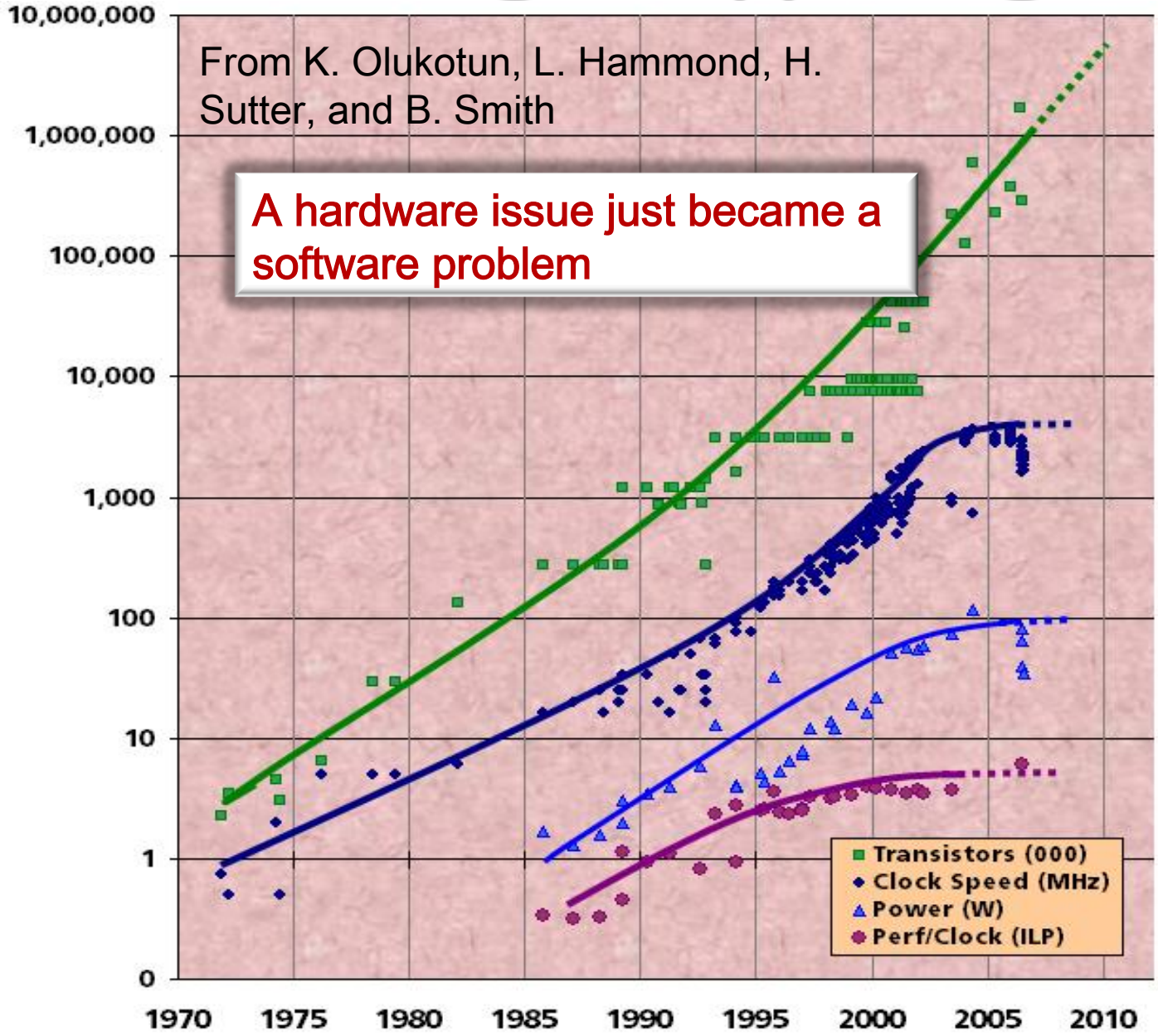
33rd List: The TOP10

Rank	Site	Computer	Country	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

33rd List: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Tflops]	% of Peak	Power [MW]	Flops/ Watt
1	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenter QS22/LS21	USA	129,600	1,105	76	2.48	446
2	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5 QC 2.3 GHz	USA	150,152	1,059	77	6.95	151
3	Forschungszentrum Juelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	825	82	2.26	365
4	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	51,200	480	79	2.09	230
5	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServer Blue Gene Solution	USA	212,992	478	80	2.32	206
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	1.26	363
8	NSF TACC/U. of Texas	Ranger / Sun SunBlade x6420	USA	62,976	433	75	2.0	217
9	DOE / NNSA Lawrence Livermore NL	Dawn / IBM Blue Gene/P Solution	USA	147,456	415	83	1.13	367
10	Forschungszentrum Juelich (FZJ)	JUROPA /Sun - Bull SA NovaScale /Sun Blade	Germany	26,304	274	89	1.54	178

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

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

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)

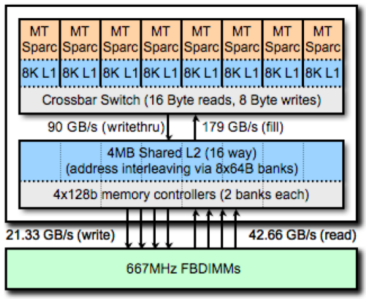
50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device

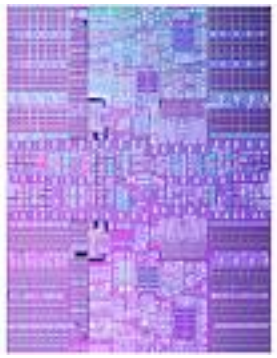
Today's Multicores

99% of Top500 Systems Are Based on Multicore

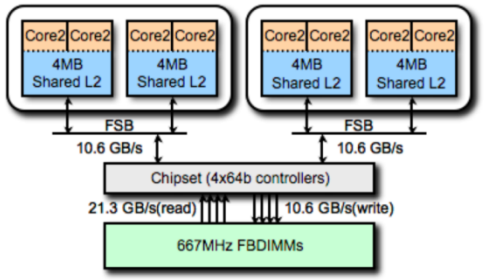
282 use Quad-Core
204 use Dual-Core
3 use Nona-core



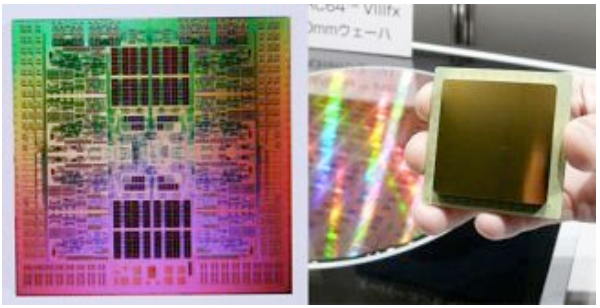
Sun Niagra2 (8 cores)



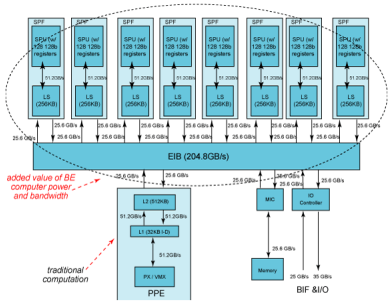
IBM Power 7 (8 cores)



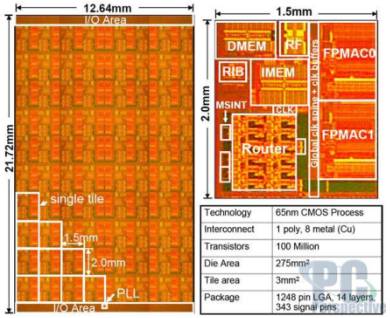
Intel Clovertown (4 cores)



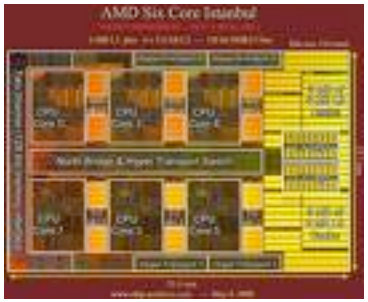
Fujitsu Venus (8 cores)



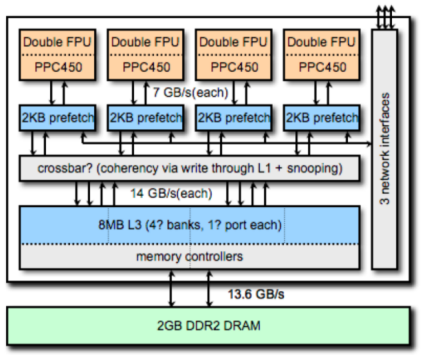
IBM Cell (9 cores)



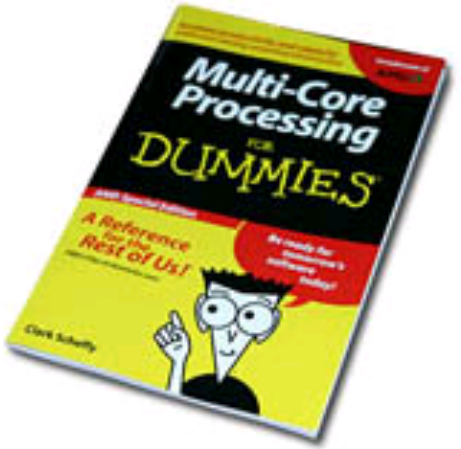
Intel Polaris [experimental] (80 cores)

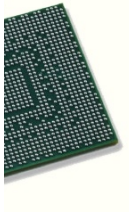


AMD Istambul (6 cores)



IBM BG/P (4 cores)





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

Major Changes to Software

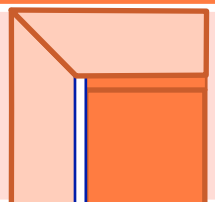
- Must rethink the design of our software
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software
- Numerical libraries for example will change
  - For example, both LAPACK and ScaLAPACK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

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)

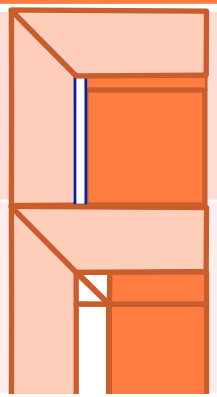
A New Generation of Software:

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

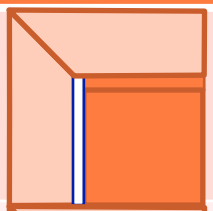
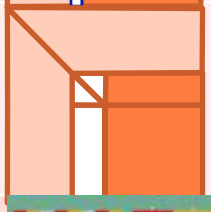
A New Generation of Software:

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 friendly)		Rely on - Level-3 BLAS operations

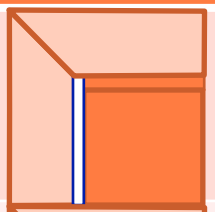
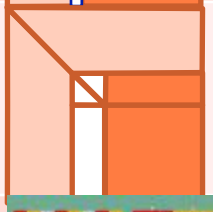
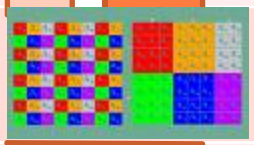
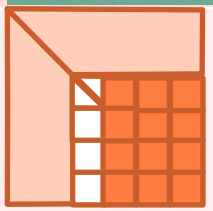
A New Generation of Software:

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

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

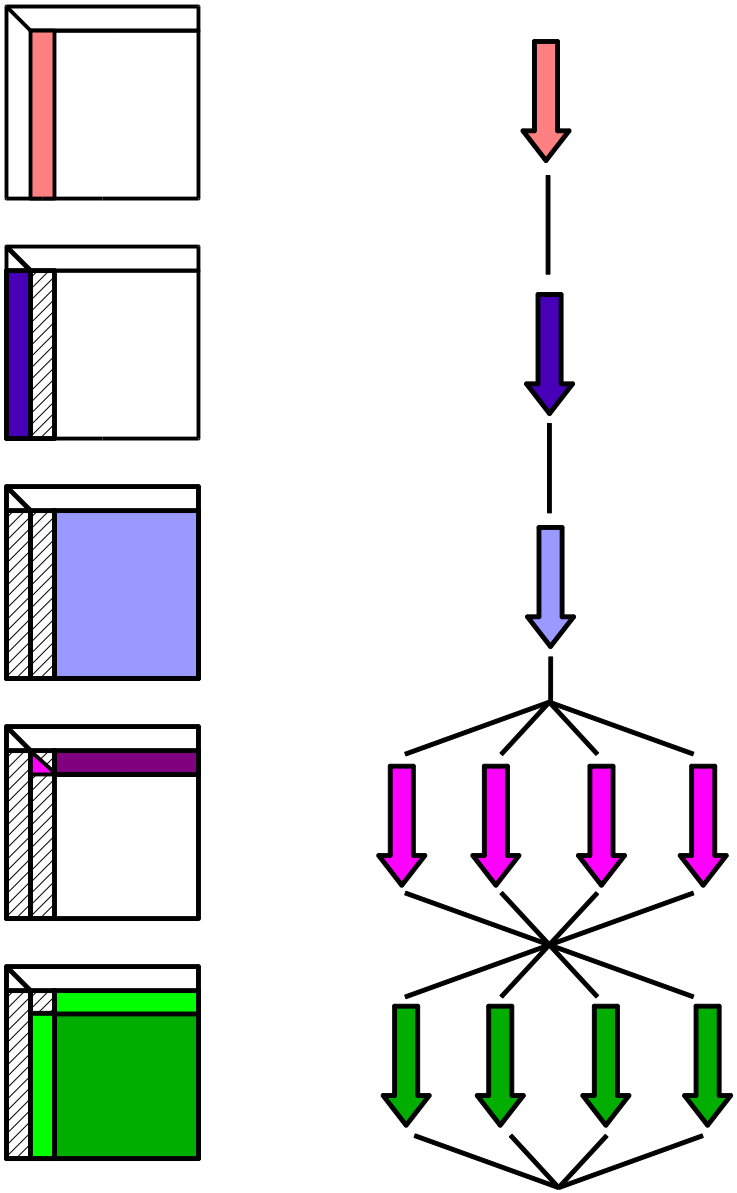
DGETF2
(Factor a panel)

DLSWP
(Backward swap)

DLSWP
(Forward swap)

DTRSM
(Triangular solve)

DGEMM
(Matrix multiply)



LAPACK

LAPACK

LAPACK

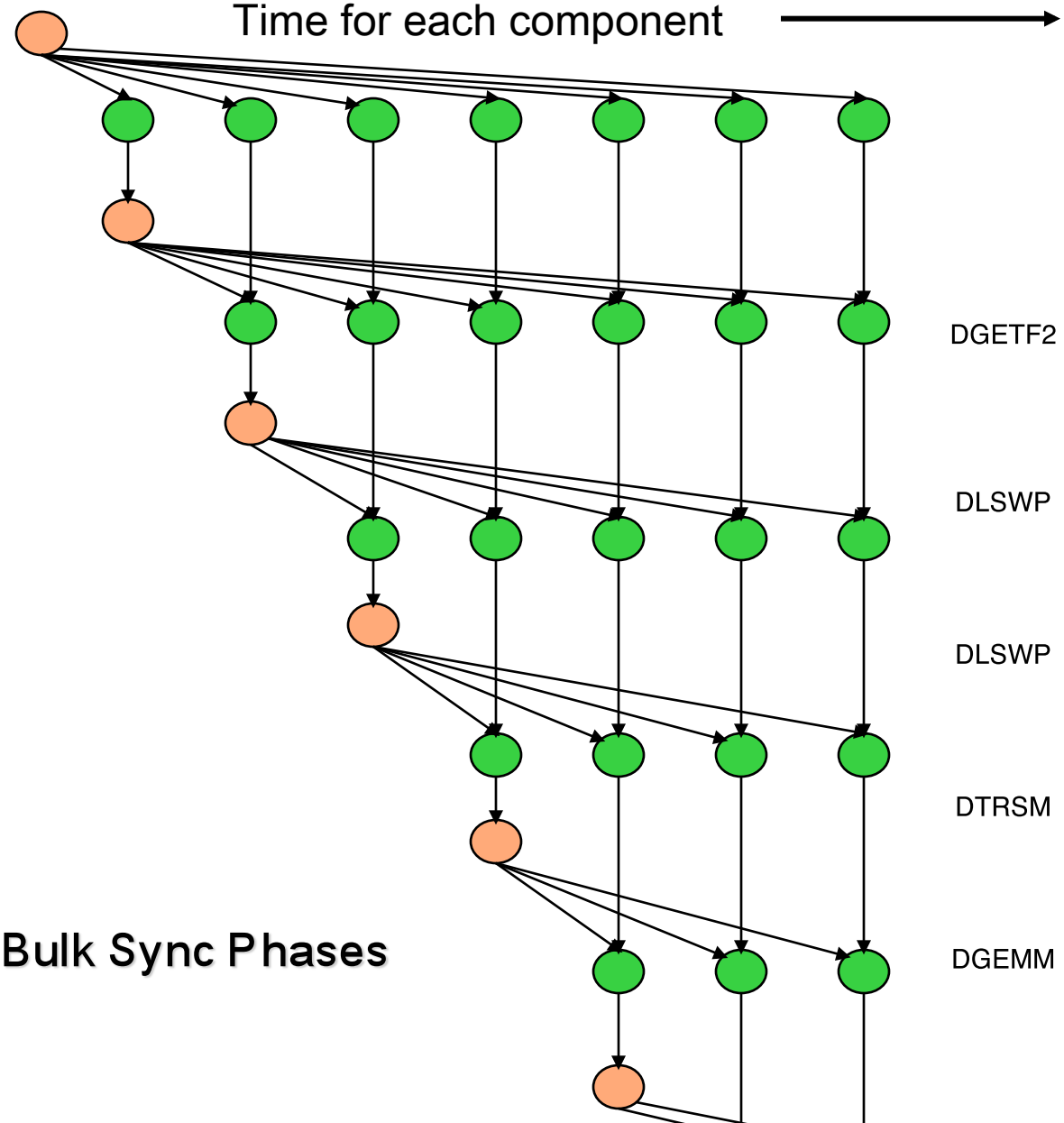
BLAS

BLAS

LU Timing Profile (16 core system)

Threads – no lookahead

Time for each component



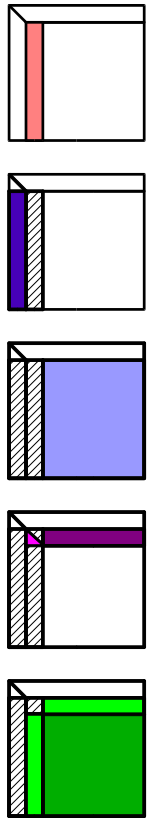
DGETF2

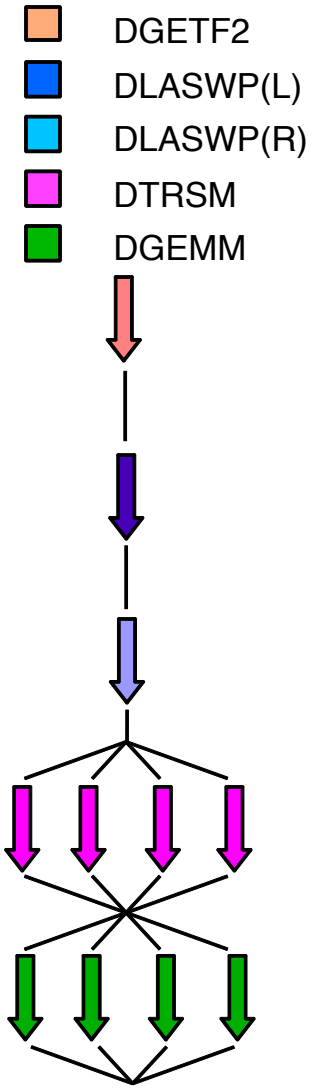
DLSWP

DLSWP

DTRSM

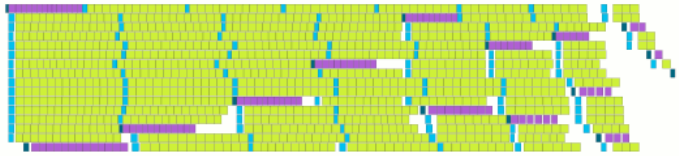
DGEMM

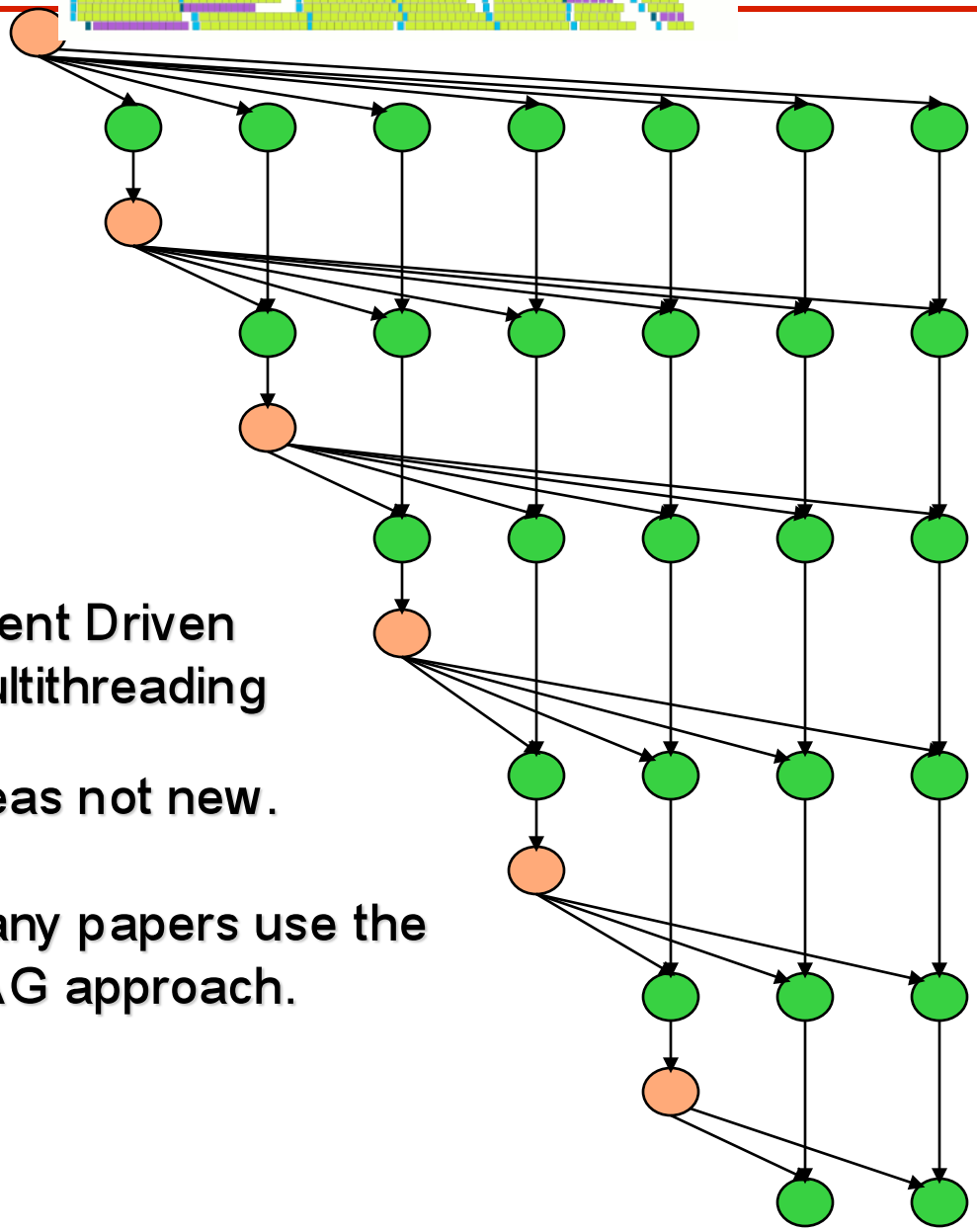




Bulk Sync Phases

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

PLASMA (Redesign LAPACK/ScaLAPACK)

Parallel Linear Algebra Software for Multicore Architectures

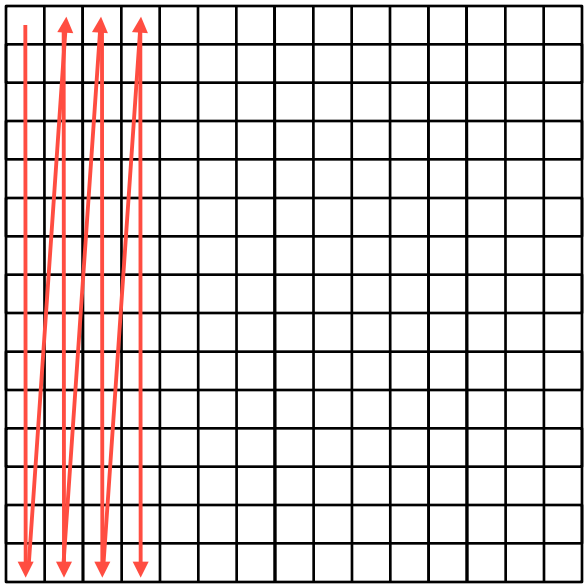
- **Asychronicity**
  - **Avoid fork-join (Bulk sync design)**
- **Dynamic Scheduling**
  - **Out of order execution**
- **Fine Granularity**
  - **Independent block operations**
- **Locality of Reference**
  - **Data storage - Block Data Layout**

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

Achieving Fine Granularity

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

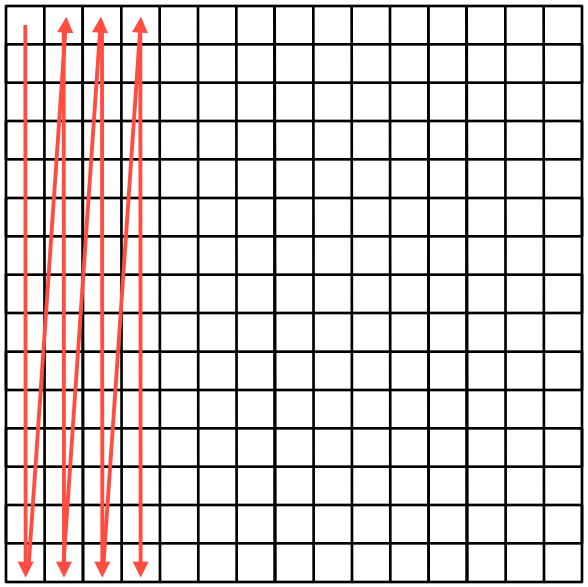
Column-Major



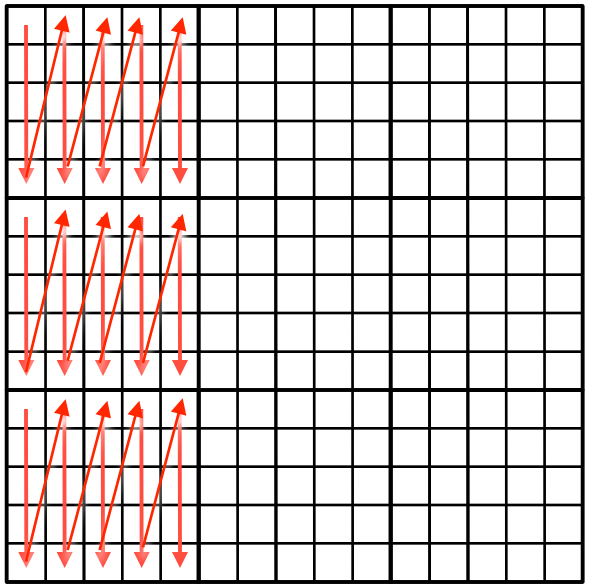
Achieving Fine Granularity

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

Column-Major

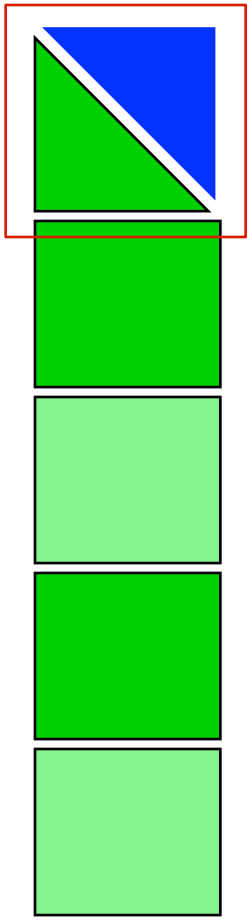


Blocked



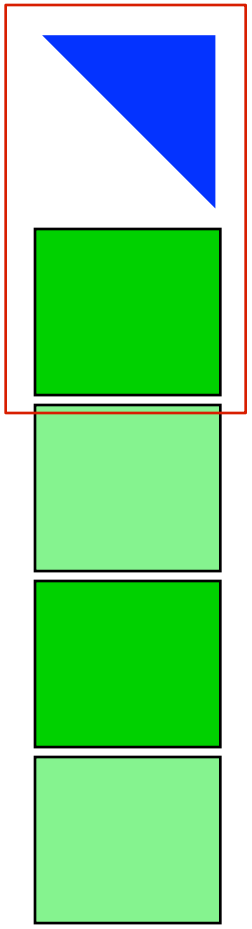
WS Tuesday Track E: Novel Data Formats and Algorithms for HPC
Fred Gustavson, Jerzy Wasniewski, and JD on A Fast Minimal Storage Sym Ind Matrix Fact.

Parallel Tasks in LU



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

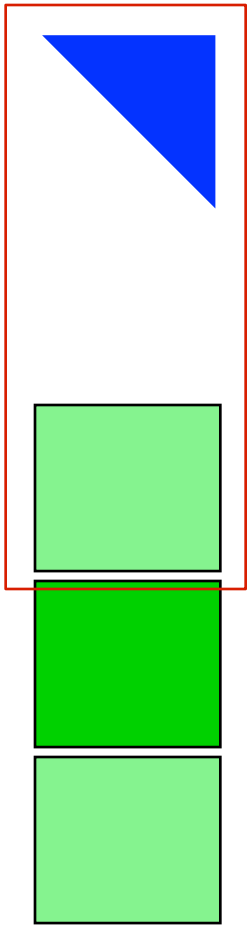
Parallel Tasks in LU



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

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

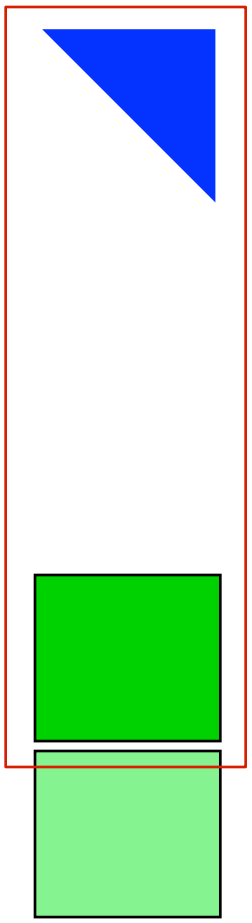
Parallel Tasks in LU



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)

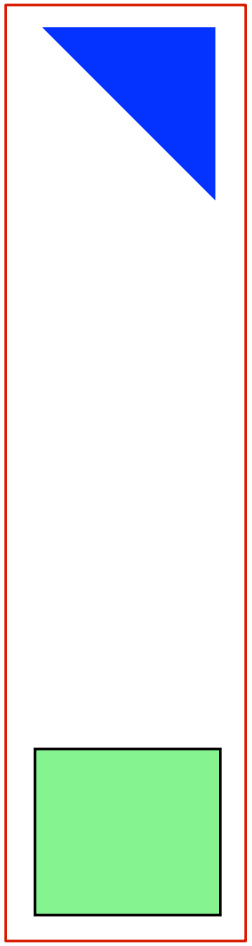
Step3: Use $U_{1,1}$ to zero $A_{1,3}$ (w/partial pivoting)

.

.

.

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)

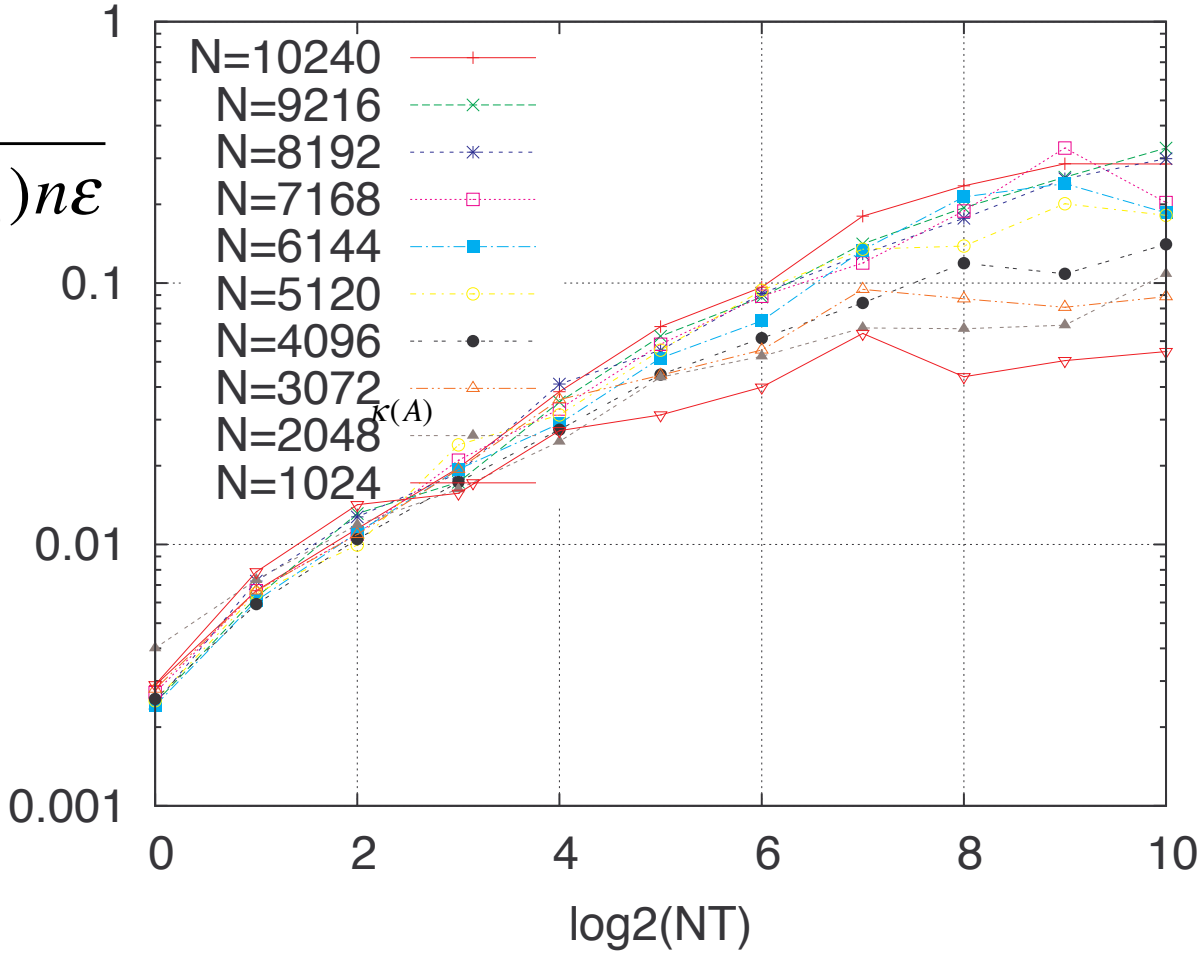
.

.

.

Residual from PLASMA's Tiled LU

$$\frac{\| Ax - b \|_\infty}{(\| A \|_\infty \| x \|_\infty + \| b \|_\infty) n \varepsilon}$$

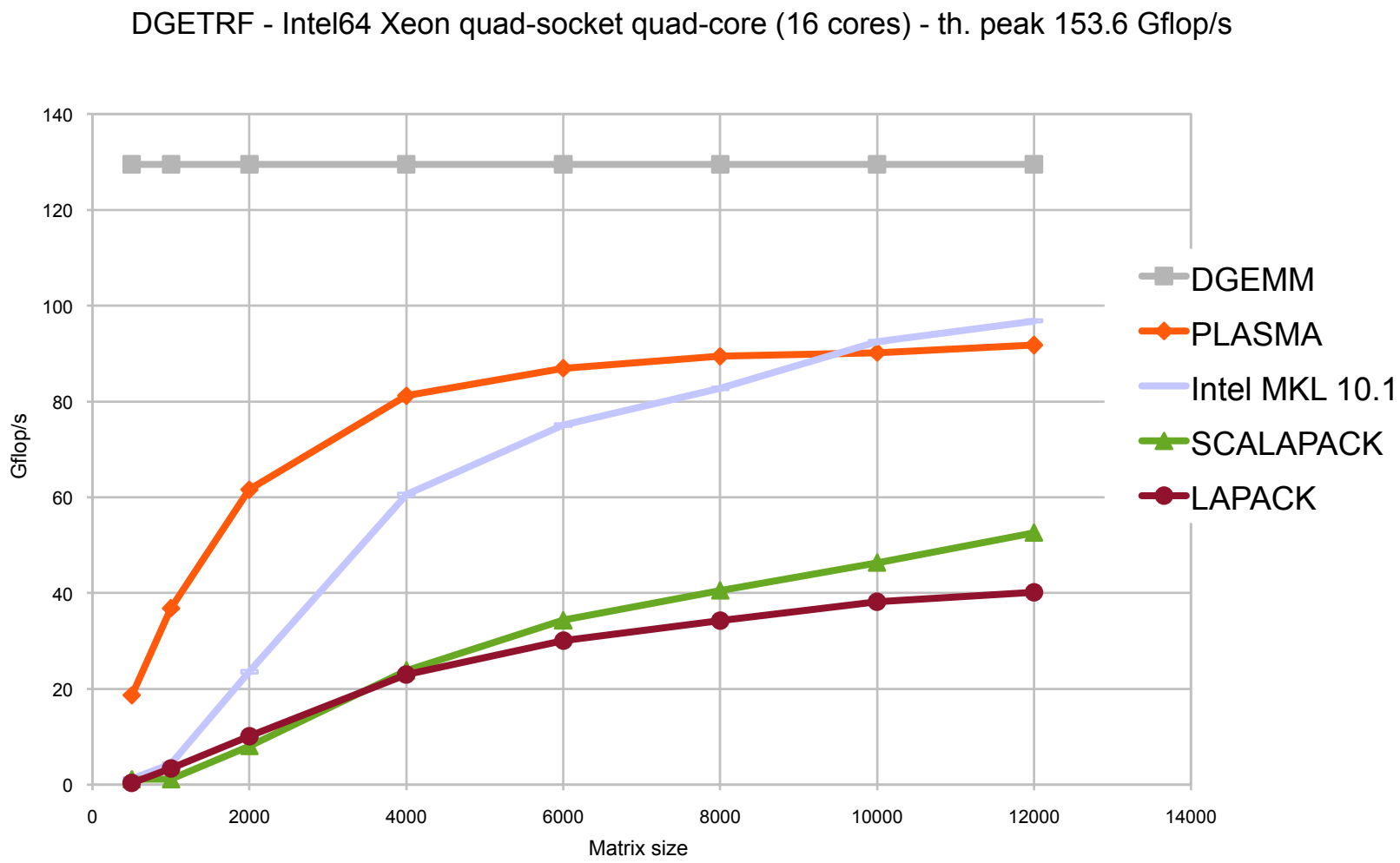


Random Matrices

NT (Number of Tiles)

$\kappa(A) \cdot 10^5 - 10^8$

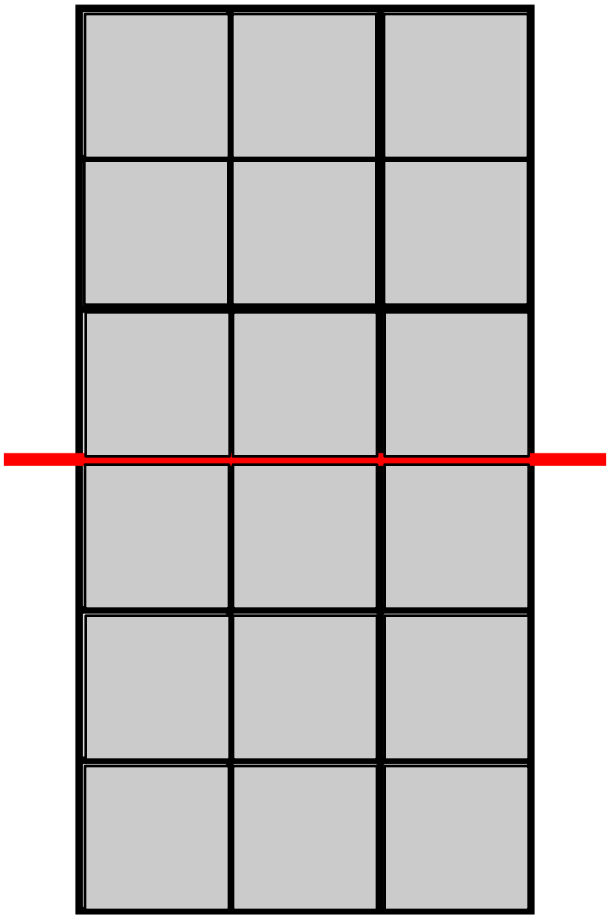
DGETRF - Intel64 - 16 cores



Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains



3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains

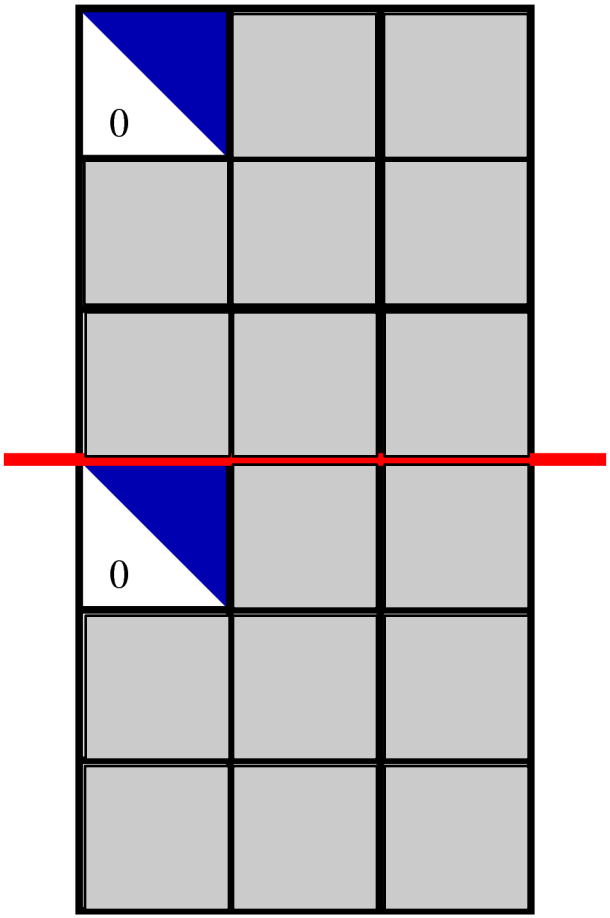
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- **panel factorization**
- updating the trailing submatrix
- merge the domains



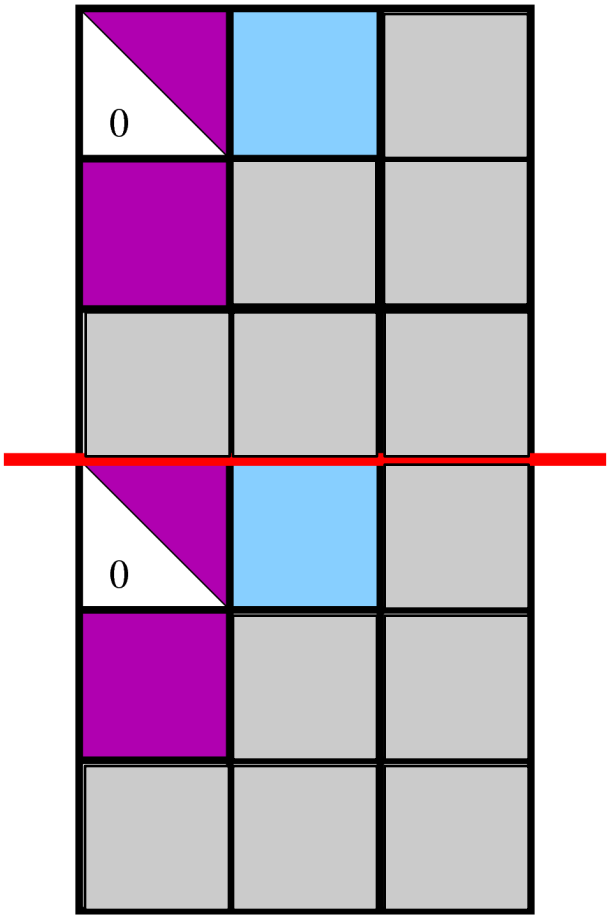
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



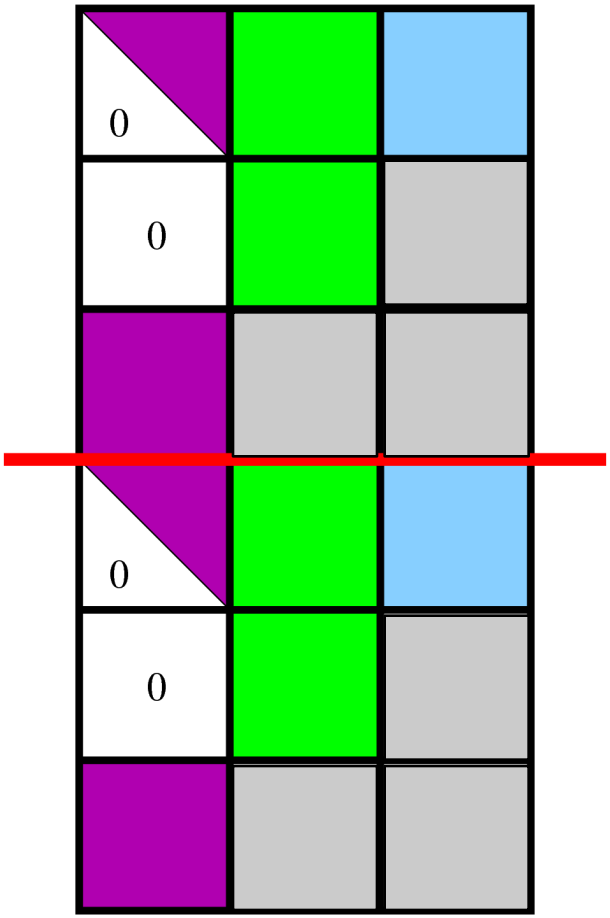
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



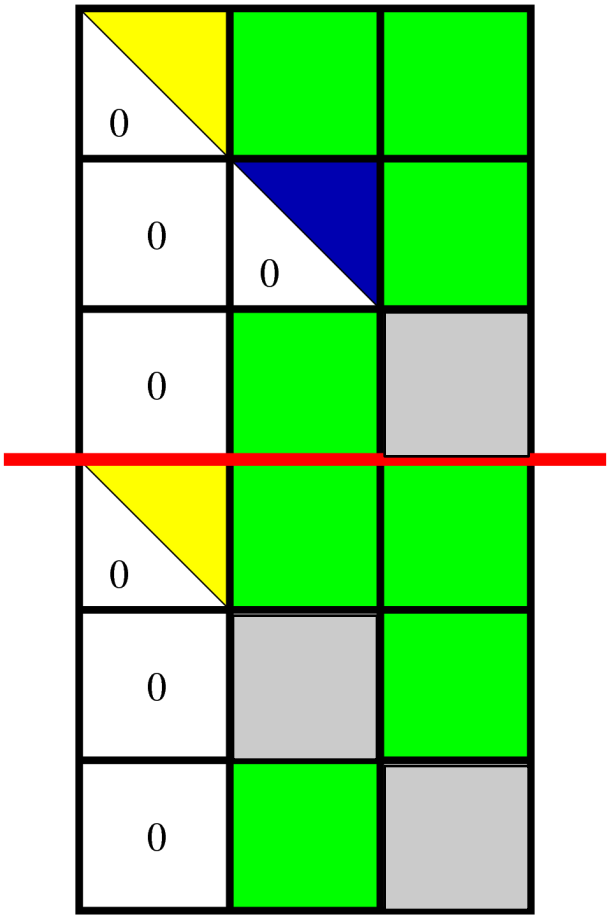
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



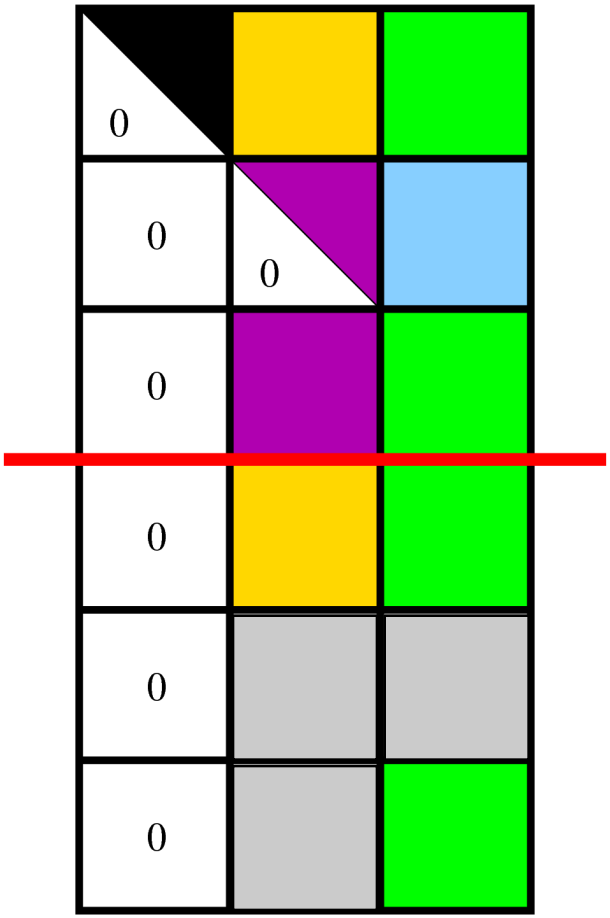
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



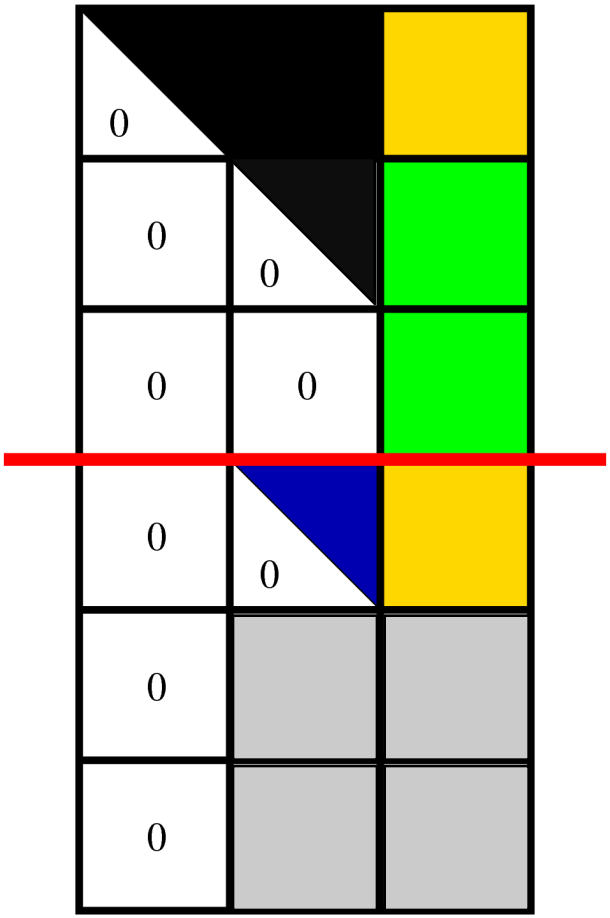
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



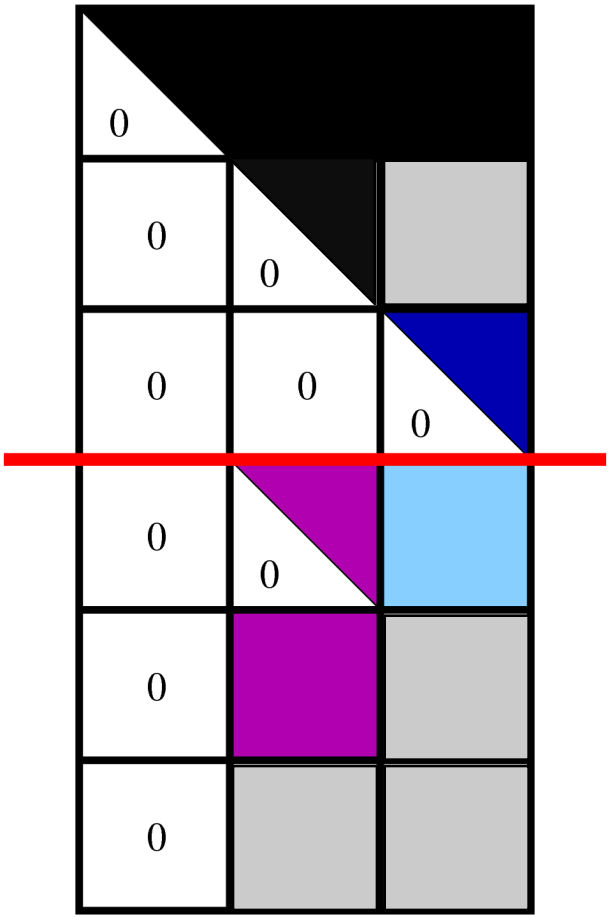
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



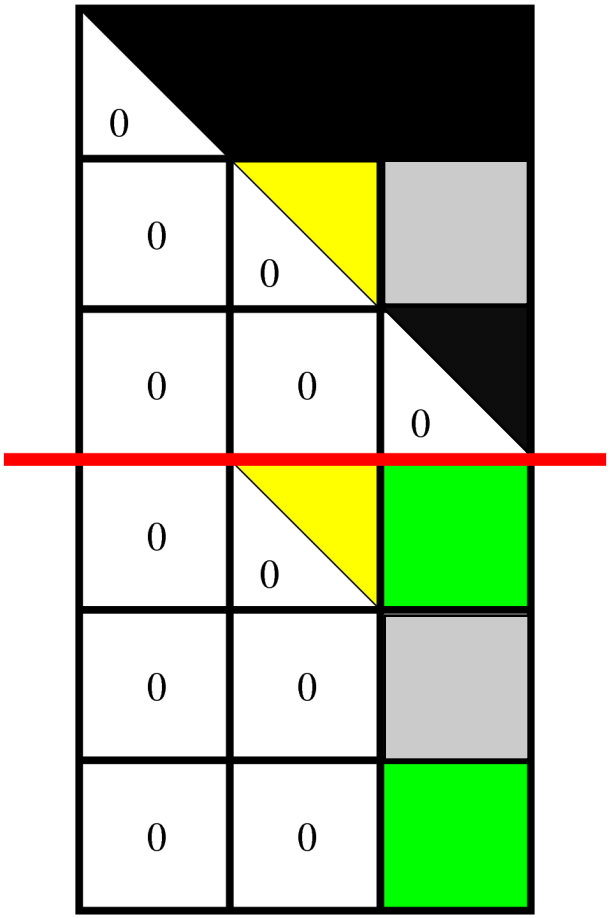
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



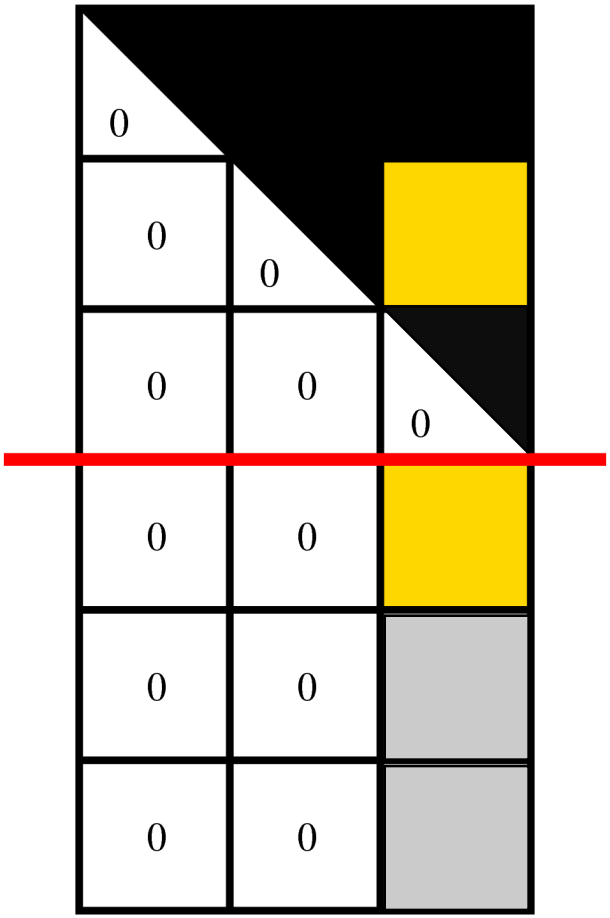
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



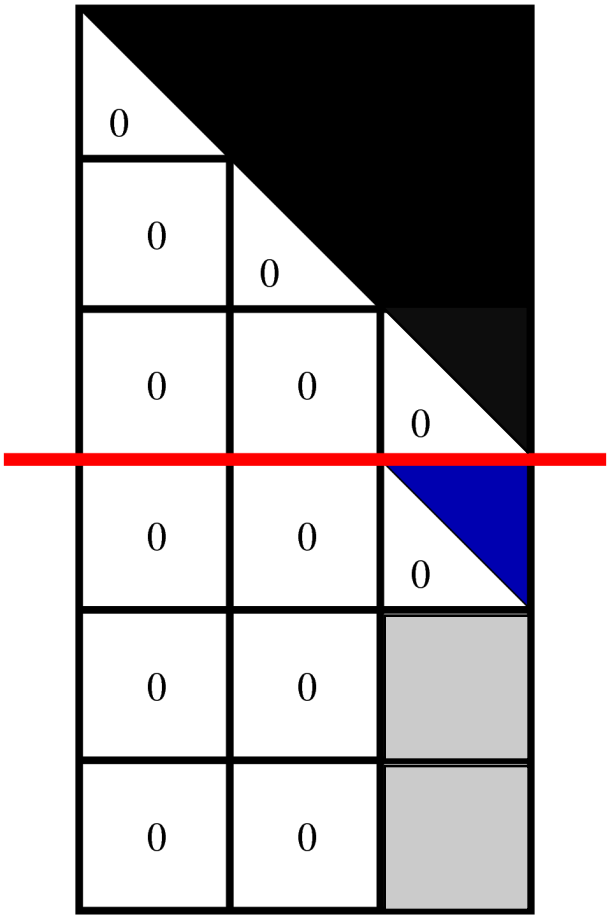
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- **panel factorization**
- updating the trailing submatrix
- merge the domains



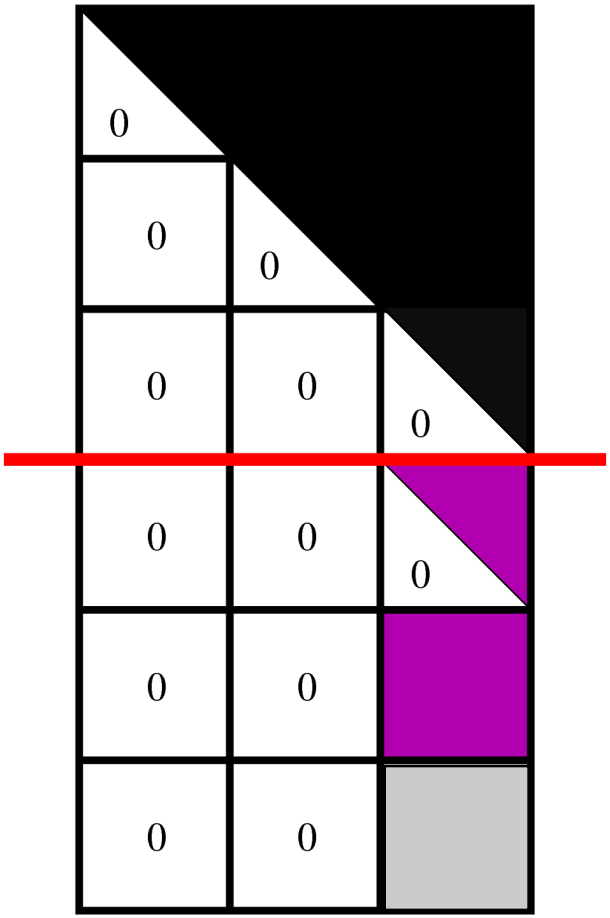
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- **panel factorization**
- updating the trailing submatrix
- merge the domains



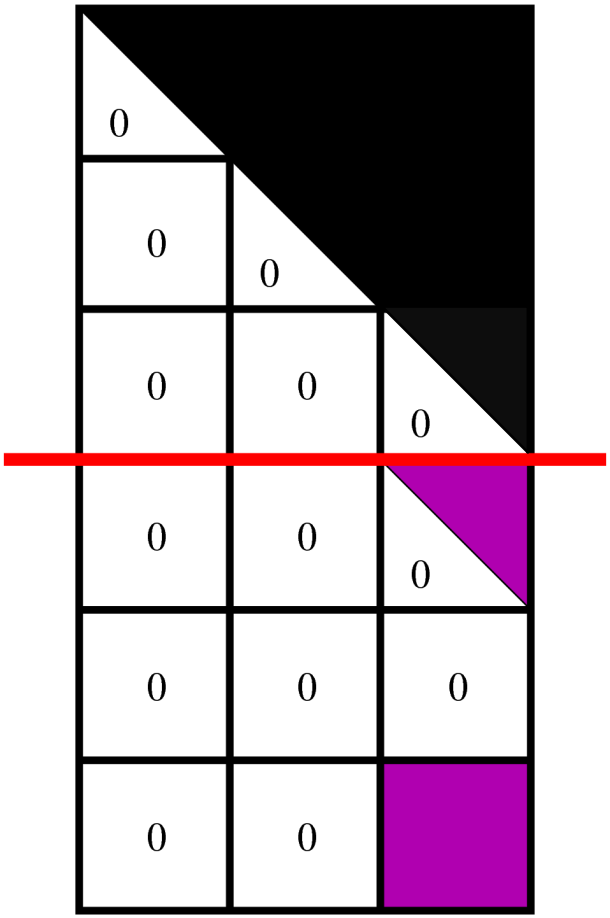
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- **panel factorization**
- updating the trailing submatrix
- merge the domains



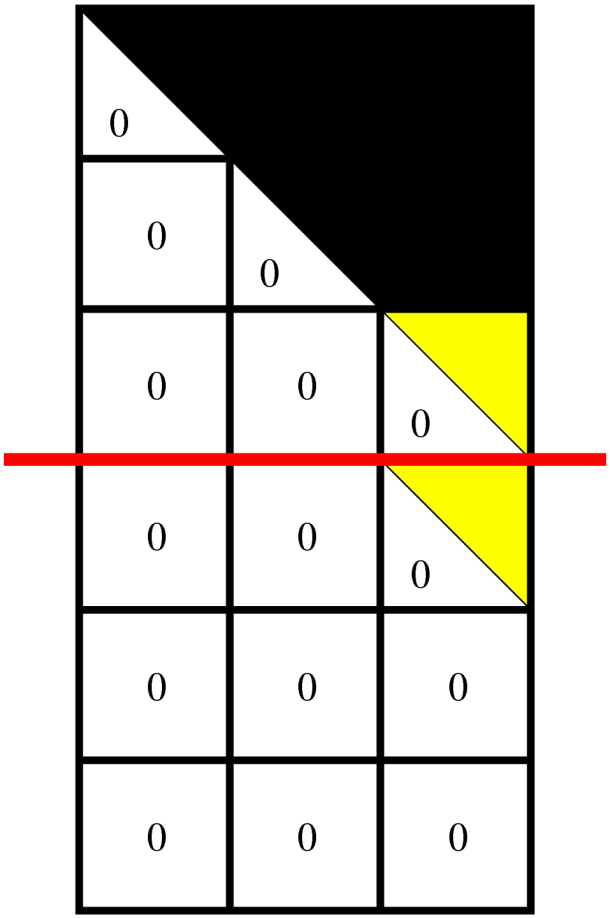
Communication Avoiding QR Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains



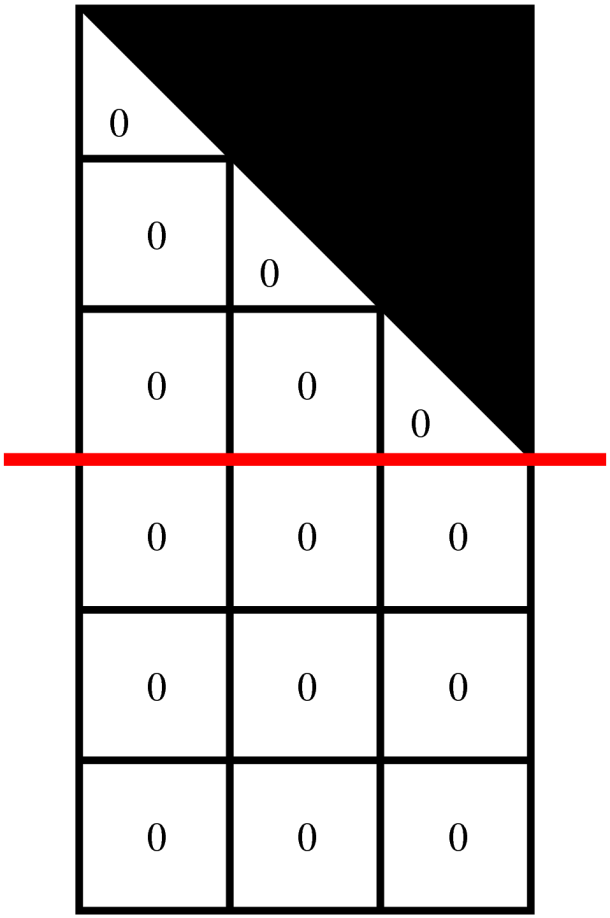
Communication Avoiding QR Factorization

TS matrix

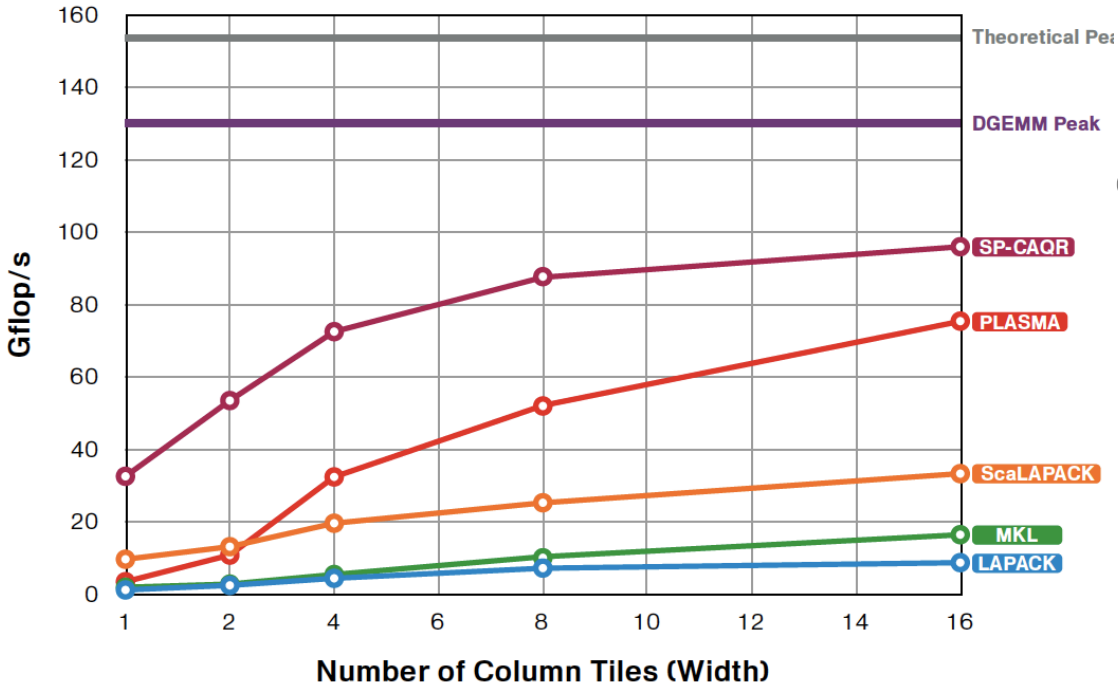
- MT=6 and NT=3
- split into 2 domains

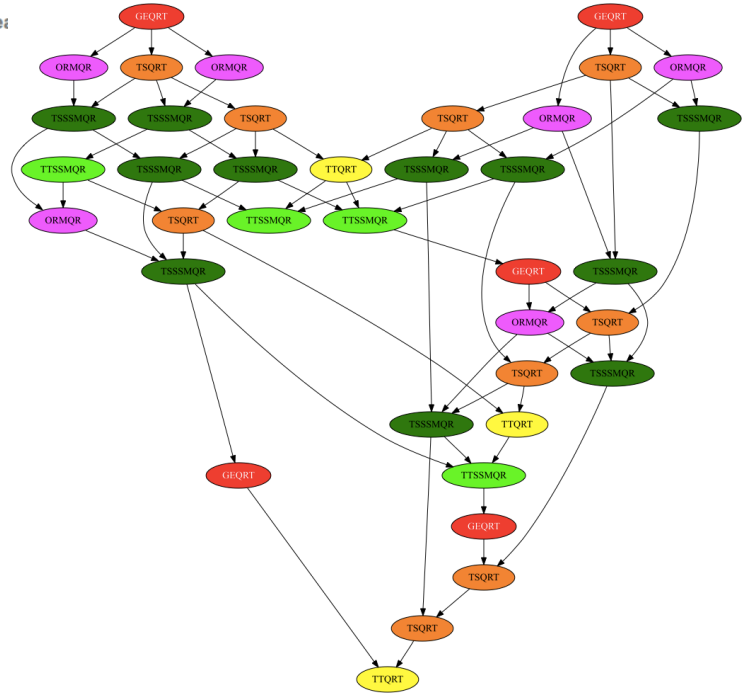
3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains
- Final R computed



Communication Avoiding QR Factorization



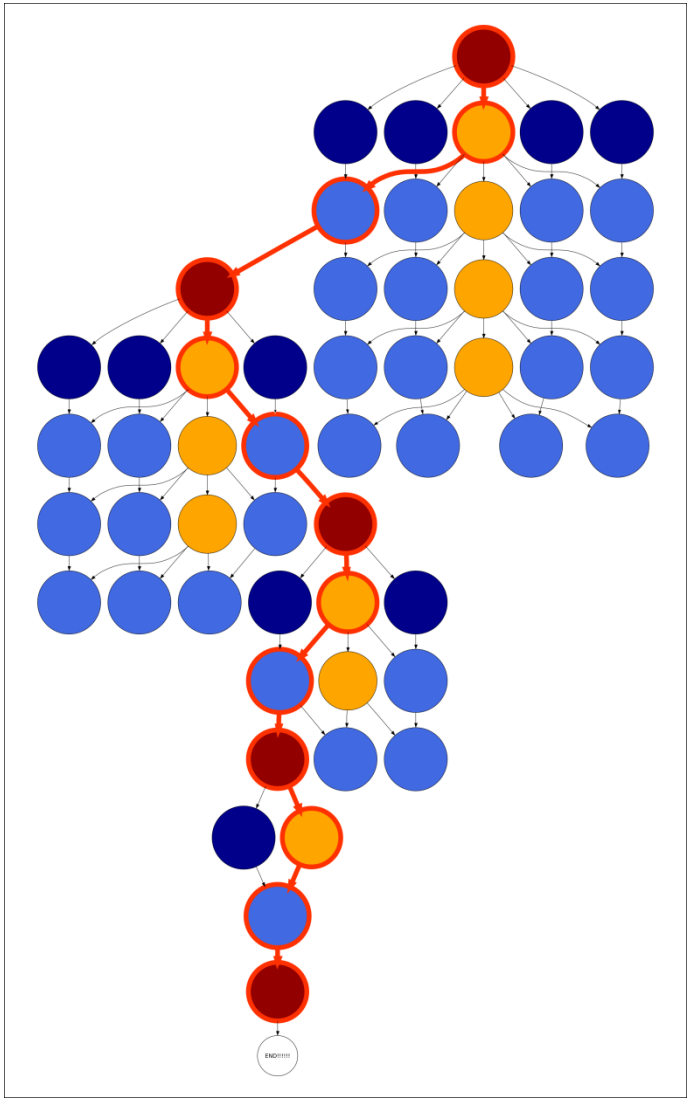


Quad-socket, quad-core machine Intel Xeon EMT64 E7340 at 2.39 GHz.
Theoretical peak is 153.2 Gflop/s with 16 cores.

Matrix size 51200 by 3200

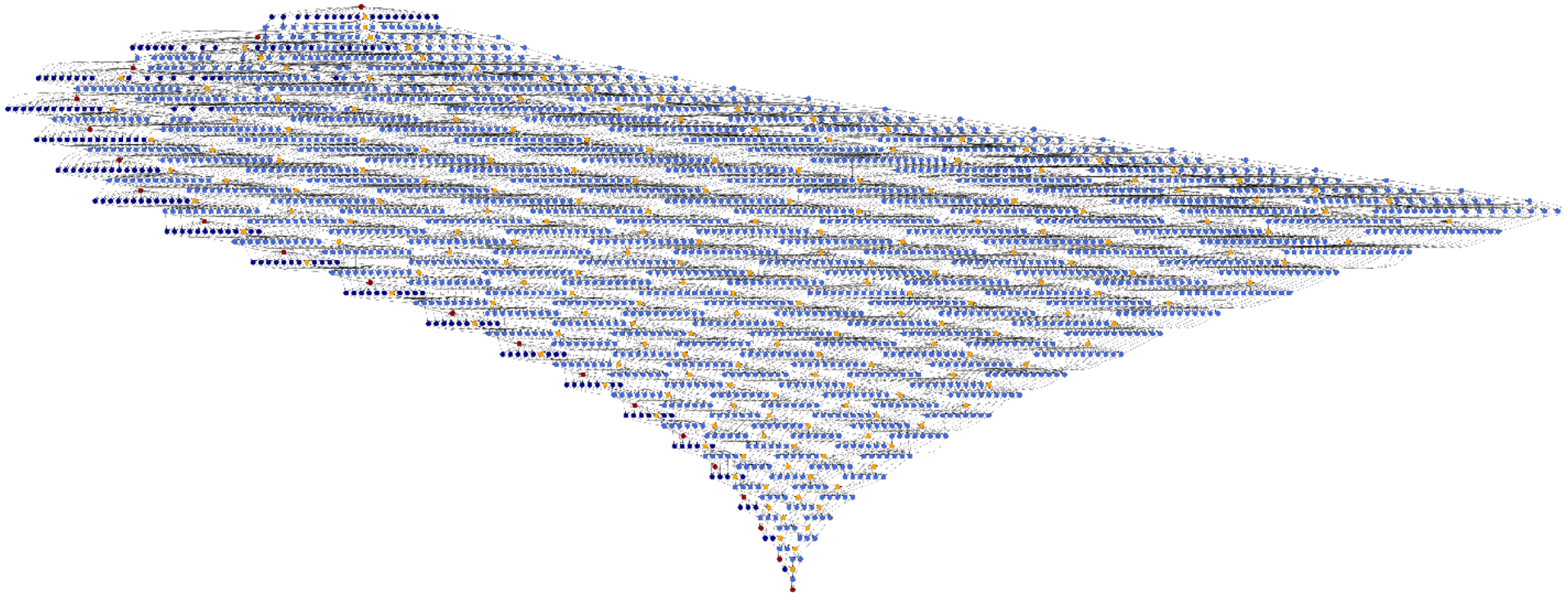
If We Had A Small Matrix Problem

- 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



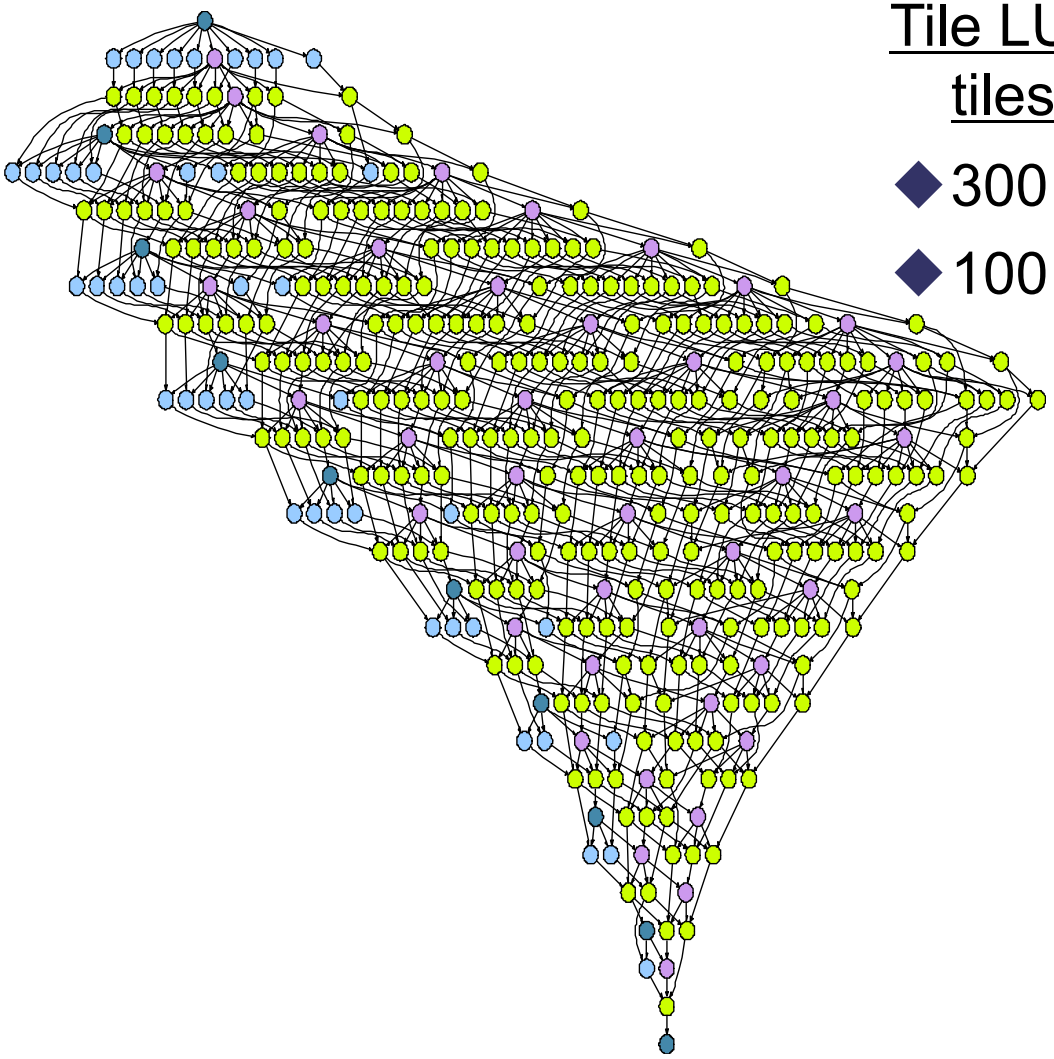
The DAGs are Large

- 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

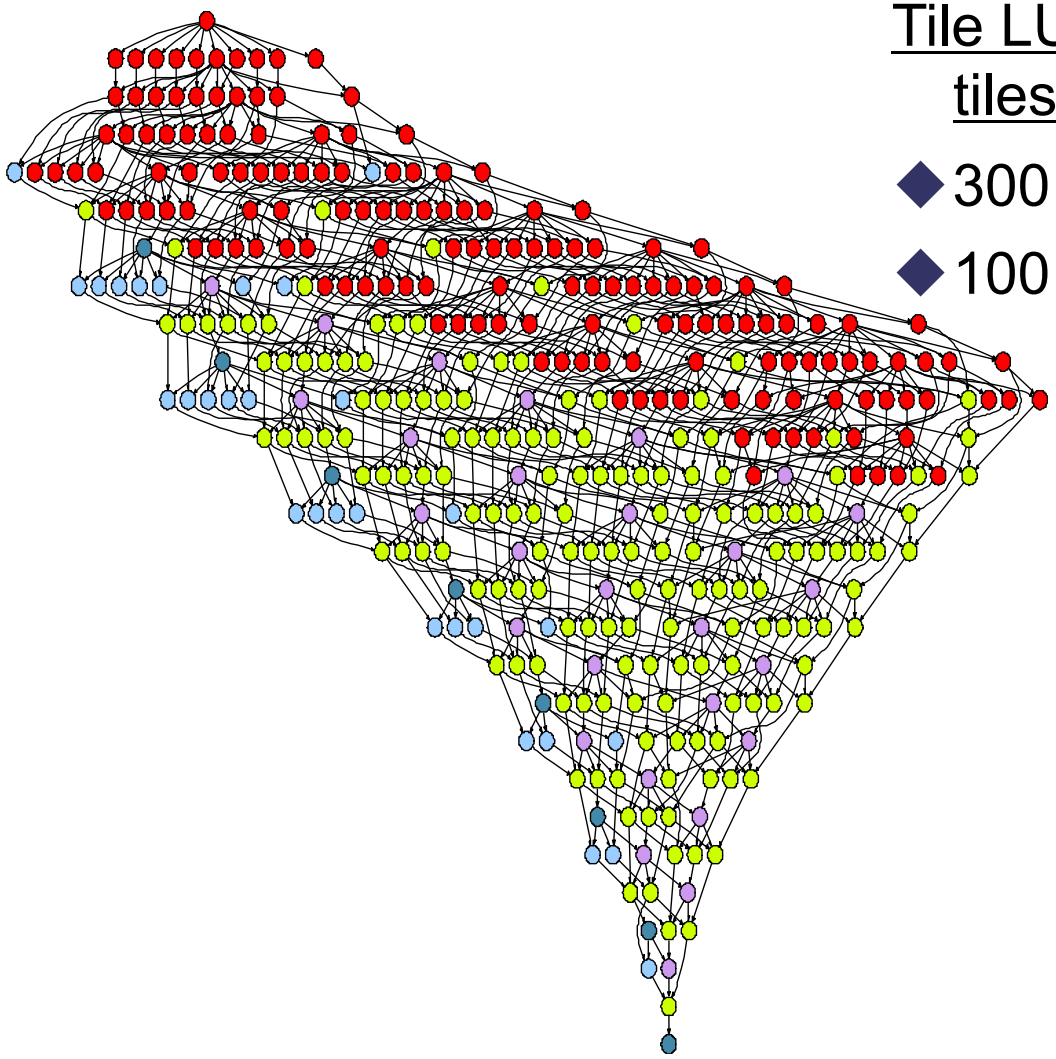
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

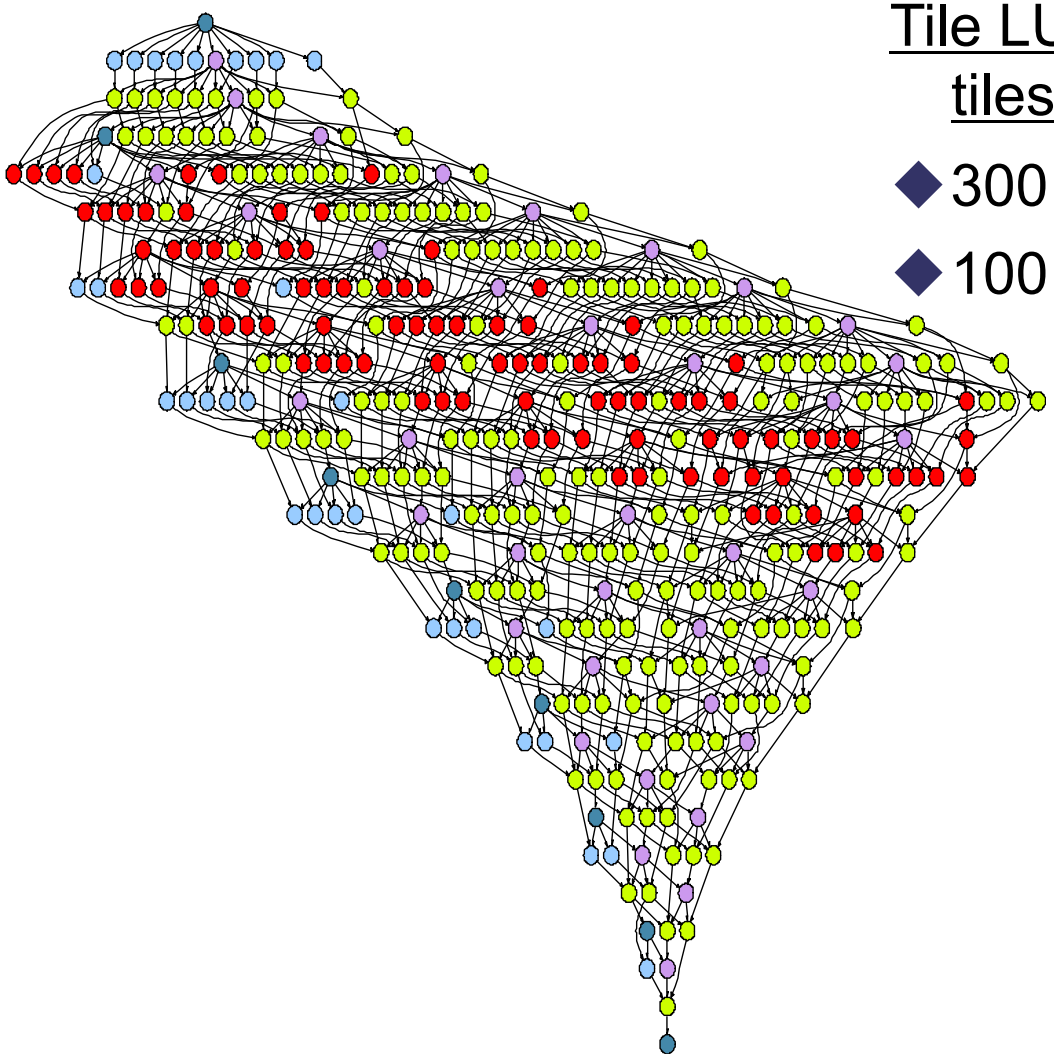
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

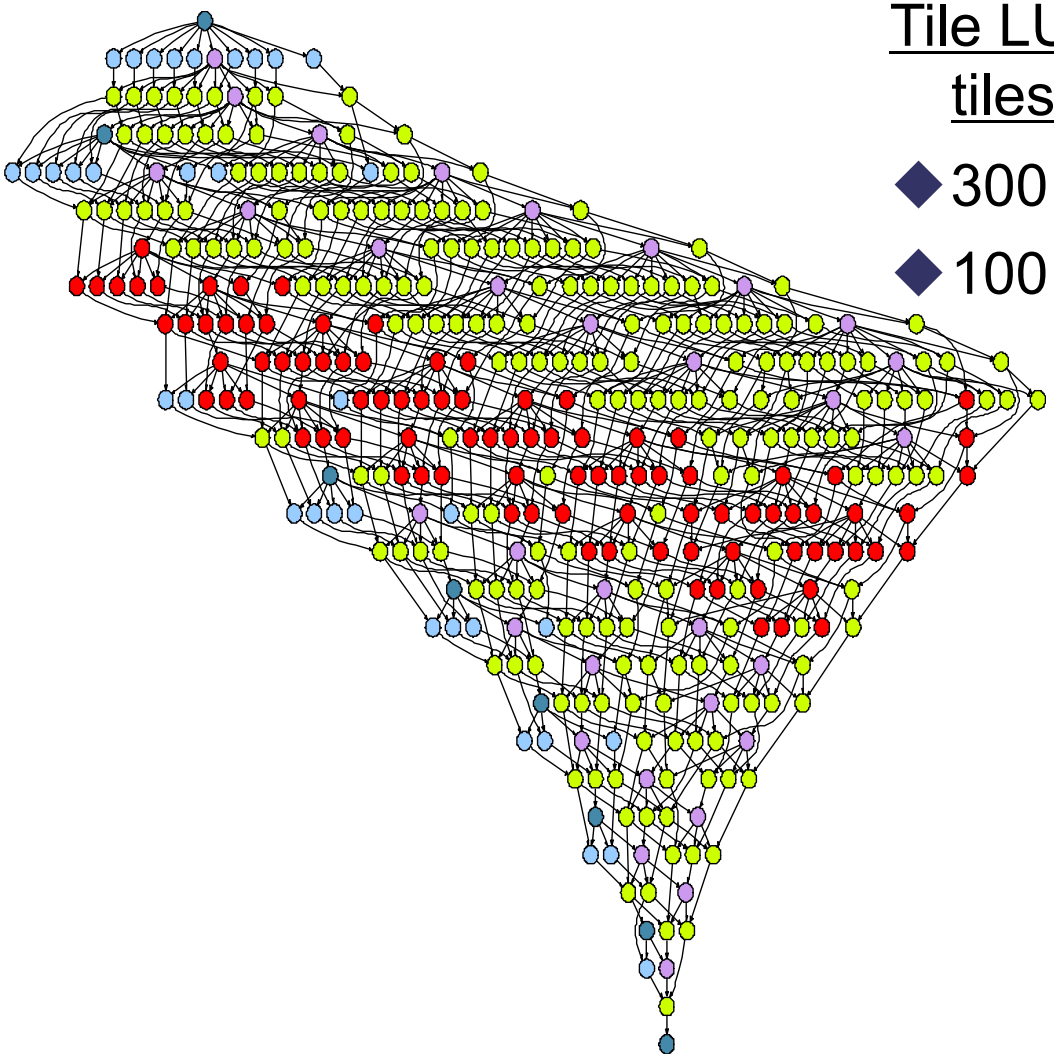
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

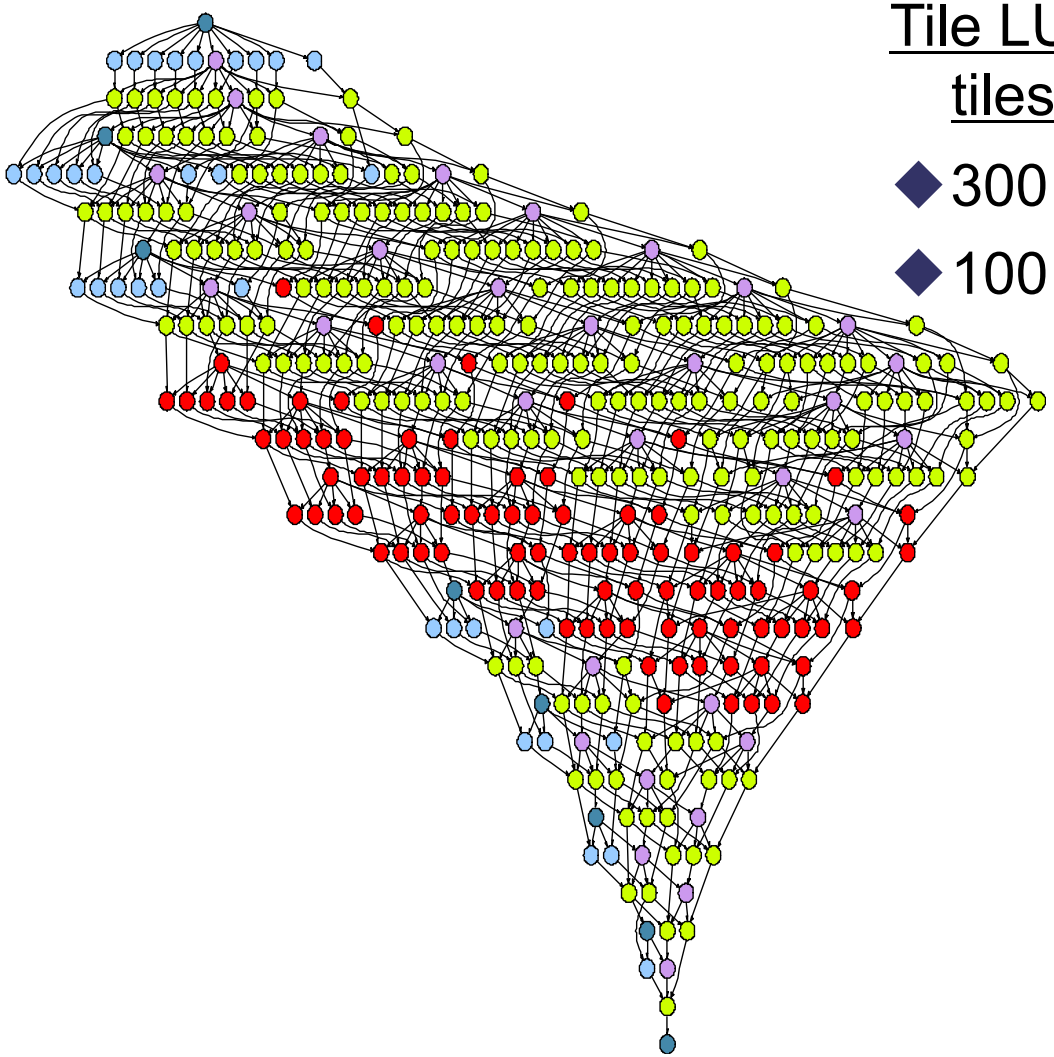
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

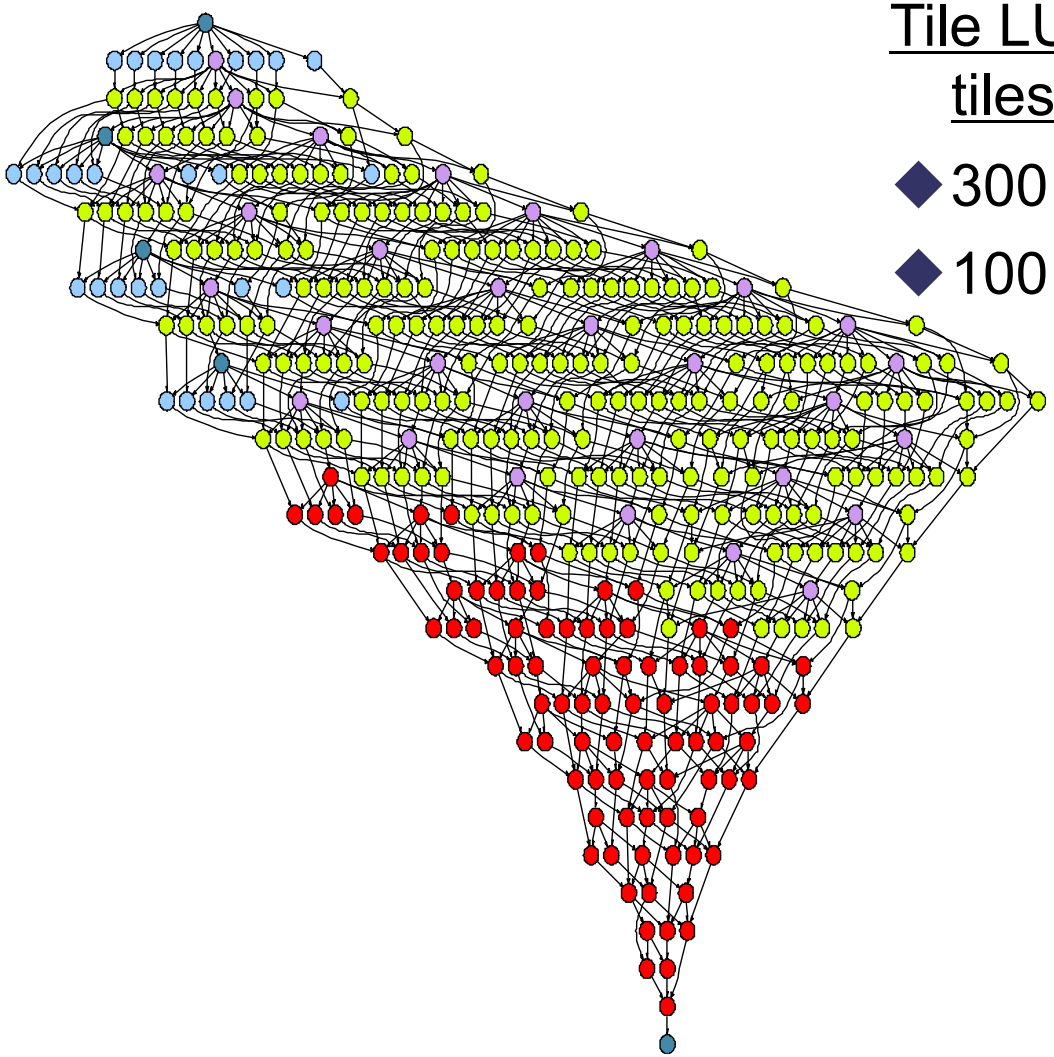
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

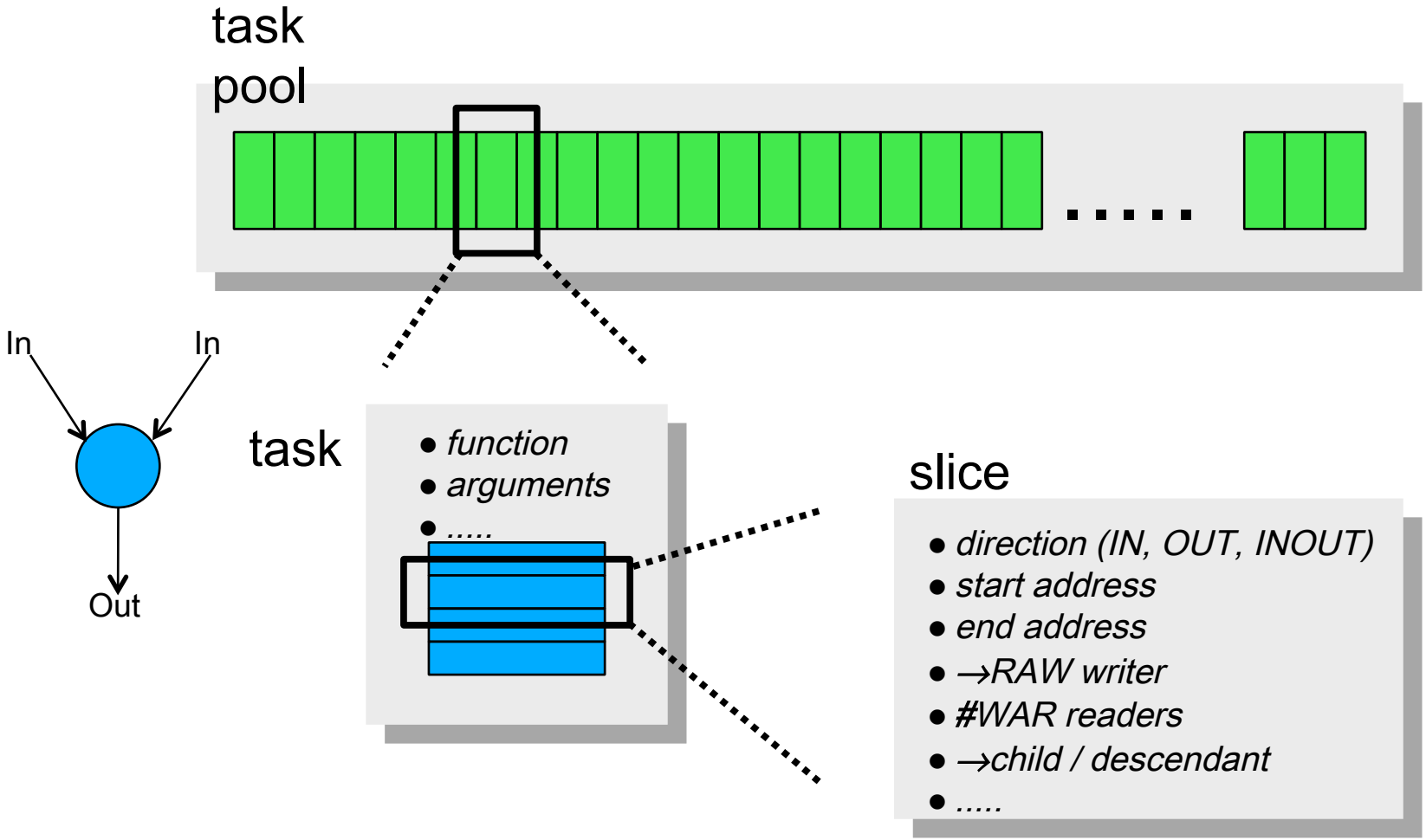
Execution of the DAG by a Sliding Window



Tile LU factorization 10x10 tiles

- 300 tasks total
- 100 task window

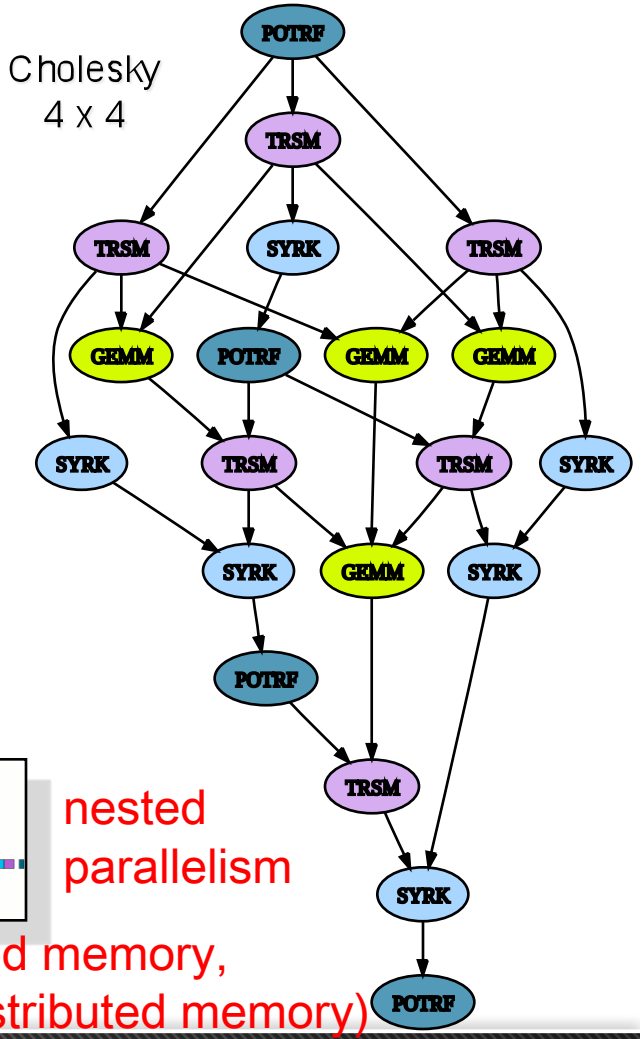
PLASMA Dynamic Task Scheduler

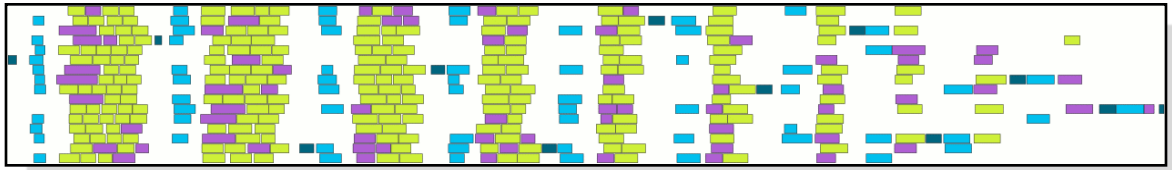


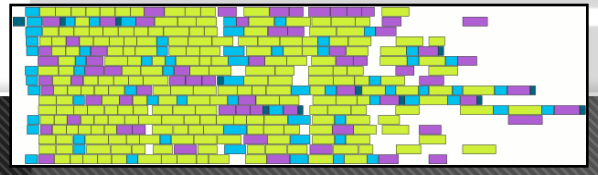
- task – a unit of scheduling (quantum of work)
- slice – a unit of dependency resolution (quantum of data)
- WS Tuesday Track E: Novel Data Formats and Algorithms for HPC
  Jakub Kurzak, Hatem Ltaief, Rosa Badia, and JD on Dependency Driven Scheduling....

PLASMA: Parallel Linear Algebra s/w for Multicore Architectures

- Objectives
  - high utilization of each core
  - scaling to large number of cores
  - shared or distributed memory
- Methodology
  - DAG scheduling
  - explicit parallelism
  - implicit communication
  - Fine granularity / block data layout
- Arbitrary DAG with dynamic scheduling







nested parallelism

PLASMA (Today shared memory, next next distributed memory)

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

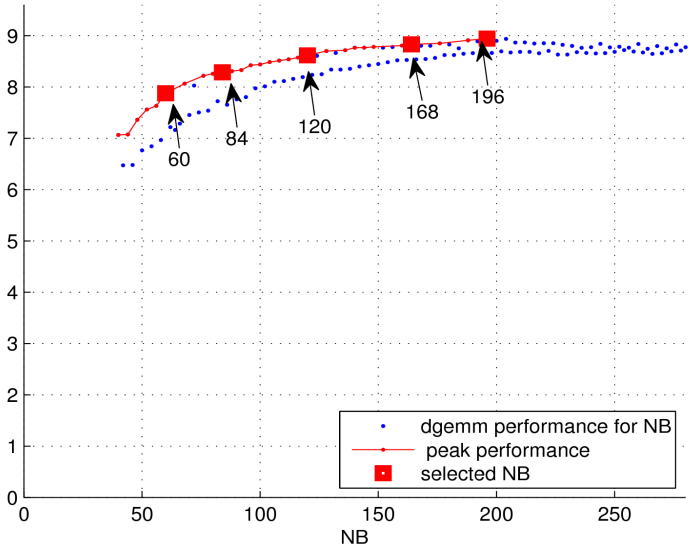
Auto-Tuning

- 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

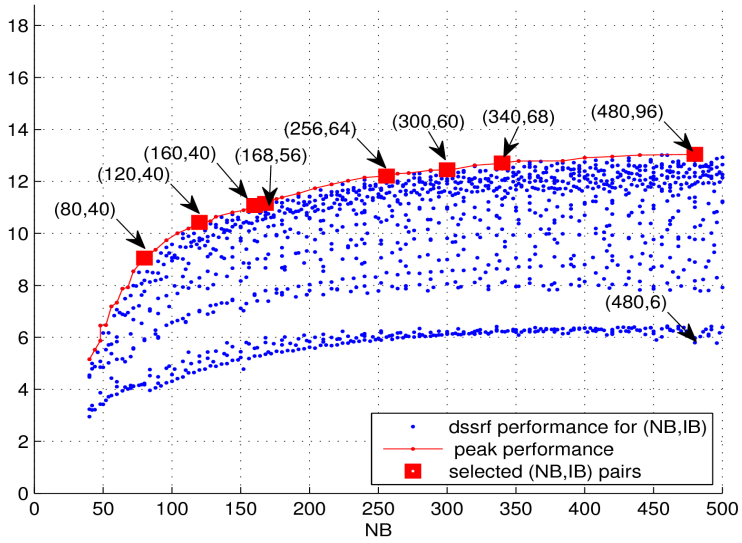
[1st model-driven to limit the search space for a 2nd empirical part]
[Another aspect is adaptivity – to treat cases where tuning can not be restricted to optimizations at design, installation, or compile time]

Pruning the Search Space

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



Intel 64 – dgemm

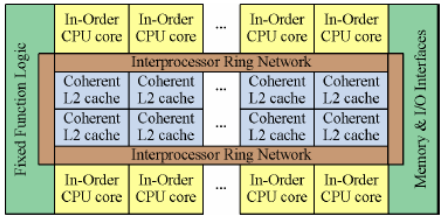


Power 6 – dssrfb

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

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?

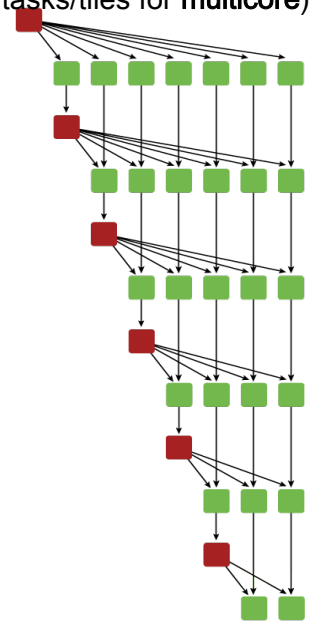


Intel Larrabee

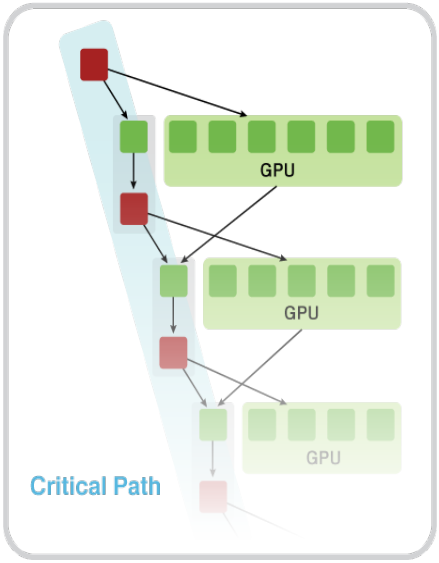
Hybrid Computing

- **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**)



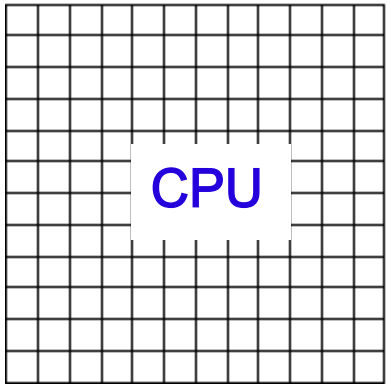
Current hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)



- **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**

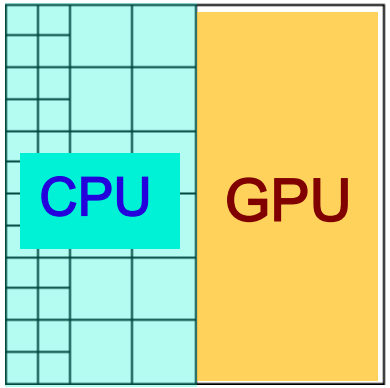
Task Splitting, Scheduling, and Data Storage

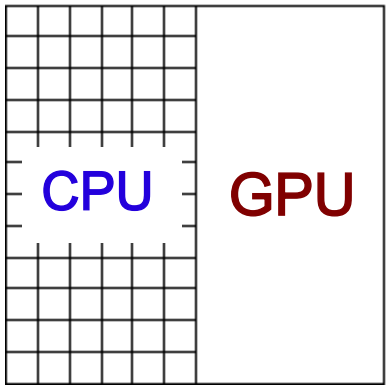
- **Task splitting**
  - **Easy : the splitting itself**
    - splitting BLAS
  - **Difficult: task granularity**
    - to be **dynamic** and **heterogeneous**
    - Ideally: scheduler to **agglomerate** small tasks into large tasks for GPUs



Homogeneous tiles for multicores (granularity is empirically tuned)

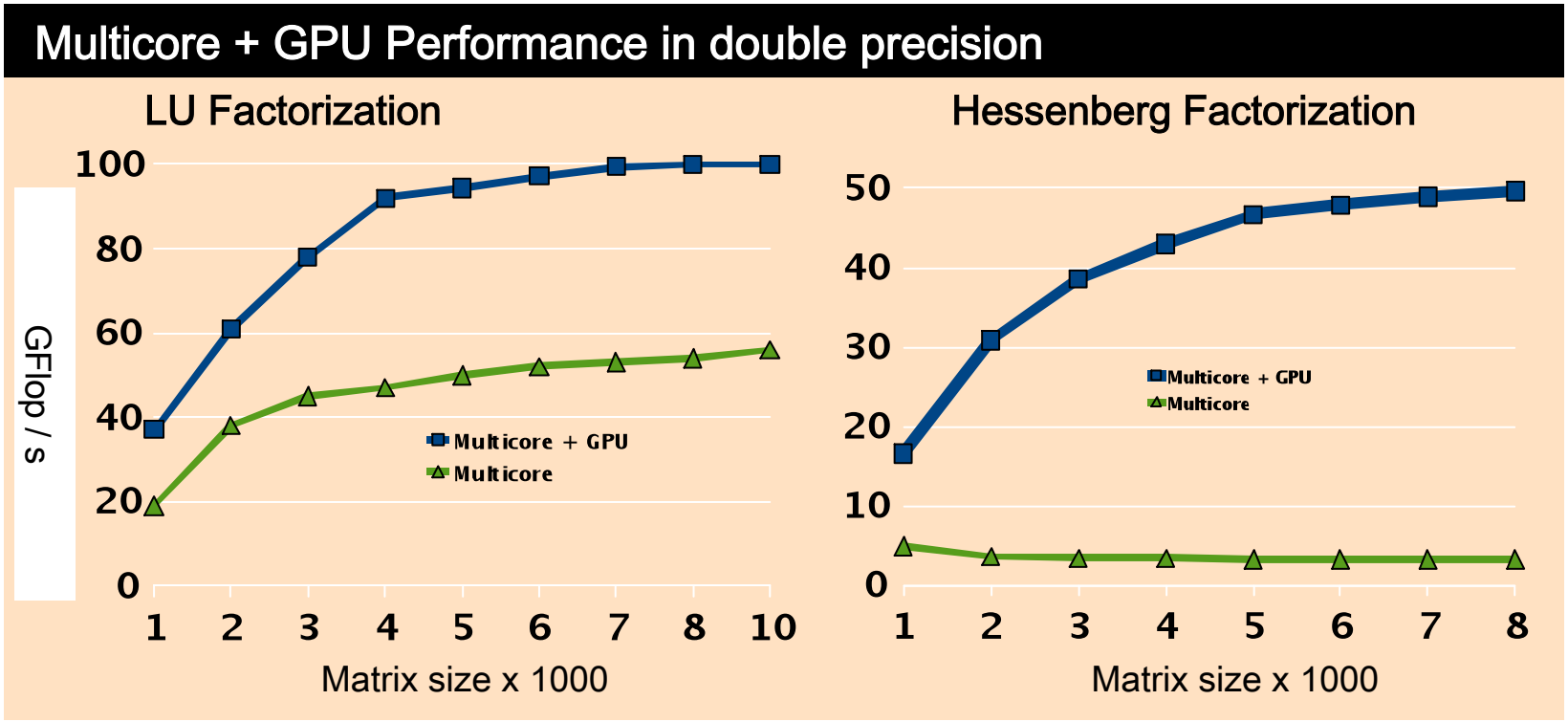
Currently:
- Multi-level blocking for the panels on the CPU
- Tiles are coarse level size (empirically tuned)
- affinity for GPUs and the sub-matrices that they correspondingly modify (to minimize communication)





Agglomerated tasks for GPUs

One and **two-sided** Multicore+GPU Factorizations

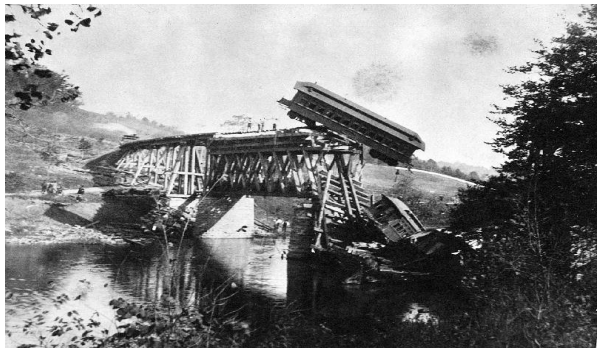


64 bit fl pt; NVIDIA's GeForce GTX 280 GPU and dual socket quad-core Intel Xeon 2.33 GHz

- These will be included in up-coming MAGMA releases
- Two-sided factorizations can not be efficiently accelerated on homogeneous x86-based multicores (above) because of memory-bound operations
  - we developed hybrid algorithms that overcome those bottlenecks (16x speedup!)

Fault Tolerance

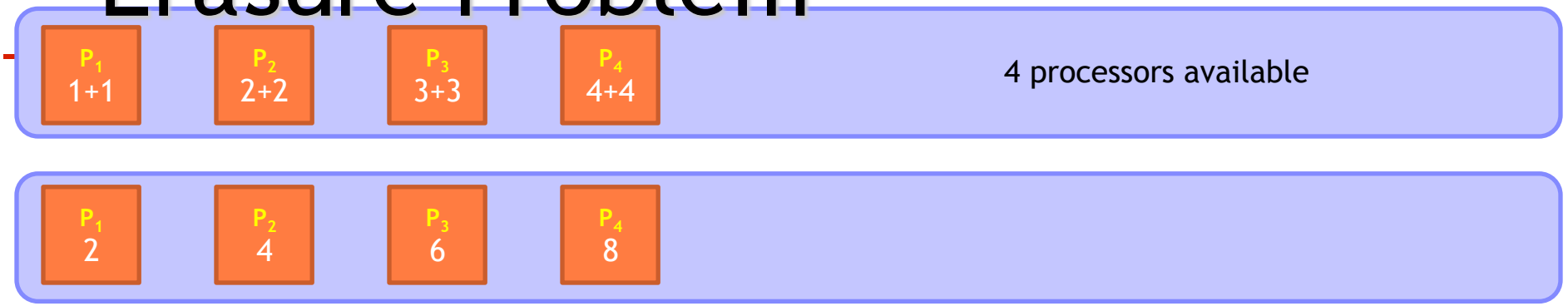
- Trends in HPC:
  - High end systems with thousand of processors.
- Increased probability of a system. failure
  - Most nodes today are robust, 3 year life.
  - Mean Time to Failure is growing shorter as systems grow and devices shrink.



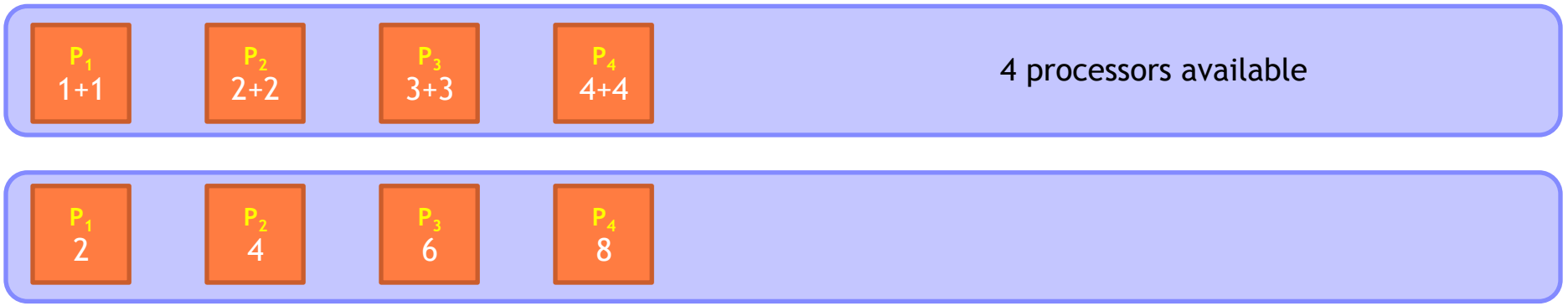
- MPI widely accepted in scientific computing.
  - Process faults not tolerated in MPI model.

Mismatch between hardware and (non fault-tolerant) programming paradigm of MPI.

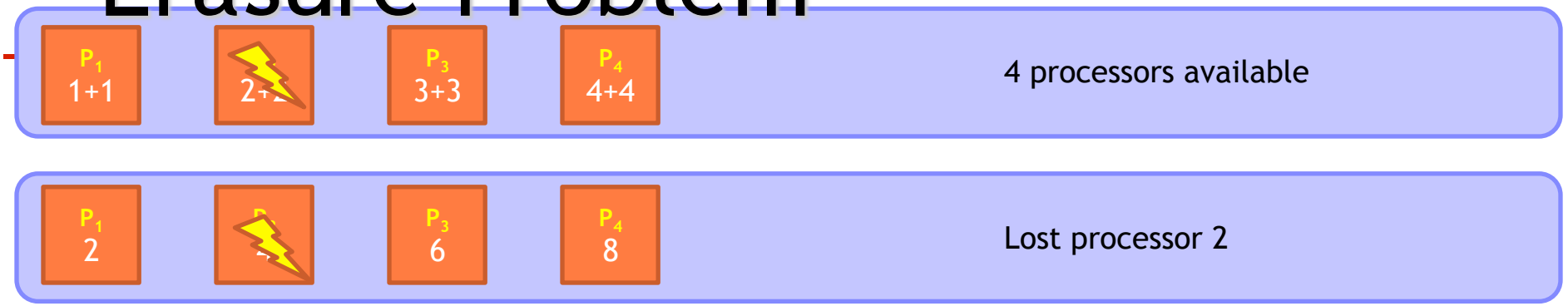
Erasure Problem



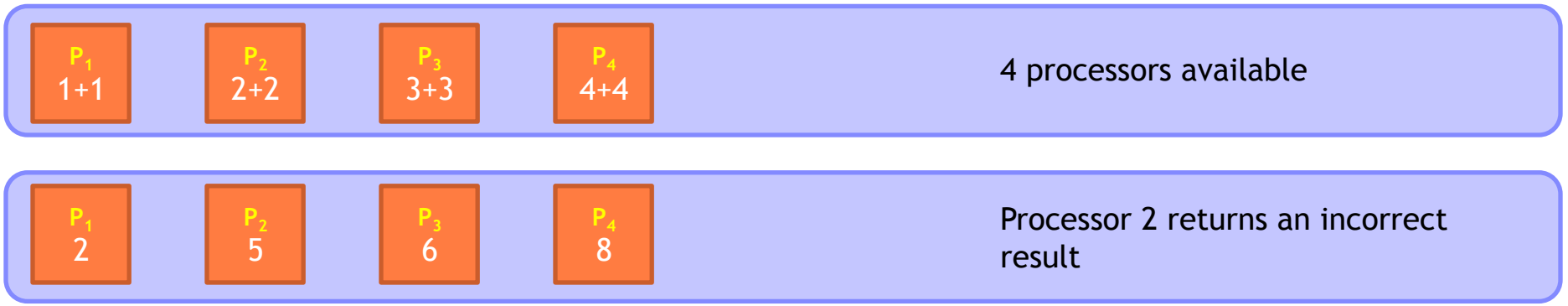
Error Problem



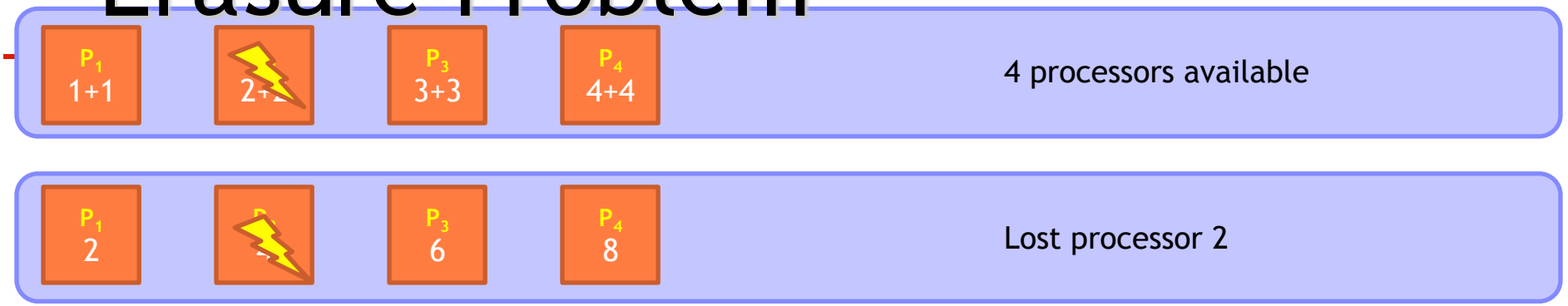
Erasure Problem



Error Problem

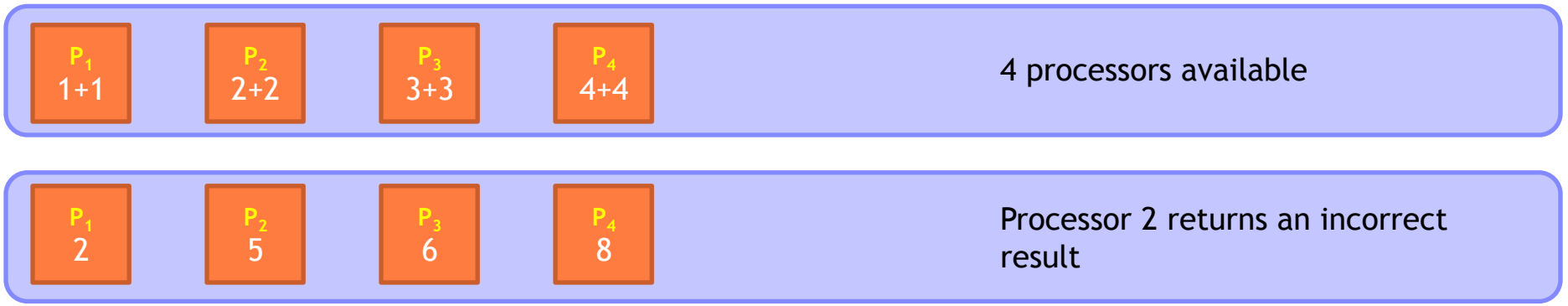


Erasure Problem



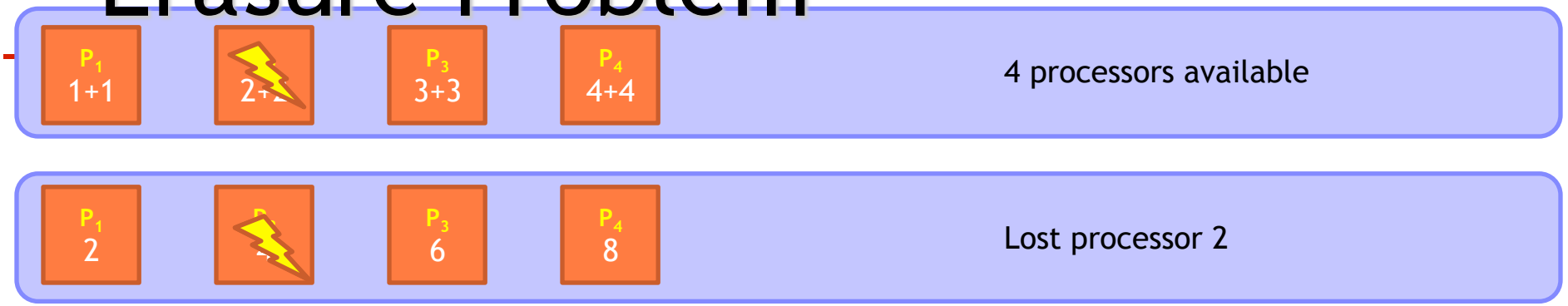
- we know whether there is an erasure or not,

Error Problem



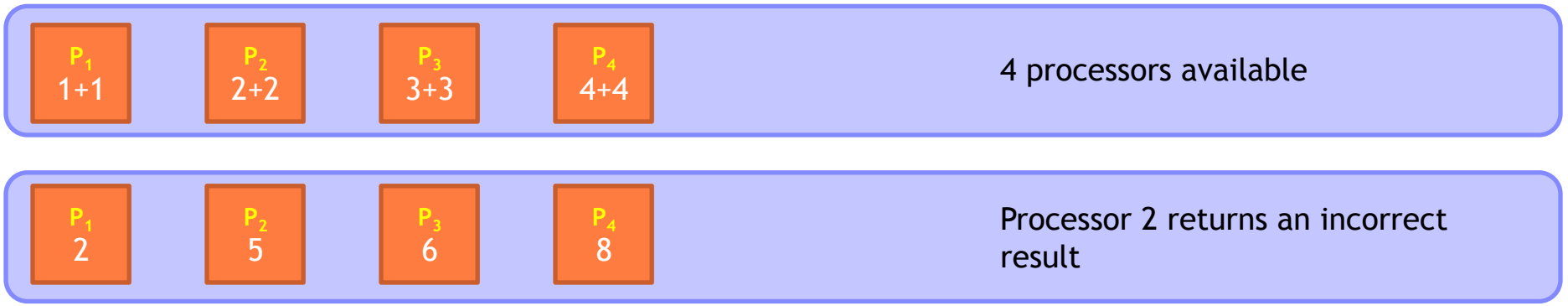
- we do not know if there is an error,

Erasure Problem



- we know whether there is an erasure or not,
- we know where the erasure is,

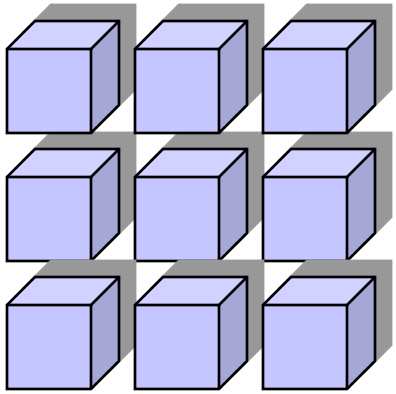
Error Problem



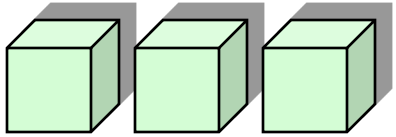
- we do not know if there is an error,
- assuming we know that an error occurs, we do not know where it is

Erasure Code

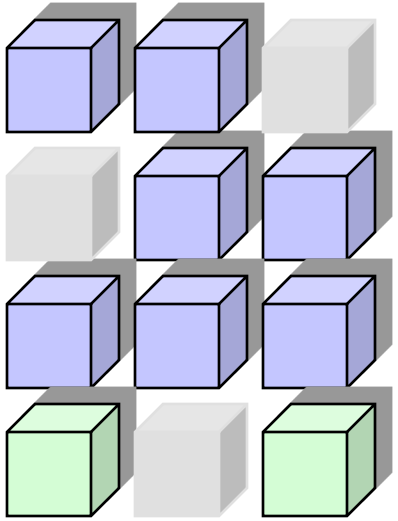
- A technique that lets you take *k* pieces of data:



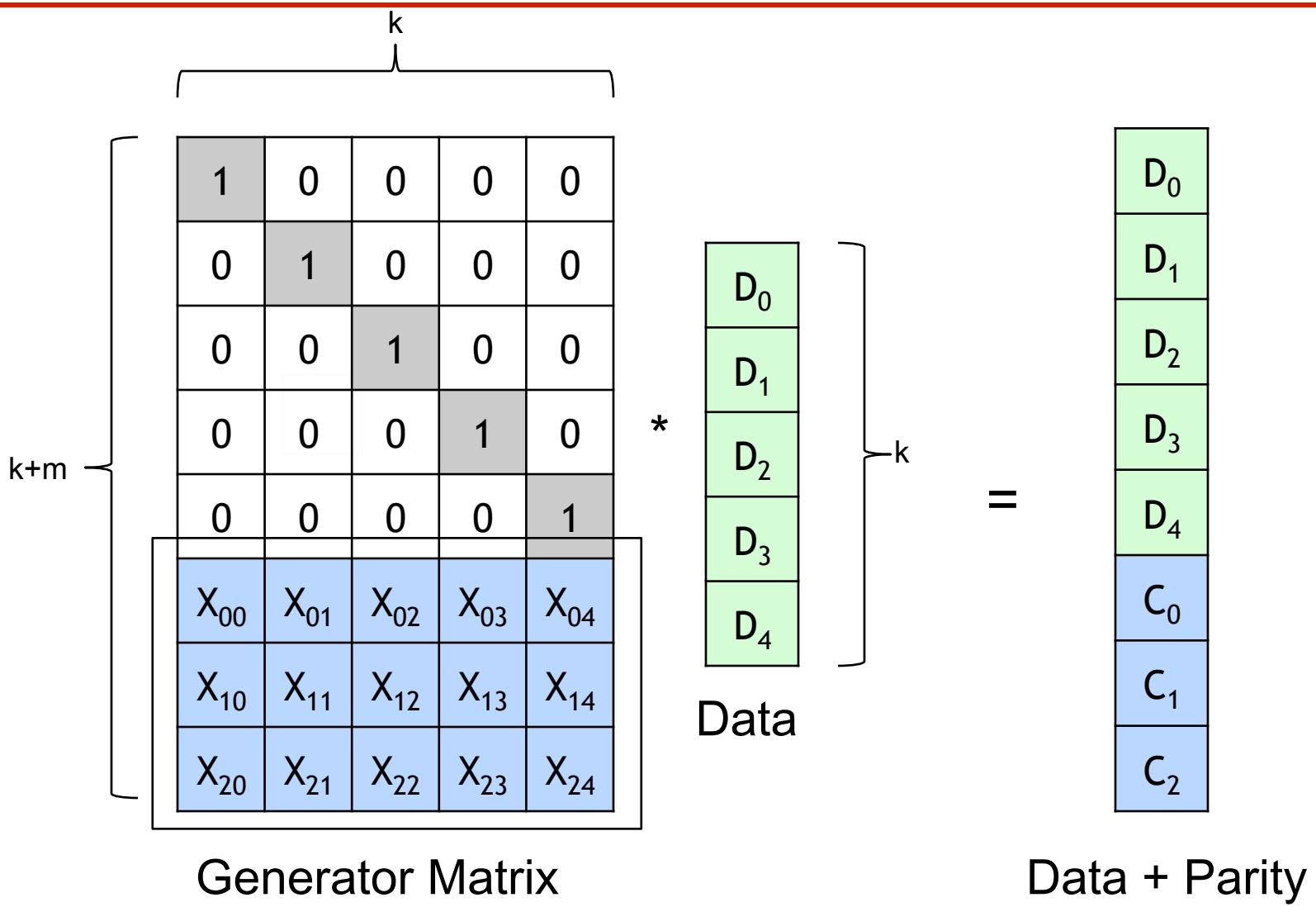
- Encode them into *m* additional pieces of data



- And rebuild the original *k* pieces of data from as few as *k* of the collection



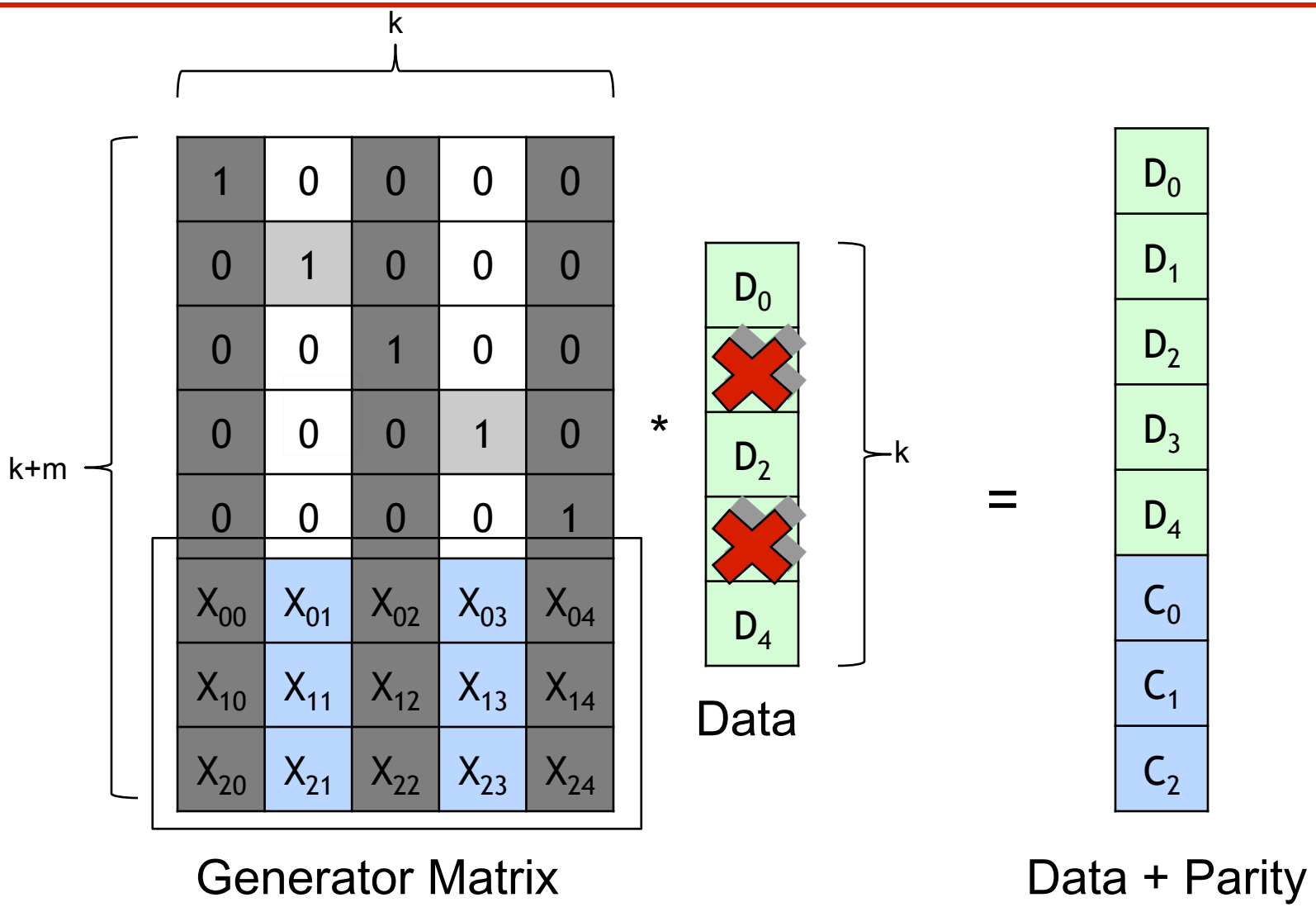
Reed-Solomon Coding



Generator matrix has to such that any square submatrix is non-singular.

Vandermonde, Cauchy matrices, but

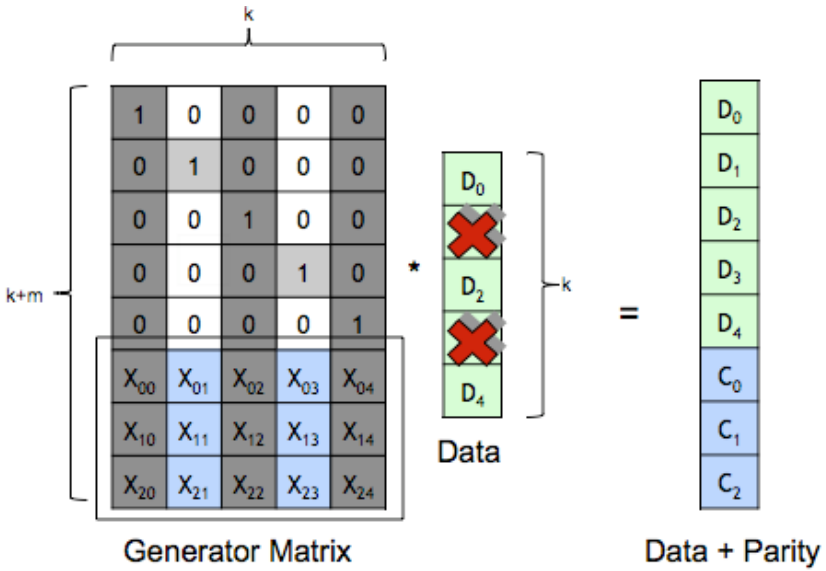
Reed-Solomon Coding



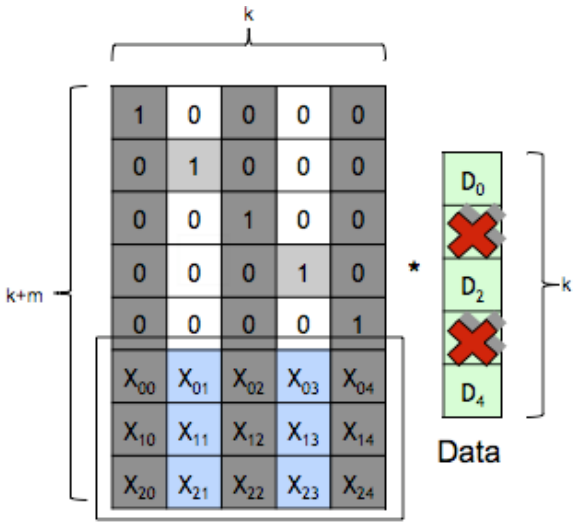
Generator matrix has to such that any square submatrix is non-singular.

Vandermonde, Cauchy matrices, but

Generator matrix has to such that any square submatrix is non-singular.



$$\begin{bmatrix} X_{01} & X_{03} \\ X_{11} & X_{13} \\ X_{21} & X_{23} \end{bmatrix} * \begin{bmatrix} D_1 \\ D_3 \end{bmatrix} = \begin{bmatrix} C_0 \\ C_1 \\ C_2 \end{bmatrix}$$

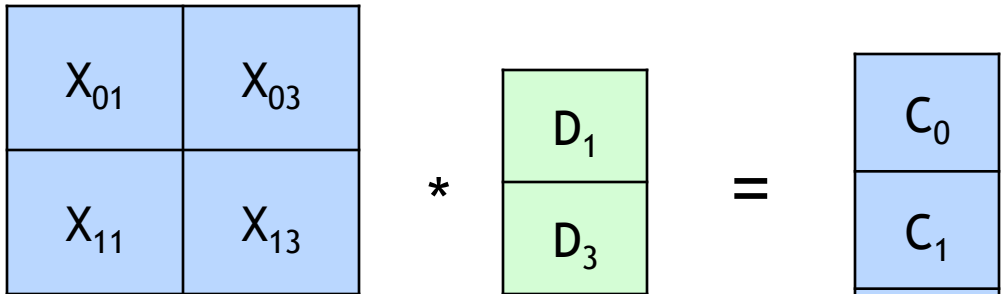


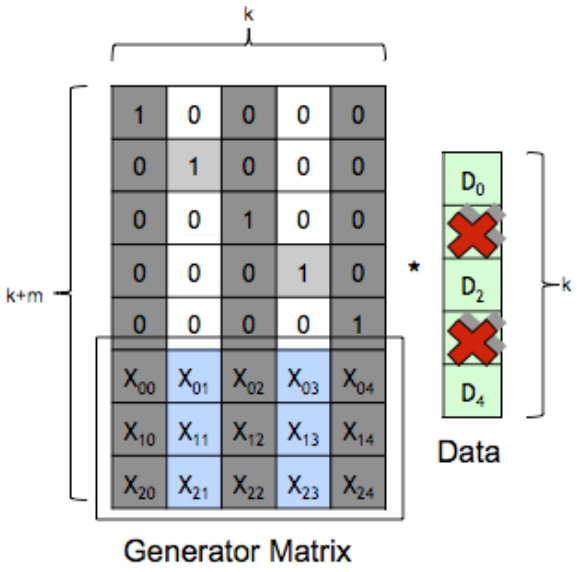
Generator matrix has to such that any square submatrix is non-singular.

Vandermonde, Cauchy matrices, but

$$= \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \dots & \alpha_m^{n-1} \end{bmatrix}$$

$$= \frac{1}{x_i - y_j}; \quad x_i - y_j \neq 0, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n$$





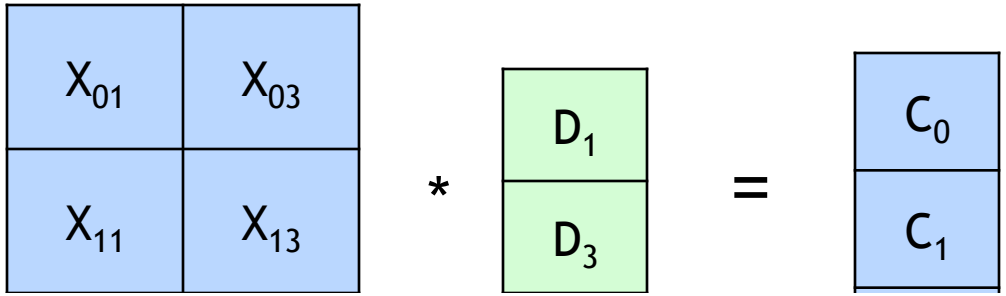
Generator matrix has to such that any square submatrix is non-singular.

Vandermonde, Cauchy matrices, but

$$= \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \dots & \alpha_m^{n-1} \end{bmatrix}$$

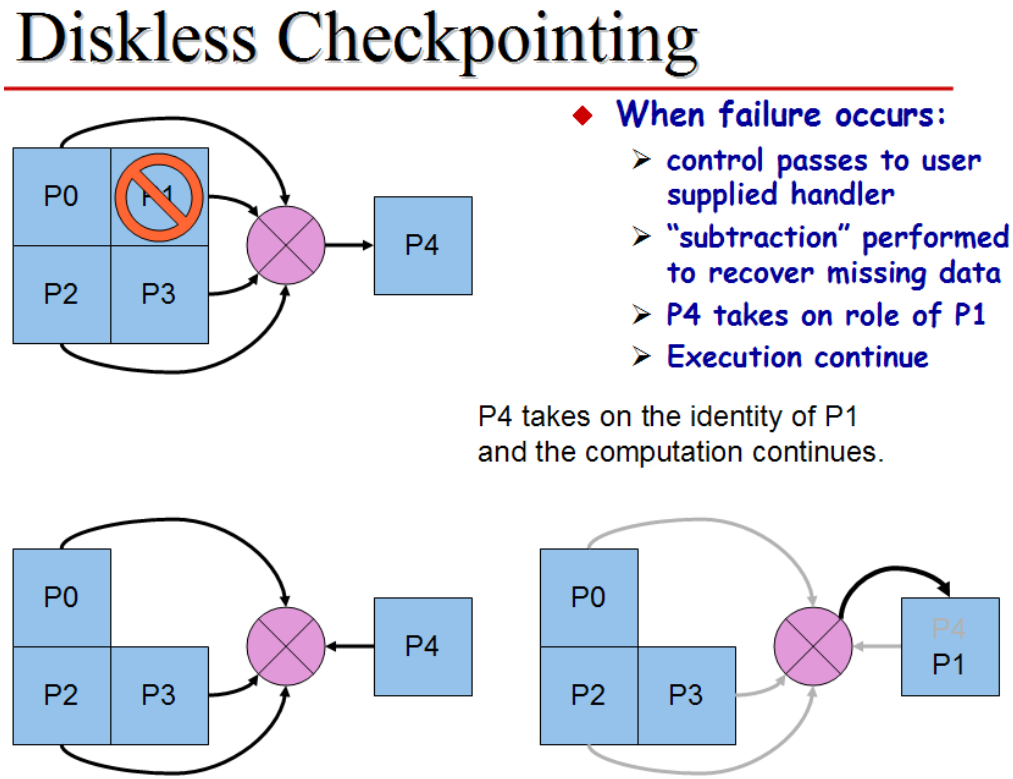
$$= \frac{1}{x_i - y_j}; \quad x_i - y_j \neq 0, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n$$

We use a random matrix for the X part of the generator matrix



Three Ideas for Fault Tolerant Linear Algebra Algorithms

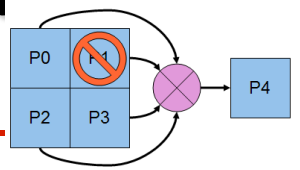
- **Lossless diskless check-pointing for iterative methods**
  - **Checksum maintained in active processors**
  - **On failure, roll back to checkpoint and continue**
  - **No lost data**



Three Ideas for Fault Tolera Linear Algebra Algorithms

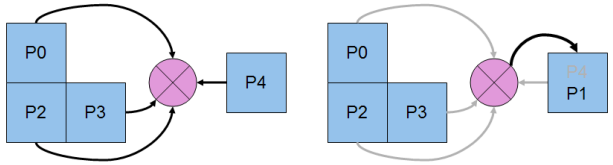
- Lossless diskless check-pointing for iterative methods
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data
- **Lossy approach for iterative methods**
  - **No checkpoint for computed data maintained**
  - **On failure, approximate missing data and carry on**
  - **Lost data but use approximation to recover**

Diskless Checkpointing



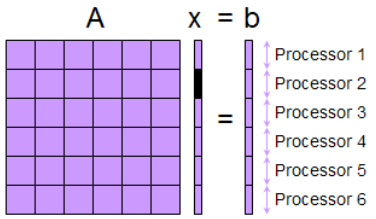
- When failure occurs:
  - control passes to user supplied handler
  - "subtraction" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

P4 takes on the identity of P1 and the computation continues.



Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows



3 steps

Step 1: recover a processor and a running parallel environment (the job of the FT-MPI library)

Step 2: recover A_{21} A_{22}, …, A_{n2} and b_2 (the original data) on the failed processor

Step 3: Notice that

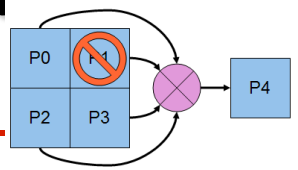
$$A_{21} x_1 + A_{22} x_2 + \ldots + A_{2n} x_n = b_2 \Rightarrow$$

$$x_2 = A_{22}^{-1} (b_2 - \Sigma_{i \neq 2} A_{2i} x_i)$$

Three Ideas for Fault Tolera Linear Algebra Algorithms

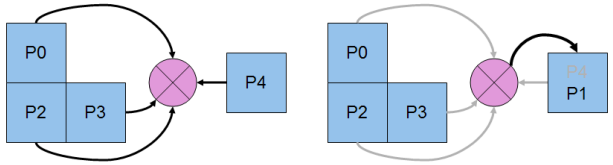
- Lossless diskless check-pointing for iterative methods
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data
- Lossy approach for iterative methods
  - No checkpoint maintained
  - On failure, approximate missing data and carry on
  - Lost data but use approximation to recover
- **Check-pointless methods for dense algorithms**
  - **Checksum maintained as part of computation**
  - **No roll back needed; No lost data**

Diskless Checkpointing



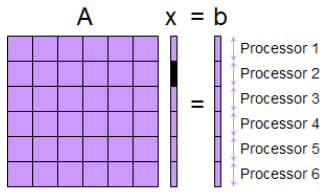
- When failure occurs:
  - control passes to user supplied handler
  - "subtraction" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

P4 takes on the identity of P1 and the computation continues.



Lossy Algorithm : Basic Idea

- Let us assume that the exact solution of the system Ax=b is stored on different processors by rows



3 steps

Step 1: recover a processor and a running parallel environment (the job of the FT-MPI library)

Step 2: recover A_{21} A_{22}, ..., A_{n2} and b_2 (the original data) on the failed processor

Step 3: Notice that

$$A_{21} x_1 + A_{22} x_2 + \ldots + A_{2n} x_n = b_2 \Rightarrow$$

An Example: ScaLAPACK/PBLAS Matrix Multiplication

$$\begin{pmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & \cdots & \vdots \\ A_{p1} & \cdots & A_{pq} \\ \sum_{i=1}^{p} A_{i1} & \cdots & \sum_{i=1}^{p} A_{iq} \end{pmatrix} * \begin{pmatrix} B_{11} & \cdots & B_{1p} & \sum_{j=1}^{p} B_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ B_{q1} & \cdots & B_{qp} & \sum_{j=1}^{p} B_{qj} \end{pmatrix}$$

$$= \begin{pmatrix} C_{11} & \cdots & C_{1p} & \sum_{j=1}^{p} C_{1j} \\ \vdots & \cdots & \vdots & \vdots \\ C_{p1} & \cdots & C_{pp} & \sum_{j=1}^{p} C_{pj} \\ \sum_{i=1}^{p} C_{i1} & \cdots & \sum_{k=1}^{p} C_{ip} & \sum_{i=1}^{p} \sum_{j=1}^{p} C_{ij} \end{pmatrix}$$

- Single failure during computation can be recovered from the checksum relationship
- By using a floating-point version Reed-Solomon code, multiple failures can be tolerated

Exascale Computing

Google: exascale computing study

ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

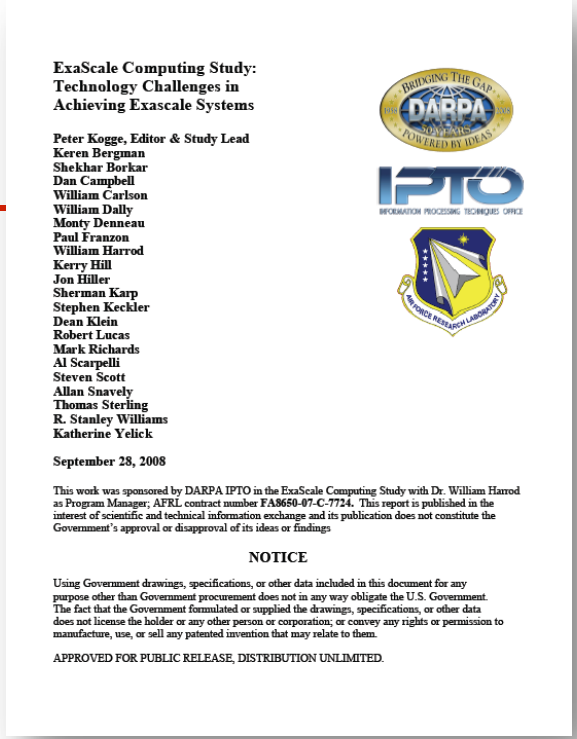
Peter Kogge, Editor & Study Lead
Keren Bergman
Shekhar Borkar
Dan Campbell
William Carlson
William Dally
Monty Denneau
Paul Franzon
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Stephen Keckler
Dean Klein
Robert Lucas
Mark Richards
Al Scarpelli
Steven Scott
Allan Snavely
Thomas Sterling
R. Stanley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

- 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

Google: exascale computing study

How will we program them

- Still an unsolved problem
- Some believe a totally new programming model and language (x10, Chapel, Fortress)
- The MPI specification and MPI implementations can both be made more scalable
- Some mechanism for dealing with shared memory will probably be necessary
  - This (whatever it is) plus MPI is the conservative view
- Whatever it is, it will need to interact properly with MPI
- May also need to deal with on-node heterogeneity
- The situation is somewhat like message-passing before MPI
  - And it is too early to standardize

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

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