Making Dataflow Programming Ubiquitous for Scientific Computing

Hatem Ltaief
KAUST Supercomputing Lab

Synchronization-reducing and Communication-reducing Algorithms and Programming Models for Large-scale Simulations

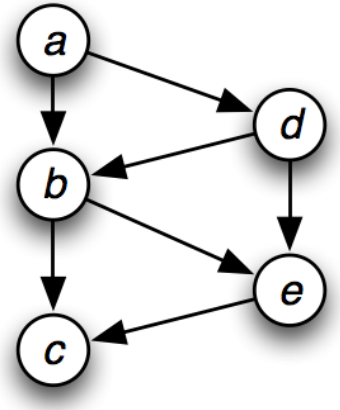
January 9-13, 2012
Providence, RI USA

Outline

Plan

DataFlow Programming

- Five decades **OLD** concept
- Programming paradigm that models a program as a directed graph of the data flowing between operations (cf. Wikipedia)
- Think "how things connect" rather than "how things happen"
- *Assembly line*
- Inherently parallel



What did they say?

- **Katherine Yelick**: – High level abstraction optimizations e.g, in the context of linear algebra, leverage BLAS optimizations to the whole numerical algorithm. – Load balancing is paramount, especially in sparse linear algebra computations. – Locality is critical when computational intensity low and memory hierarchies are deep.
- **Victor Eijkhout**: – Integrative Model for Parallelism design: describe parallel algorithms based on explicit partitioning of input and output data. – MPI instruction commands are encapsulated into derivable objects. No need for direct MPI user coding $\Longrightarrow$ productivity!
- **Jonathan Cohen**: – Expose as much as possible fine-grain parallelism to exploit the underlying hardware components.

Plan

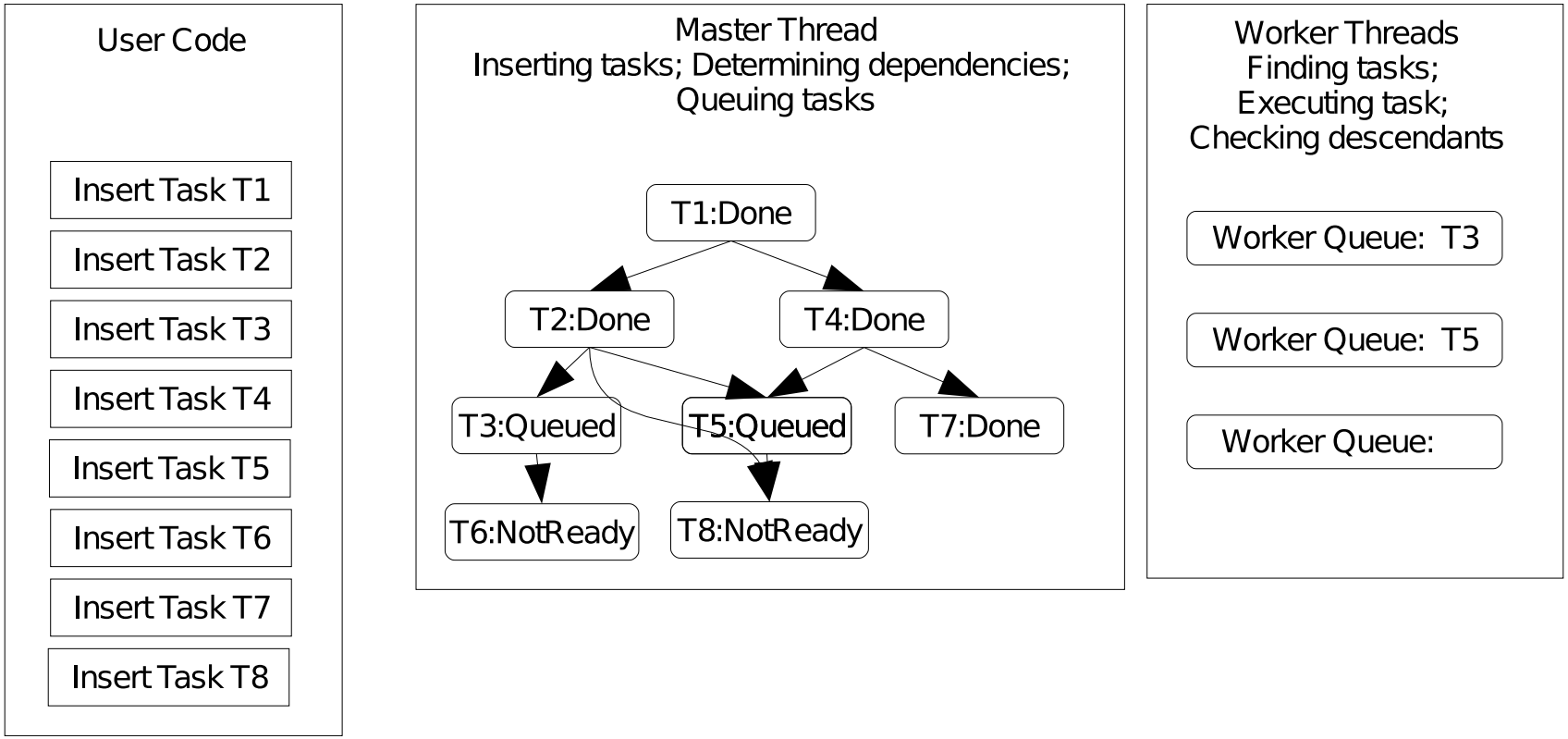
Matrices Over Runtime Systems at Exascale

- Mission statement: *"Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems"*.
- Runtime challenges due to the ever growing hardware complexity.
- Algorithmic challenges to exploit the hardware capabilities at most.

QUARK

- From Sequential Nested-Loop Code to Parallel Execution
- Task-based parallelism
- Out-of-order dynamic scheduling
- Scheduling a Window of Tasks
- Data Locality and Cache Reuse
- High user-productivity
- Shipped within PLASMA but standalone project

QUARK

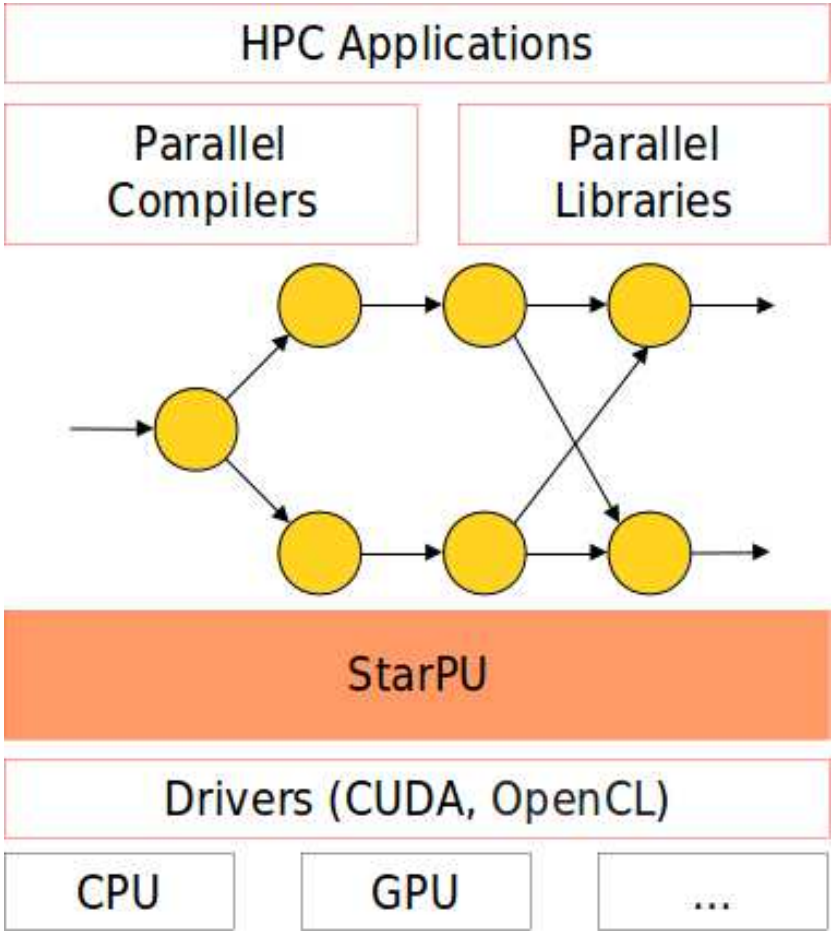

User Code
Insert Task T1
Insert Task T2
Insert Task T3
Insert Task T4
Insert Task T5
Insert Task T6
Insert Task T7
Insert Task T8
Master Thread
Inserting tasks; Determining dependencies;
Queuing tasks
T1:Done
T2:Done
T4:Done
T3:Queued
T5:Queued
T7:Done
T6:NotReady
T8:NotReady
Worker Threads
Finding tasks;
Executing task;
Checking descendants
Worker Queue: T3
Worker Queue: T5
Worker Queue:

QUARK

```
int QUARK_core_dpotrf( Quark *quark, char uplo, int n,
                       double *A, int lda, int *info )
{
  QUARK_Insert_Task( quark, TASK_core_dpotrf, 0x00,
    sizeof(char),          &uplo,     VALUE,
    sizeof(int),           &n,        VALUE,
    sizeof(double)*n*n,    A,         INOUT | LOCALITY,
    sizeof(int),           &lda,      VALUE,
    sizeof(int),           info,      OUTPUT,
    0);
}
void TASK_core_dpotrf(Quark *quark)
{
  char uplo; int n; double *A; int lda;  int *info;
  quark_unpack_args_5( quark, uplo, n, A, lda, info );
  dpotrf_( &uplo, &n, A, &lda, info );
}
```

StarPU

- RunTime which provides:
  $\Longrightarrow$ Task scheduling
  $\Longrightarrow$ Memory management
- Supports:
  $\Longrightarrow$ SMP/Multicore Processors (x86, PPC, ...)
  $\Longrightarrow$ NVIDIA GPUs (e.g. heterogeneous multi-GPU)
  $\Longrightarrow$ OpenCL devices
  $\Longrightarrow$ Cell Processors (experimental)



StarPU

```
starpu_Insert_Task(&cl_dpotrf,
    VALUE,   &uplo,          sizeof(char),
    VALUE,   &n,             sizeof(int),
    INOUT,   Ahandle(k, k),
    VALUE,   &lda,           sizeof(int),
    OUTPUT,  &info,          sizeof(int)
    CALLBACK, profiling?cl_dpotrf_callback:NULL, NULL,
    0);
```

SMPSs

- Compiler technology.
- Task parameters and directionality defined by the user through pragmas
- Translates C codes with pragma annotations to standard C99 code
- Embedded Locality optimizations
- Data renaming feature to reduce dependencies, leaving only the true dependencies.

SMPSs

```
#pragma css task input(A[NB][NB]) inout(T[NB][NB])
void dsyrk(double *A, double *T);

#pragma css task inout(T[NB][NB])
void dpotrf(double *T);

#pragma css task input(A[NB][NB], B[NB][NB]) inout(C[NB][NB])
void dgemm(double *A, double *B, double *C);

#pragma css task input(T[NB][NB]) inout(B[NB][NB])
void dtrsm(double *T, double *C);


#pragma css start
for (k = 0; k < TILES; k++) {

   for (n = 0; n < k; n++)
      dsyrk(A[k][n], A[k][k]);
   dpotrf(A[k][k]);

   for (m = k+1; m < TILES; m++) {
     for (n = 0; n < k; n++)
         dgemm(A[k][n], A[m][n], A[m][k]);
      dtrsm(A[k][k], A[m][k]);
   }
}
#pragma css finish
```

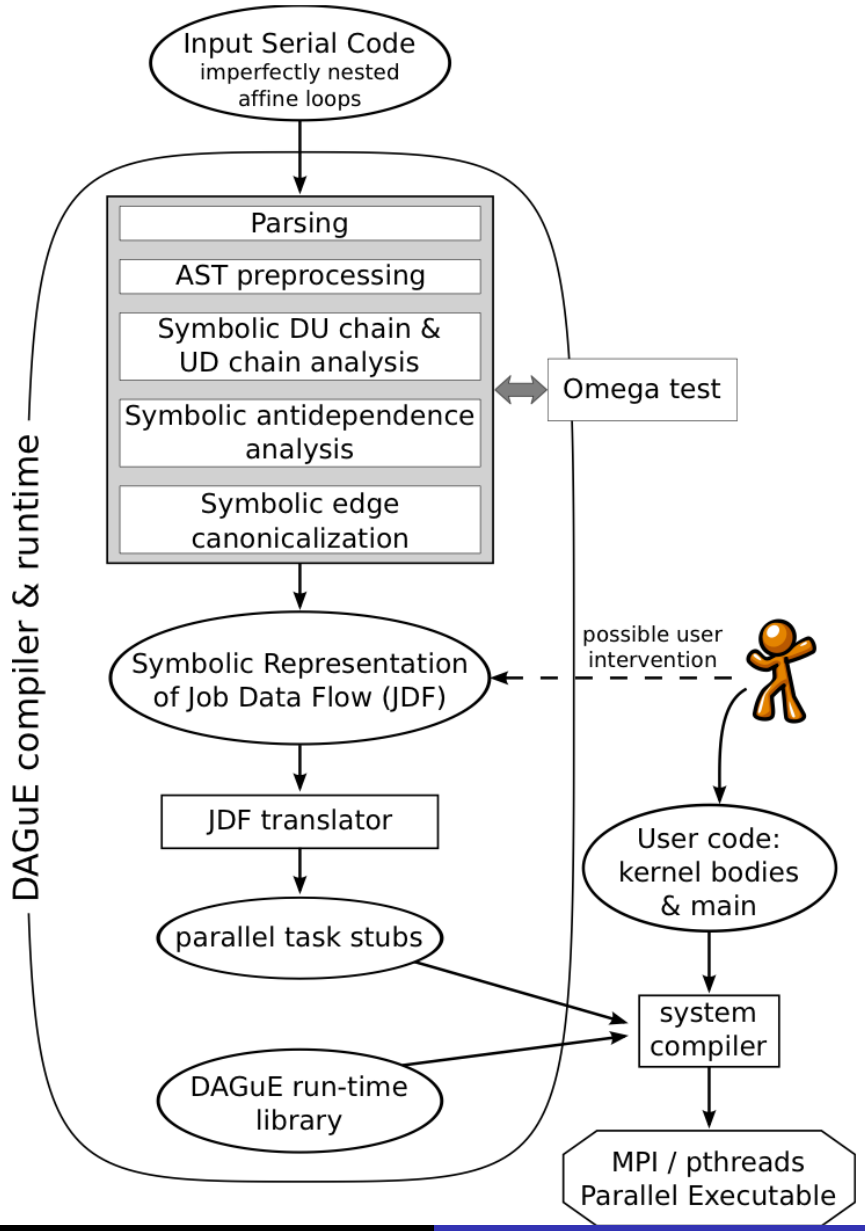
Standardization???

- Efforts to define an API standard for these runtime systems.
- Difficult task...
- But worth the time and sacrifice when it comes to making end users life easier.

DAGuE

- Compiler technology.
- Converting Sequential Code to a DAG representation.
- Parametrized DAG scheduler for distributed memory systems.
- Engine of DPLASMA library

DAGuE


Input Serial Code
imperfectly nested
affine loops
DAGuE compiler & runtime
Parsing
AST preprocessing
Symbolic DU chain &
UD chain analysis
Symbolic antidependence
analysis
Symbolic edge
canonicalization
Omega test
Symbolic Representation
of Job Data Flow (JDF)
possible user
intervention
JDF translator
parallel task stubs
DAGuE run-time
library
User code:
kernel bodies
& main
system
compiler
MPI / pthreads
Parallel Executable

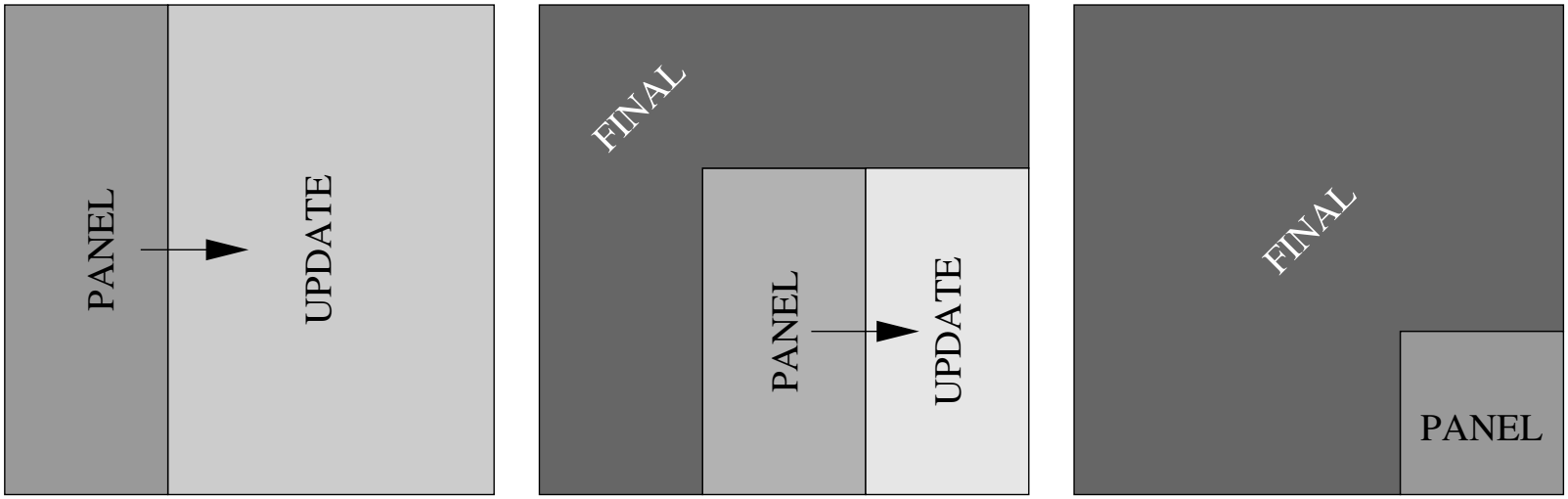
DAGuE

```
DGEQRT(k) executes on A(k, k)
 k = 0..SIZE-1

 V <- (k==0)        ? A(0,0) : C2 DSSMQR(k-1,k,k)
   -> (k==SIZE-1) ? A(k,k) : R DTSQRT(k,k+1)    [U]
   -> (k!=SIZE-1) ? V DORMQR(k, k+1..SIZE-1)    [L]
   -> A(k,k)                                    [L]
 T -> T DORMQR(k, k+1..SIZE-1)                  [T]
   -> T(k,k)                                    [T]

DTSQRT(k,m) executes on A(m, k)
 k = 0..SIZE-2
 m = k+1..SIZE-1

 V <- (k==0) ? A(m,0) : C2 DSSMQR(k-1,k,m)
   -> V DSSMQR(k,k+1..SIZE-1,m)
   -> A(m,k)
 R <- (m==k+1) ? V DGEQRT(k) :
                 R DTSQRT(k,m-1)      [U]
   -> (m==SIZE-1) ? A(k, k) :
                   R DTSQRT(k,m+1)    [U]
 T -> T DSSMQR(k,k+1..SIZE-1,m)       [T]
   -> T(m,k)                          [T]
```

```
DORMQR(k,n)  executes on A(k, n)
 k = 0..SIZE-2
 n = k+1..SIZE-1

 T  <- T DGEQRT(k)                          [T]
 V  <- V DGEQRT(k)                          [L]
 C  <- (k==0) ? A(k,n) : C2 DSSMQR(k-1,n,k)
    -> C1 DSSMQR(k,n,k+1)

DSSMQR(k,n,m) executes on A(m, n)
 k = 0    .. SIZE-2
 n = k+1  .. SIZE-1
 m = k+1  .. SIZE-1

 V  <- V DTSQRT(k,m)
 T  <- T DTSQRT(k,m)                        [T]
 C2 <- (k==0) ? A(m,n) : C2 DSSMQR(k-1,n,m)
    -> (n==k+1 & m==k+1) ? V DGEQRT(k+1)
    -> (n==k+1 & k<m-1) ? V DTSQRT(k+1,m)
    -> (k<n-1 & m==k+1) ? C DORMQR(k+1,n)
    -> (k<n-1 & k<m-1) ? C2 DSSMQR(k+1,n,m)
 C1 <- (m==k+1) ? C DORMQR(k,n) :
                  C1 DSSMQR(k,n,m-1)
    -> (m==SIZE-1) ? A (k,n) : C1 DSSMQR(k,n,m+1)
```

Plan

Blocked Algorithms



(a) First step. (b) Second step. (c) Third step.

Figure: Panel-update sequences for the LAPACK factorizations.

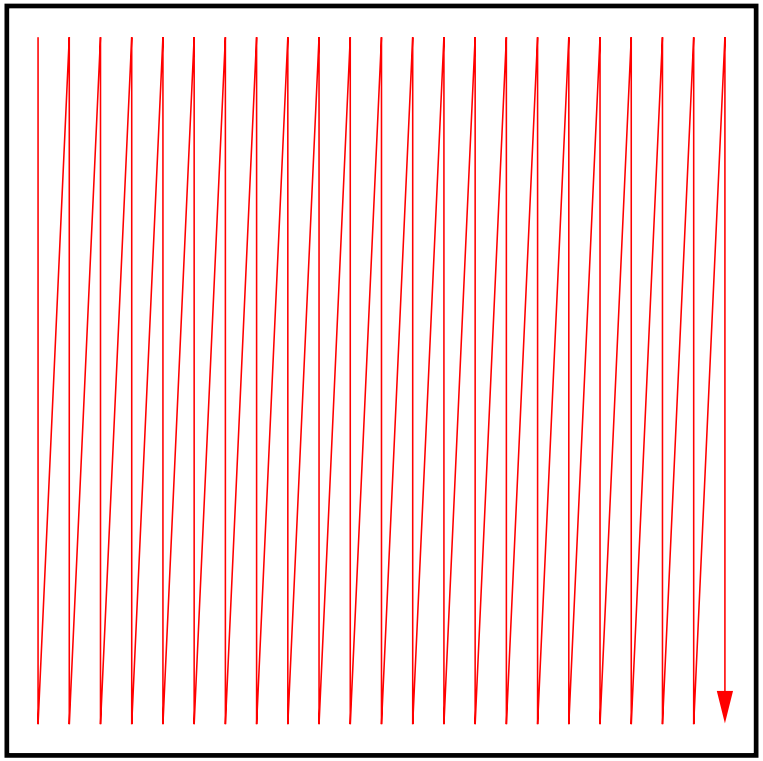
Blocked Algorithms

Principles:

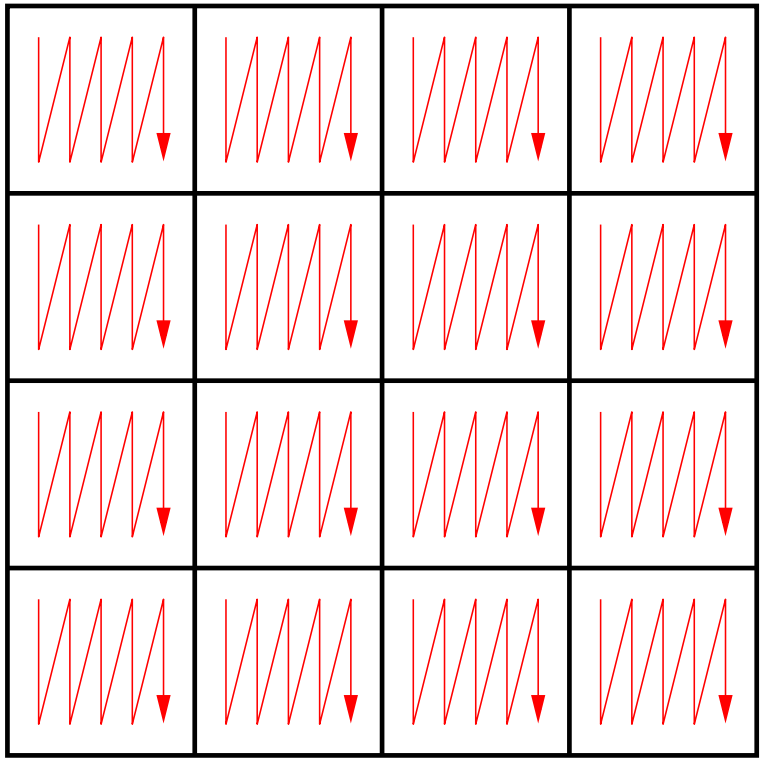
- Panel-Update Sequence
- Transformations are blocked/accumulated within the Panel (Level 2 BLAS)
- Transformations applied at once on the trailing submatrix (Level 3 BLAS)
- Parallelism hidden inside the BLAS
- Fork-join Model

Tile Data Layout Format

LAPACK: column-major format



PLASMA: tile format



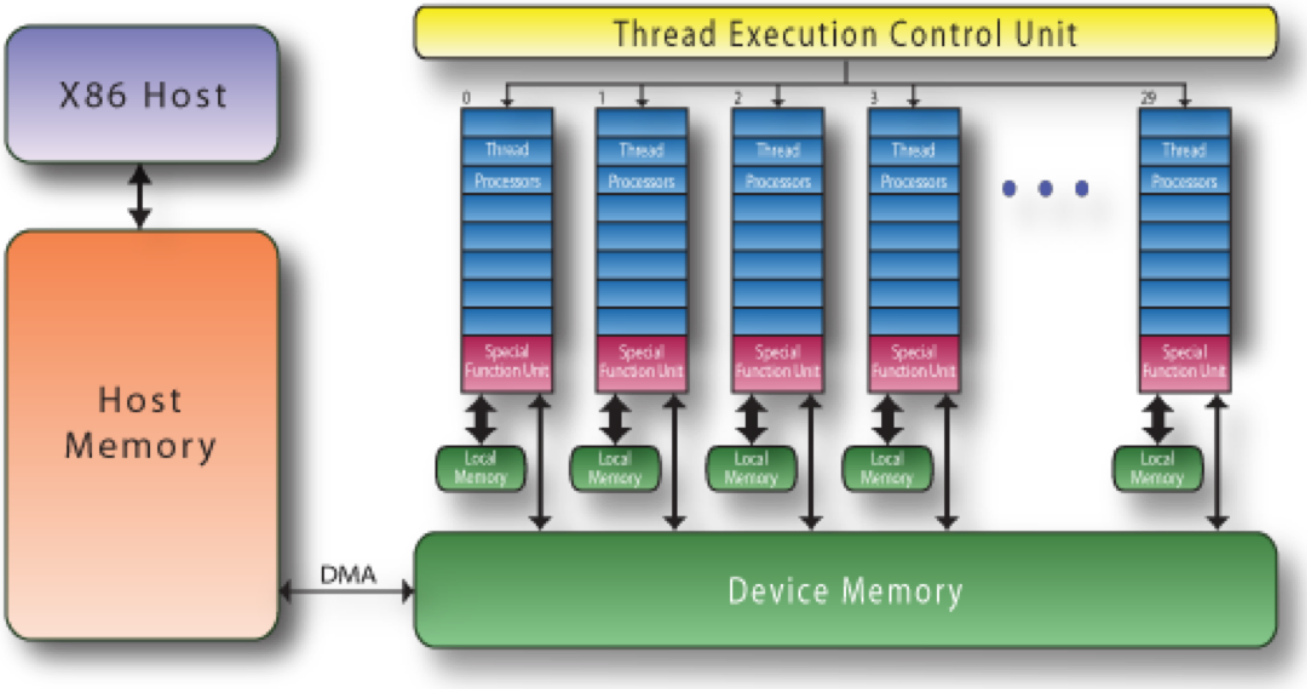
Tile Algorithms

- Parallelism is brought to the fore
- May require the redesign of linear algebra algorithms
- Tile data layout translation
- Remove unnecessary synchronization points between Panel-Update sequences
- DAG execution where nodes represent tasks and edges define dependencies between them
- Feed the dynamic runtime system

A^{-1}, Seriously???

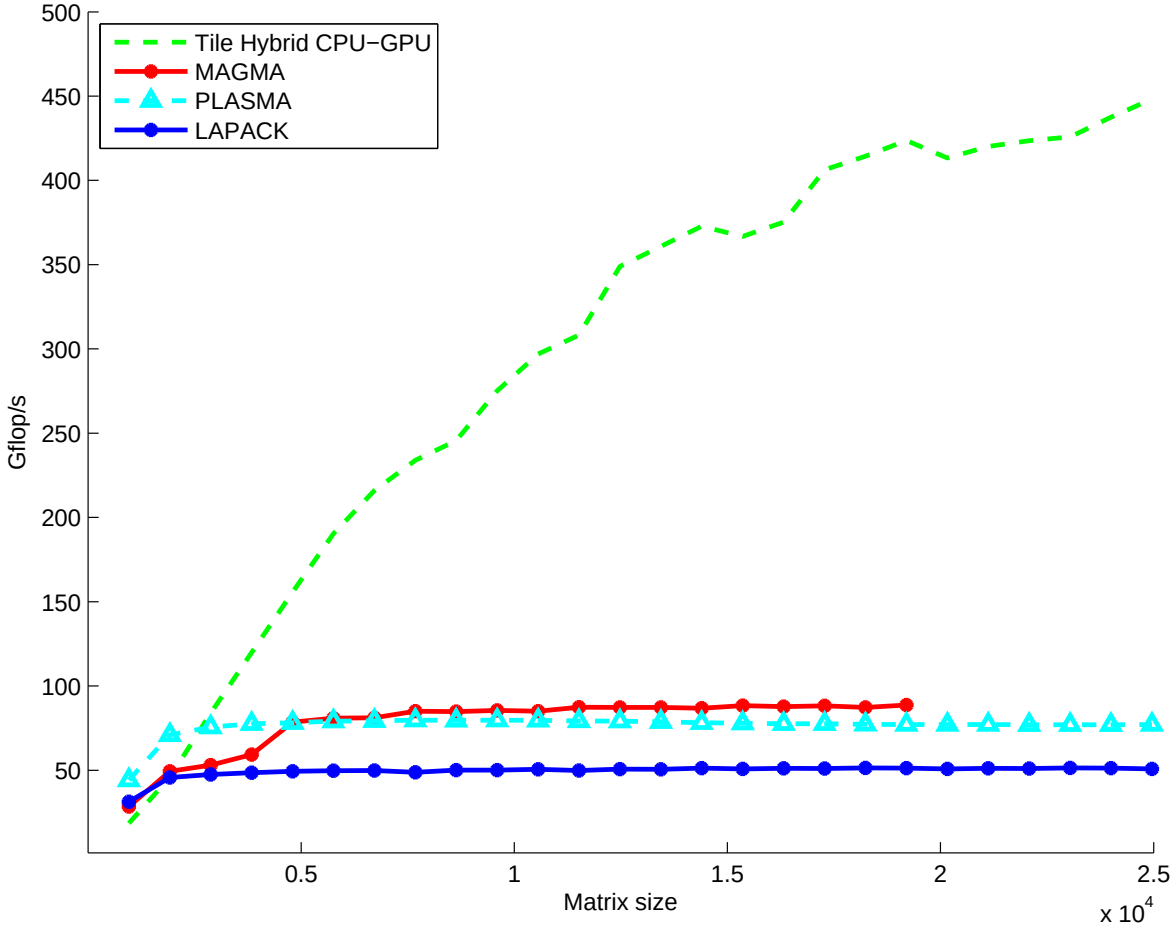
- YES!
- Critical component of the variance-covariance matrix computation in statistics (cf. Higham, Accuracy and Stability of Numerical Algorithms, Second Edition, SIAM, 2002)
- A is a dense symmetric matrix
- Three steps:
  1. Cholesky factorization (DPOTRF)
  2. Inverting the Cholesky factor (DTRTRI)
  3. Calculating the product of the inverted Cholesky factor with its transpose (DLAUUM)
- StarPU runtime used here

A^{-1}, Hybrid Architecture Targeted



- $\Longrightarrow$ PCI Interconnect 16X 64Gb/s, very thin pipe!
- $\Longrightarrow$ Fermi C2050 448 cuda cores 515 Gflop/s

A^{-1}, Preliminary Results



H. Ibeid, D. Kaushik, D. Keyes and H. Ltaief
Student Minisymposium, HIPC'11, India

GSEVP: What we solve?

$$Ax = \lambda Bx$$

$A, B \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}$ or $A, B \in \mathbb{C}^{n\times n}, x \in \mathbb{C}^n$, $\lambda \in \mathbb{R}$

$A = A^T$ or $A = A^H$

$xBx^H > 0$

A is symmetric or Hermitian

B is symmetric positive definite

GSEVP: Why we solve it?

To obtain energy eigenstates in:

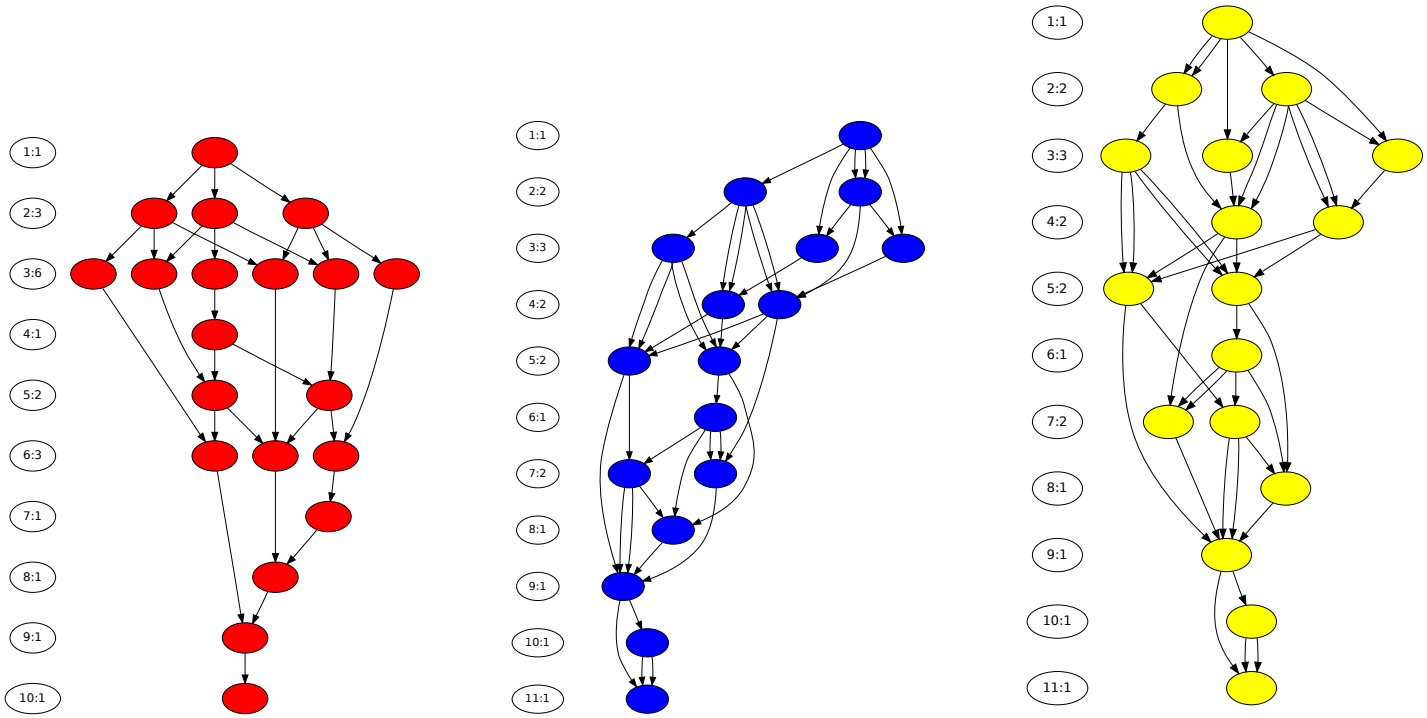
- Chemical cluster theory
- Electronic structure of semiconductors
- Ab-initio energy calculations of solids

GSEVP: How to solve it?

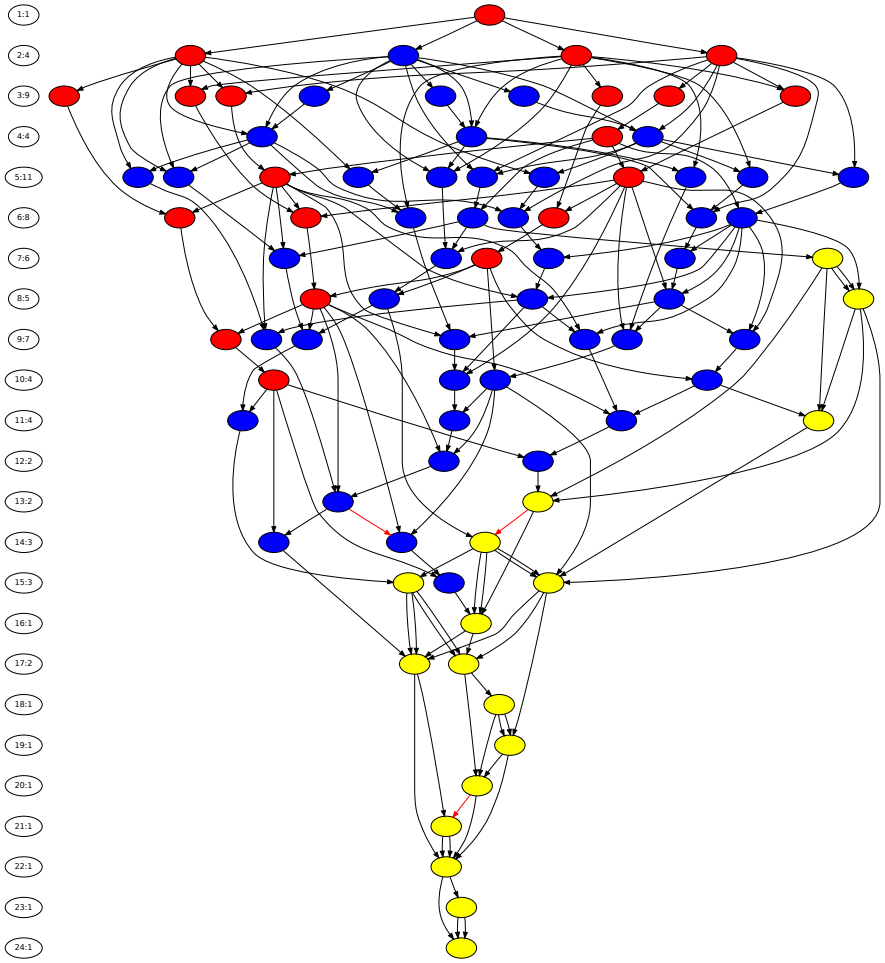
$$Ax = \lambda Bx$$

Operation	Explanation	LAPACK routine name
1. $B = L \times L^T$	Cholesky factorization	POTRF
2. $C = L^{-1} \times A \times L^{-T}$ or HEGST	application of triangular factors	SYGST
3. $T = Q^T \times C \times Q$	tridiagonal reduction	SYEVD or HEEVD
4. $Tx = \lambda x$	QR iteration	STERF

All computational stages: separately

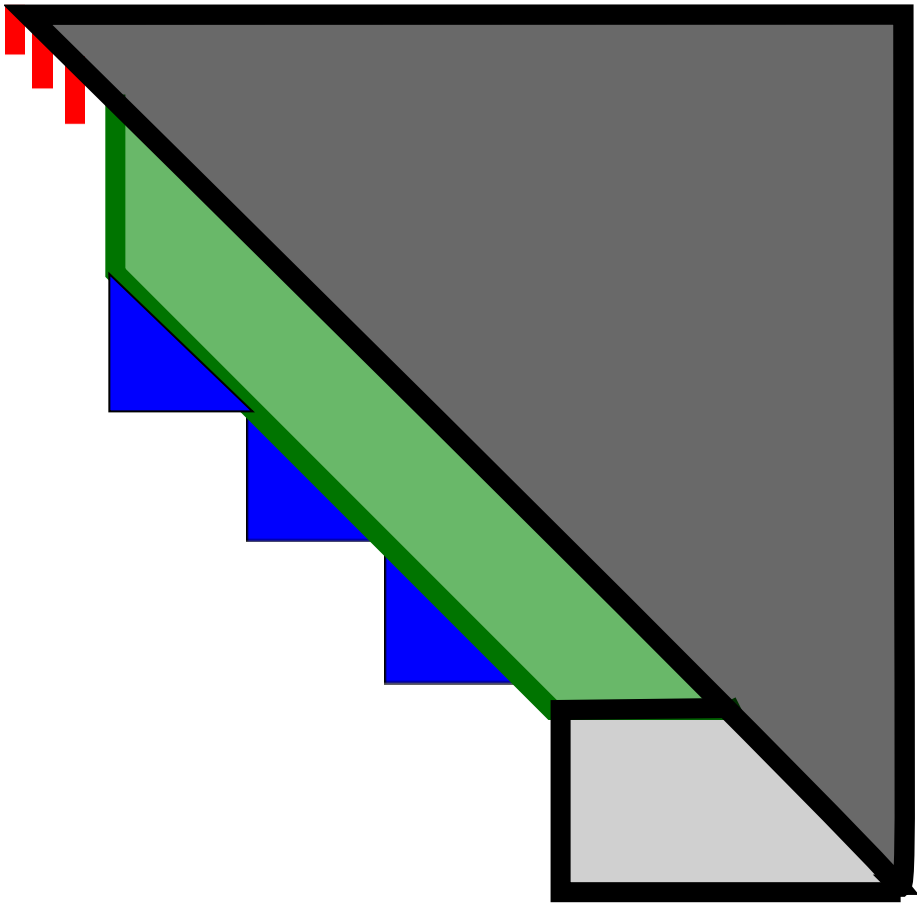


All computational stages: combined

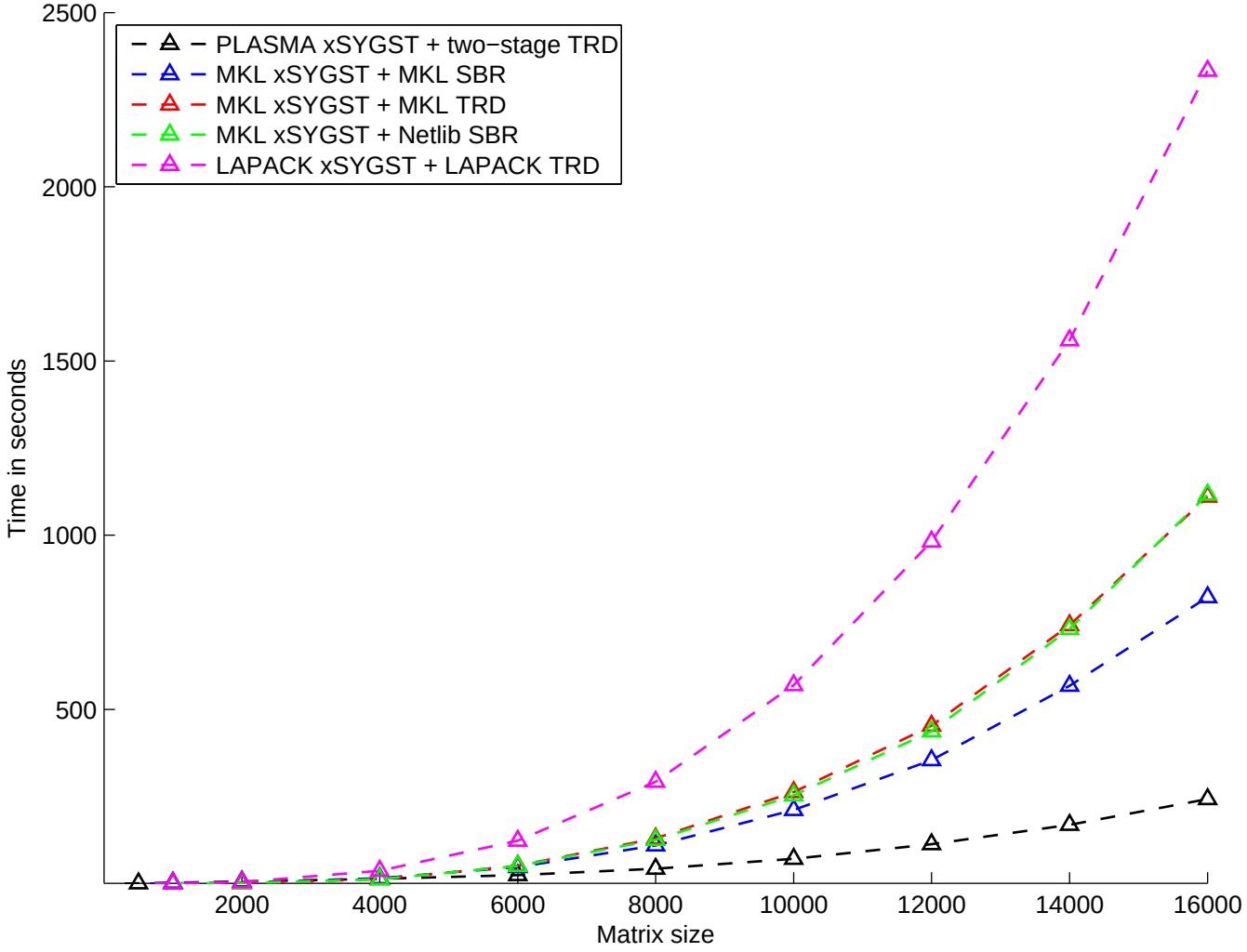


Dependencies are tracked inside PLASMA by QUARK.

Combining stages: matrix view



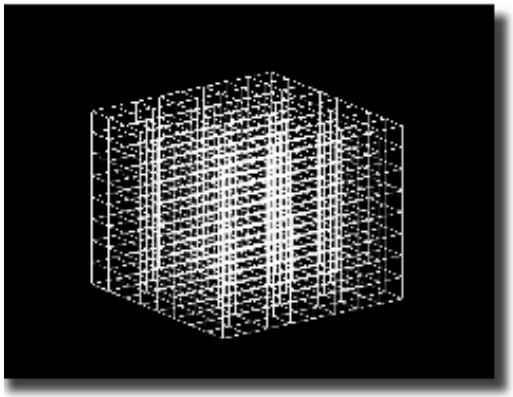
Results on 4-socket AMD Magny Cours (48 cores)



H. Ltaief, P. Luszczek, A. Haidar and J. Dongarra, ParCo'11, Belgium

Plan

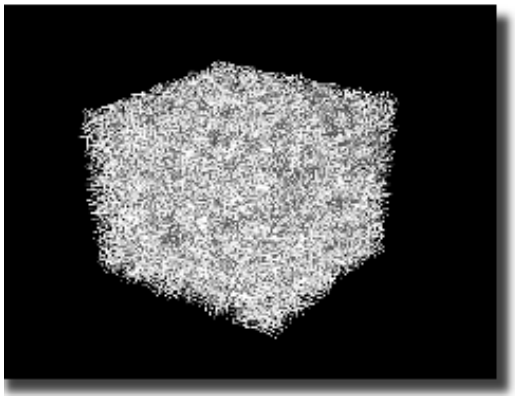
Turbulence Simulations w/ R. Yokota



Pseudo Spectral Method (2/3 dealiasing)

Re_λ: 500

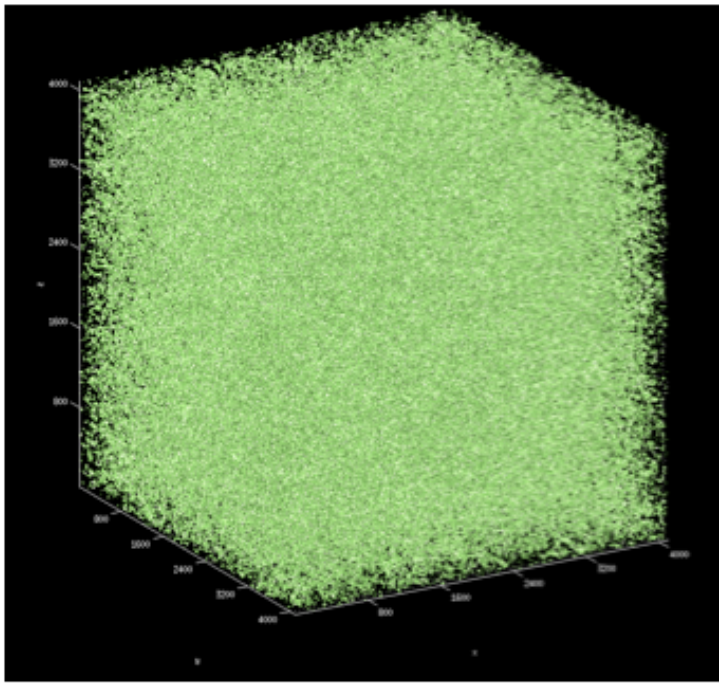
N : 4096^3



Vortex Particle Method (Reinitialized CSM)

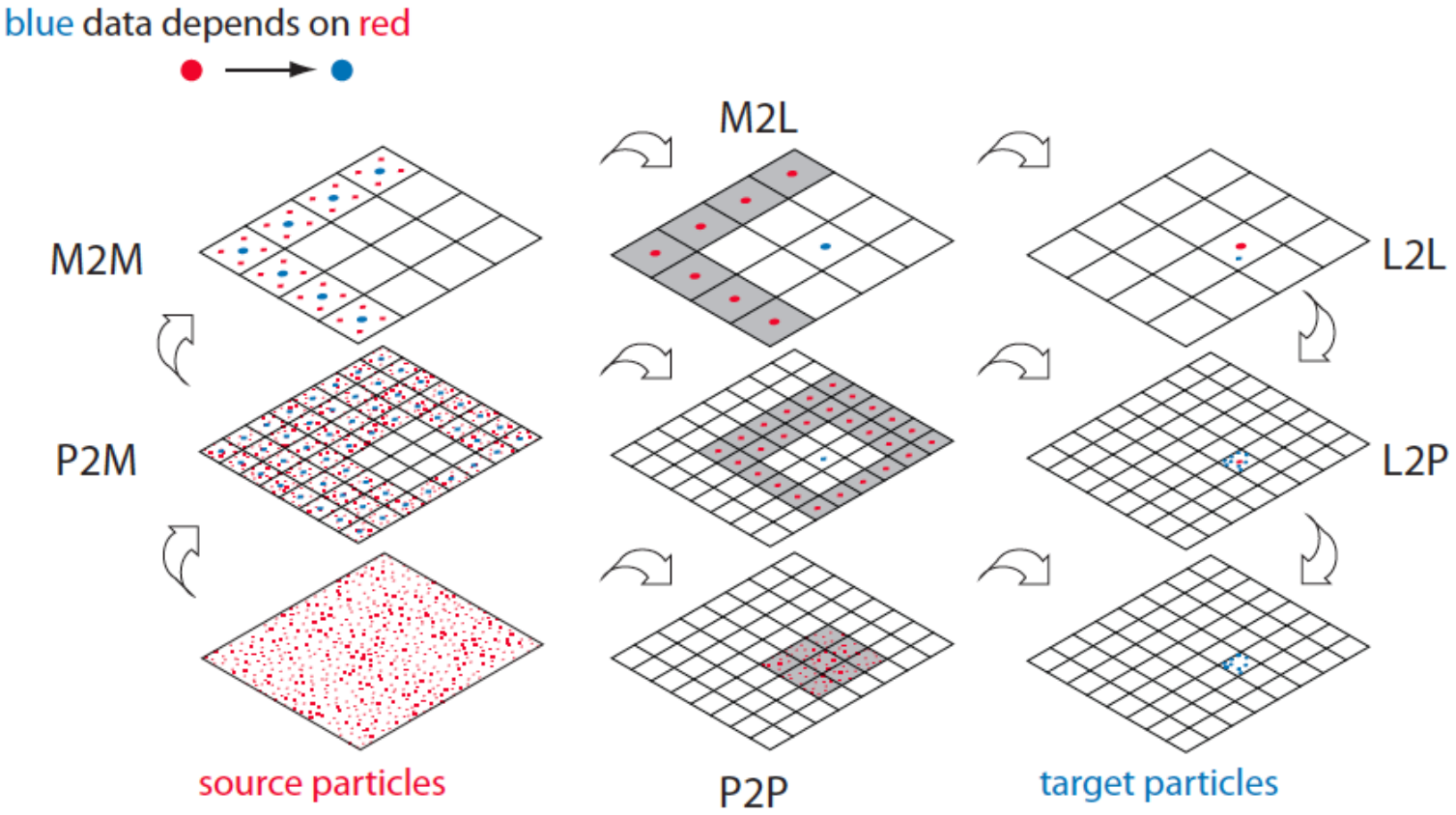
Re_λ: 500

N : 4096^3

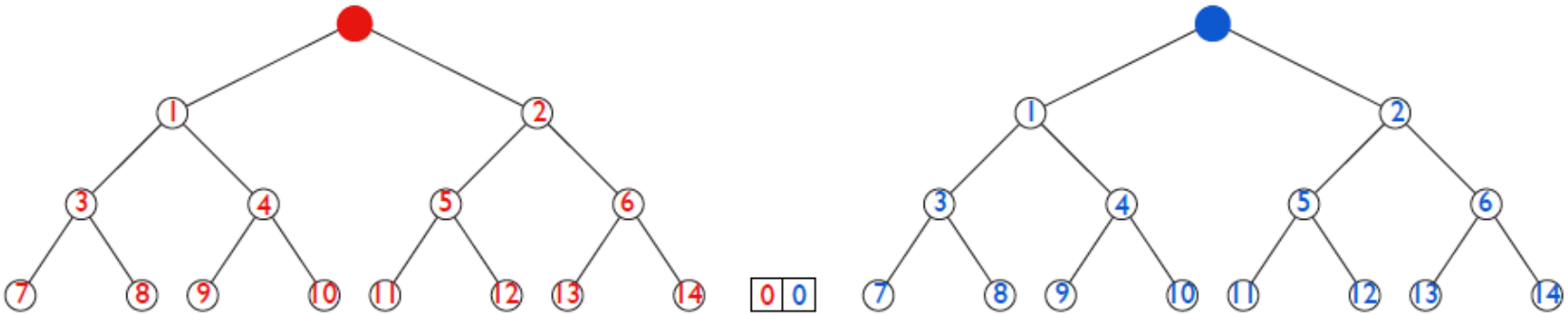


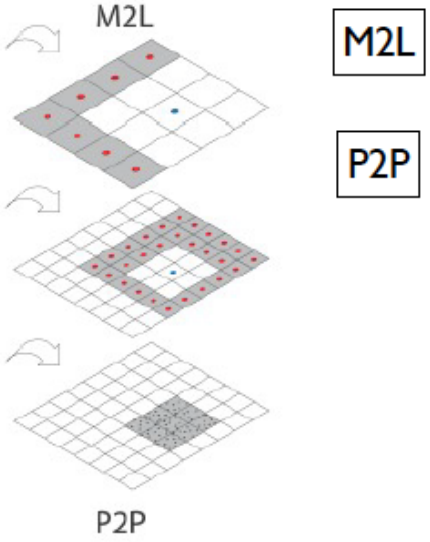
64 billion particles

Data Driven Fast Multipole Method



Dual Tree Traversal

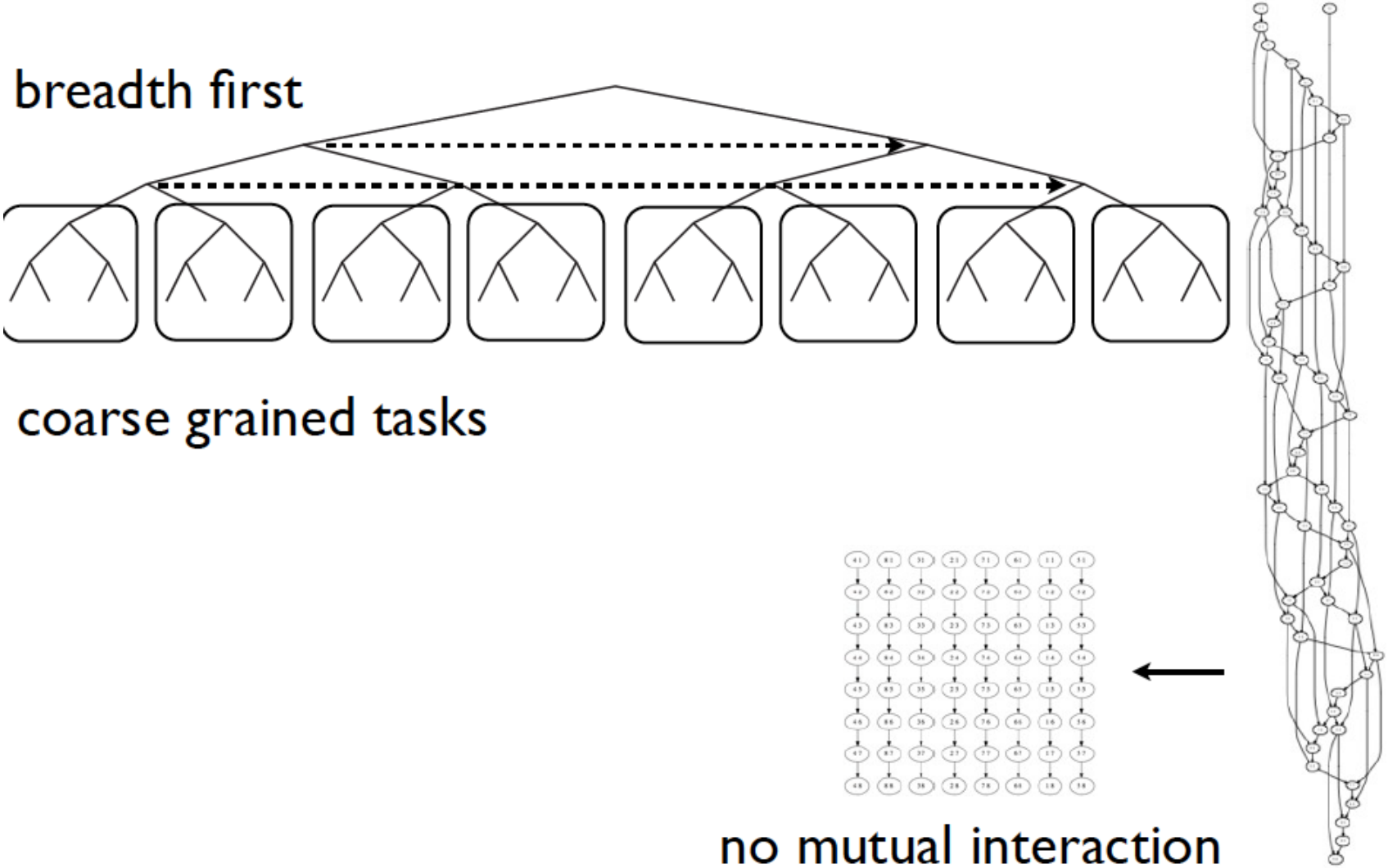




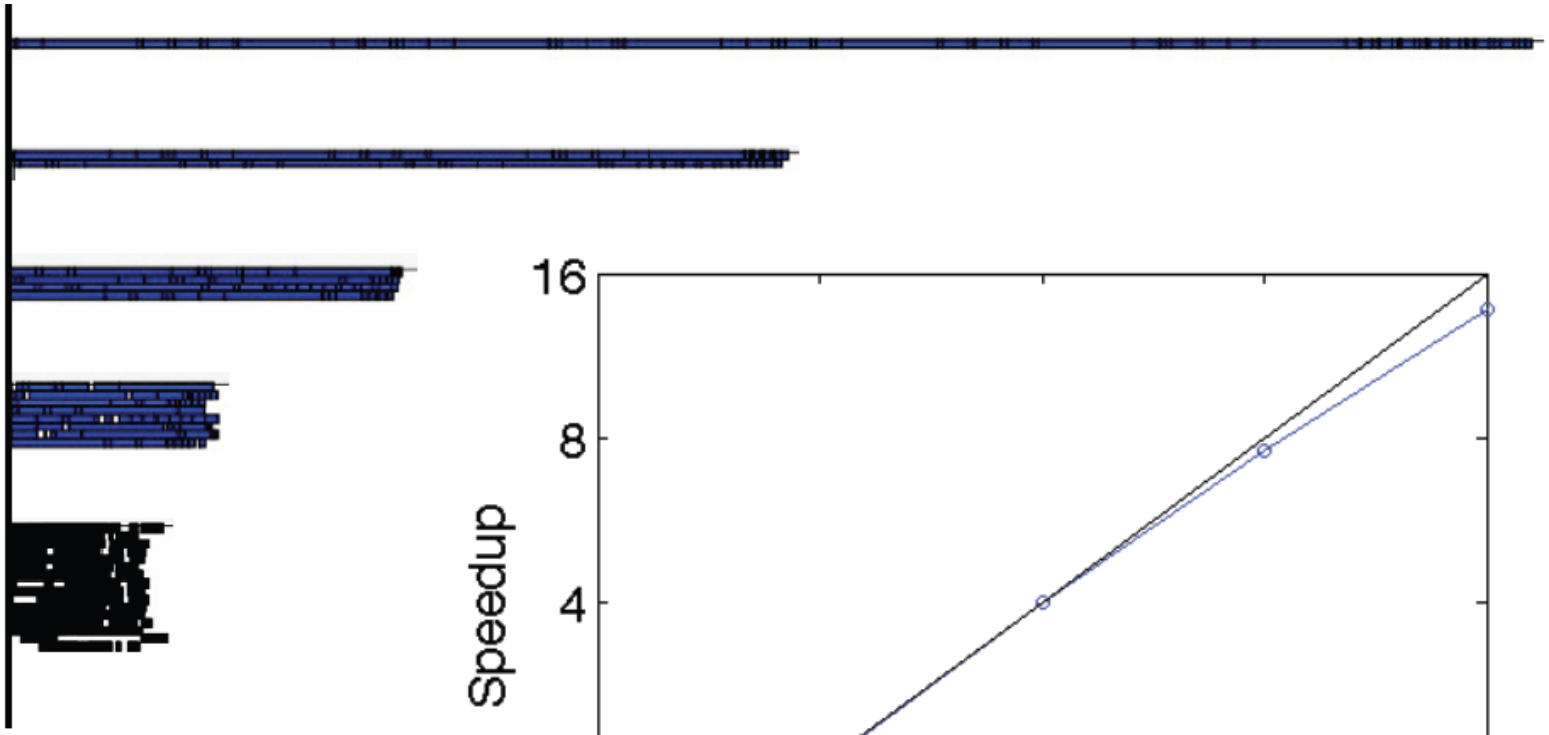
M2L

P2P

Adjustable Granularity



Strong Scaling w/ QUARK



H. Ltaief and R. Yokota, to be submitted at EuroPar'12

What is next?

- Heterogeneous architecture w/ hardware accelerators (StarPU)
- Distributed memory systems (DAGuE)
- Implementation of the reduction operation

Plan

Stencil kernels

- Explicit time integration scheme with domain decomposition
- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
- Two phases: calculate solutions inside a cone and then outside the cone
- The cone kernel becomes the fine-grain task to schedule
- No numerical instabilities added to the original scheme
- May have load imbalance due to PML high compute intensity
- Similar (to some extend) to Demmel's approach on the Matrix power kernel.
- Work in progress with A. Abdelfettah, PhD Student / Saudi Aramco / TOTAL

Plan

Conclusion

- Dataflow programming could play a major role for exascale challenges
- Need efficient runtime with high productivity *in mind*
- Need new flexible algorithms
- Work potentially for dense as well as sparse computations
- Need to determine an appropriate granularity though to hide scheduling overhead

Thank you!