

DAGuE: A Generic Distributed DAG Engine for HPC

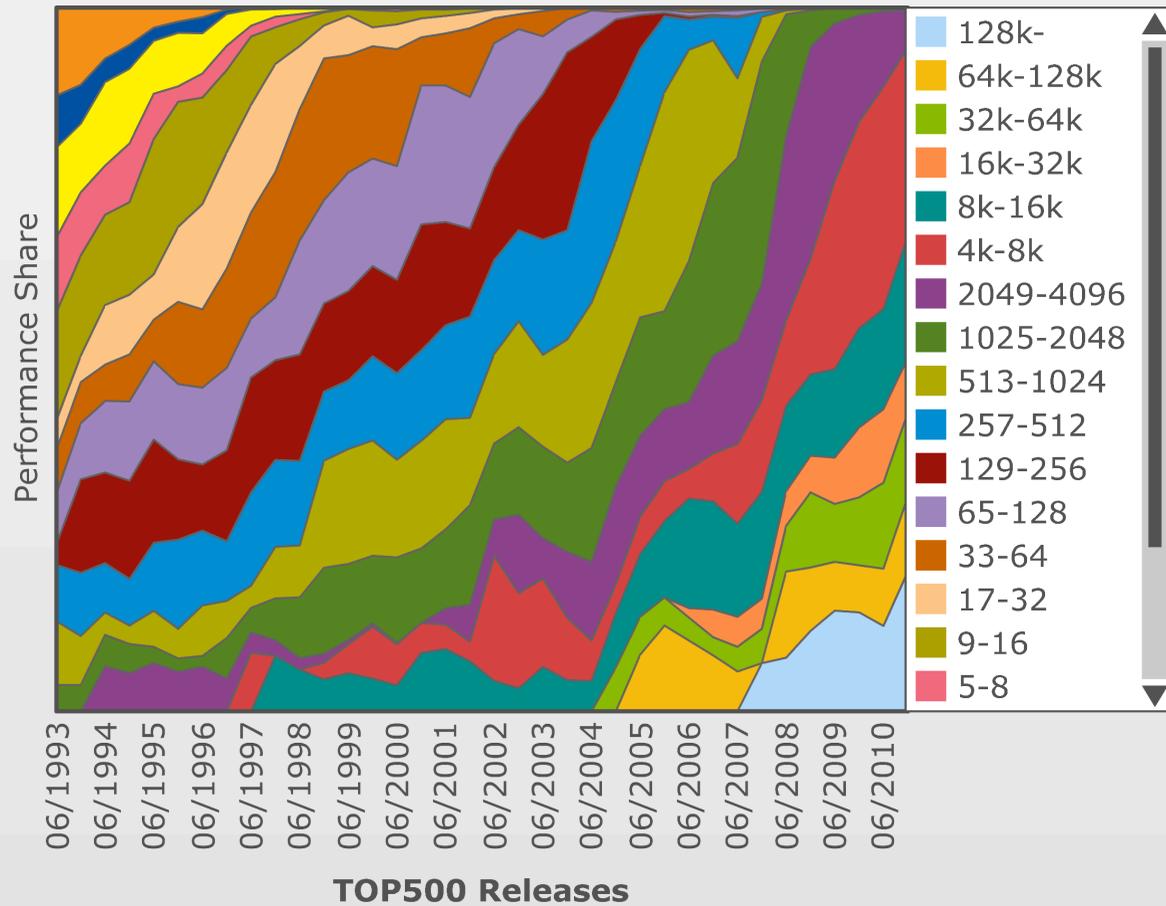
George Bosilca, Aurélien Bouteiller, Anthony Danalis,
Mathieu Faverge, Thomas Herault

Pierre Lemarinier, Emmanuel Agullo, Hatem Ltaeif

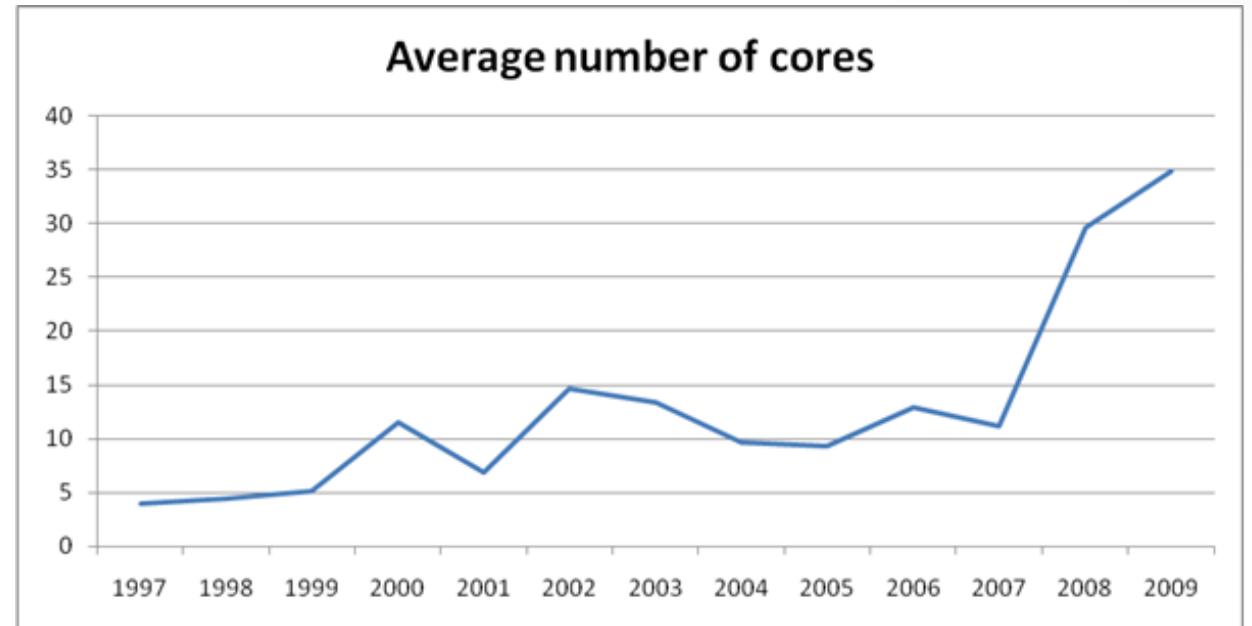
INRIA/UTK Associated Team MORSE

Top 500

Number of Processors Share Over Time
1993-2010



Average number of cores

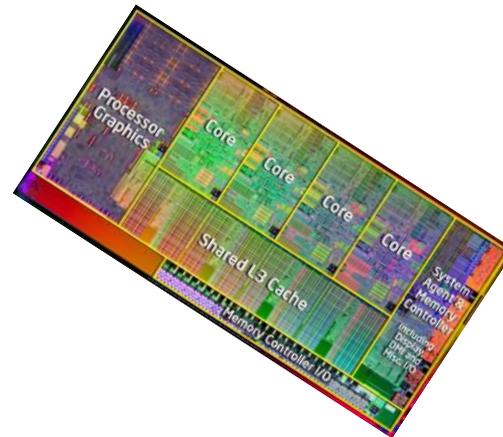
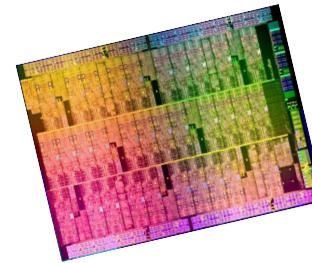
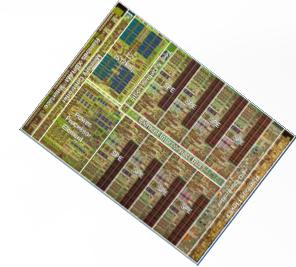


Hardware Complexity

- Hierarchies of Multi-Cores
- Non Uniform Memory Access
- Accelerators
- Networks with deep hierarchies

Portability

- Programming Portability
- Performance Portability

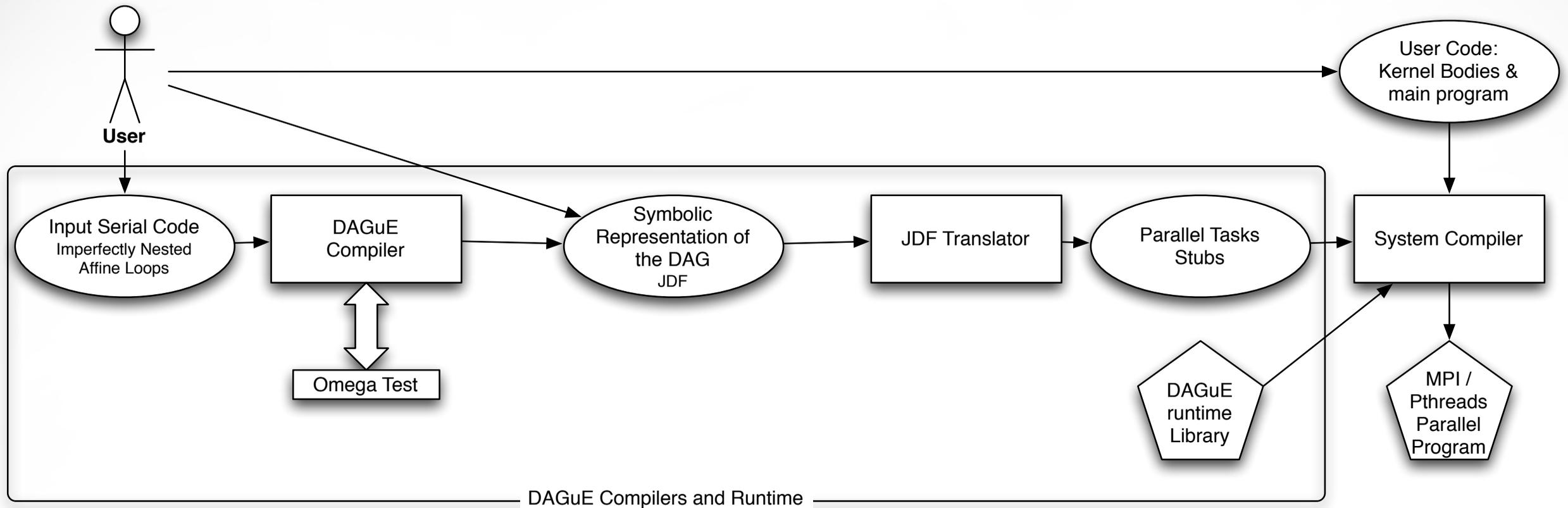


alls for Dynamic / Asynchronous Programming Model

DAGuE Goals

- Keep the algorithm as simple as possible
 - Depict only the flow of data between tasks
 - *Distributed Dataflow Environment based on Dynamic Scheduling of (Micro) Tasks*
- Programmability: layered approach
 - Algorithm / Data Distribution
- Portability / Efficiency
 - Use all available hardware; overlap comm / comp
- Decouple “System issues” from Algorithm

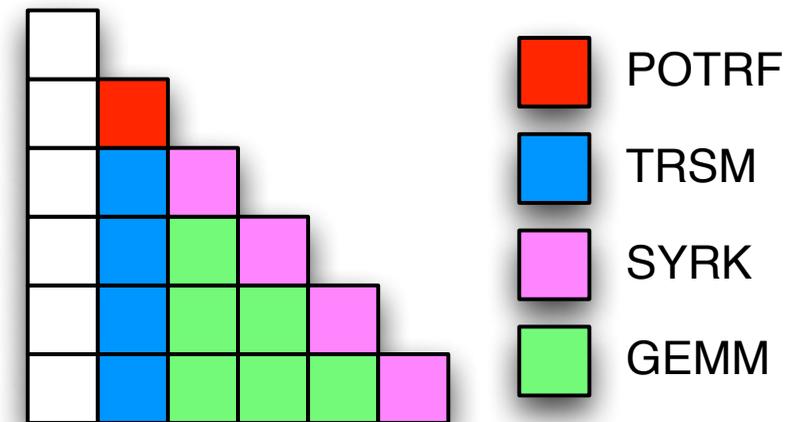
DAGuE toolchain



Example: Cholesky Factorization

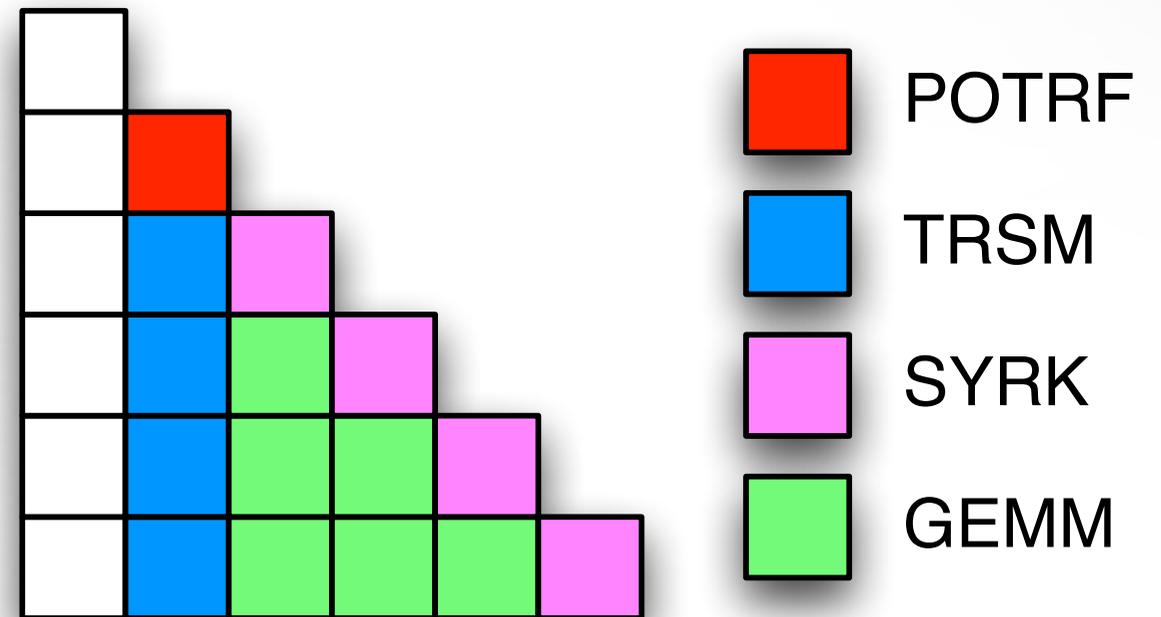
- Cholesky Decomposition
 - Let A be a real symmetric positive definite matrix
 - Find L such that $A = LL^T$

Tiled Algorithm in A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, A class of parallel tiled linear algebra algorithms for multicore architectures, *Parallel Computing*, 2008



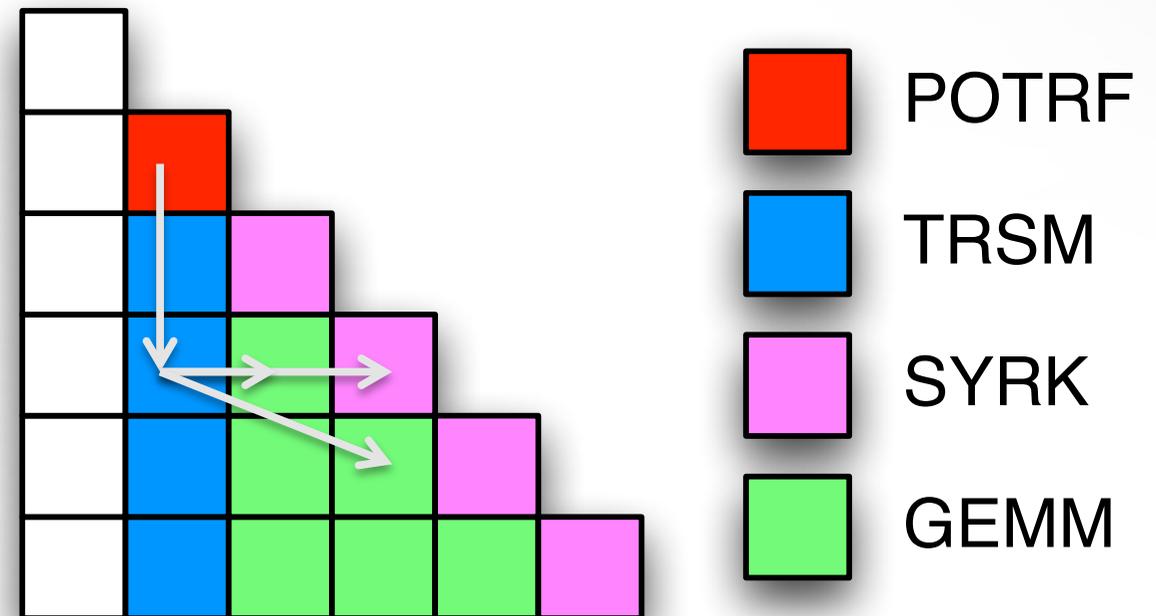
Input Format: SMPSS-Like

```
FOR k = 0..TILES-1
  A[k][k] ← DPOTRF(A[k][k])
  FOR m = k+1..TILES-1
    A[m][k] ← DTRSM(A[k][k], A[m][k])
  FOR n = k+1..TILES-1
    A[n][n] ← DSYRK(A[n][k], A[n][n])
  FOR m = n+1..TILES-1
    A[m][n] ← DGEMM(A[m][k], A[n][k], A[m][n])
```



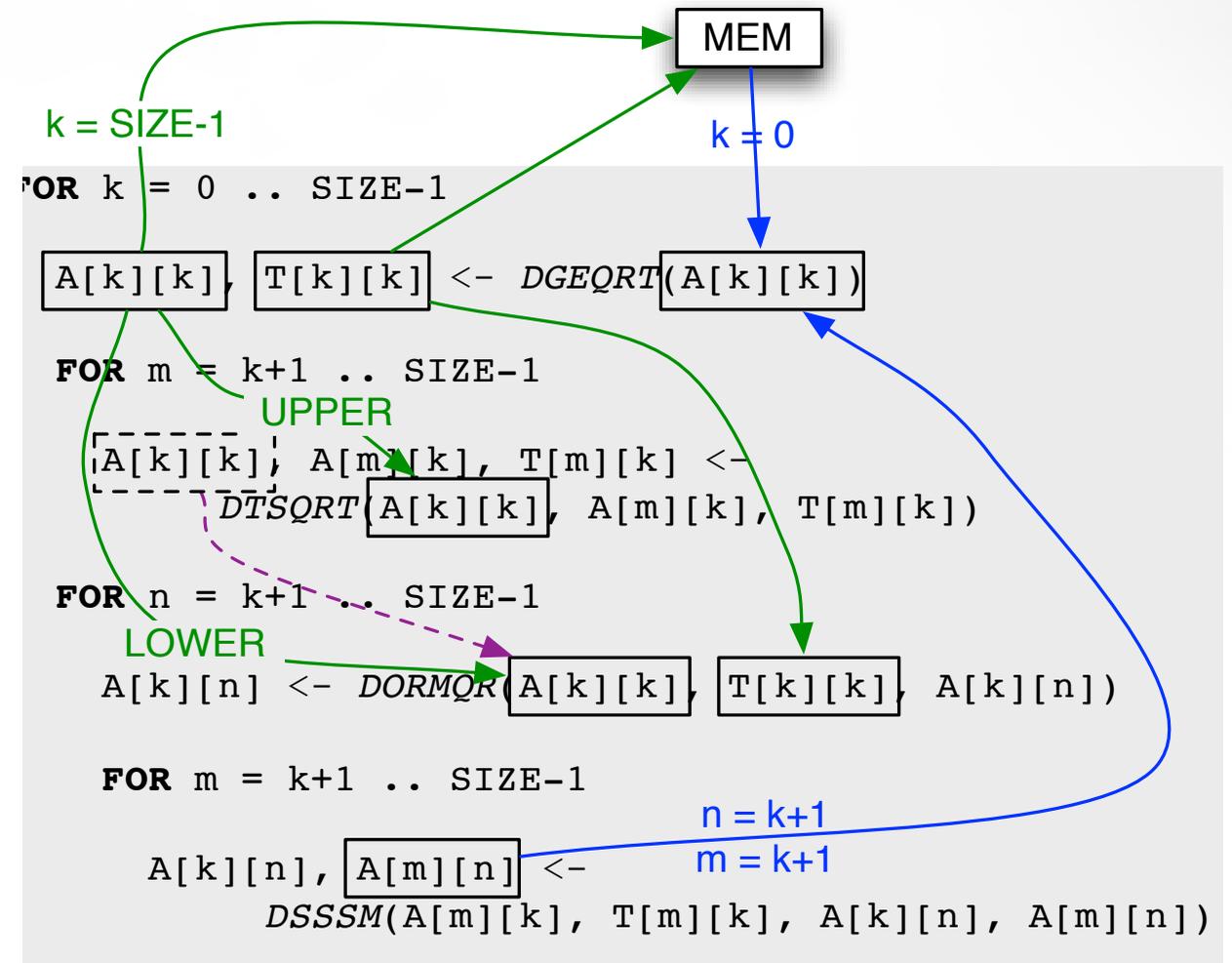
Input Format: Job Data Flow

```
TRSM(k, n)
// Execution space
k = 0..SIZE-1
n = k+1..SIZE-1
: A(n, k) // Parallel Partitionning
READ  T <- T POTRF(k)
RW    C <- (k == 0) ? A(n, k)
      : C GEMM(k-1, n, k)
      -> A SYRK(k, n)
      -> A GEMM(k, n+1..SIZE-1, n)
      -> B GEMM(k, n, k+1..n-1)
      -> A(n, k)
```

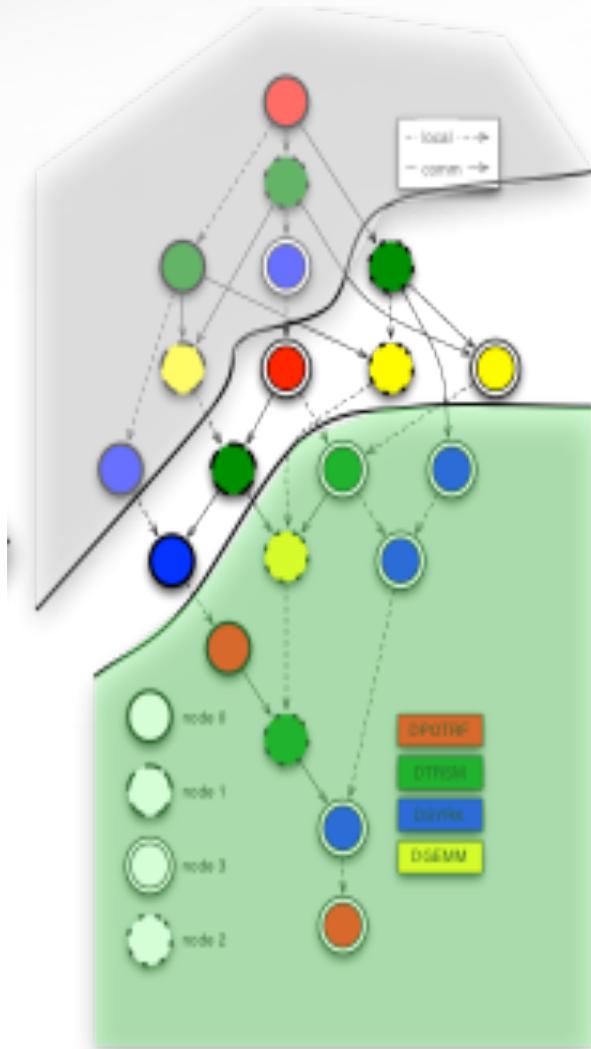


From Seq. to JDF

- DAGuE Compiler
 - Analysis the data flow using algebraic expressions
 - Omega Test used to compute algebraic relations between edges
 - Imperfectly nested affine loop tests
 - Anti-Dependencies may introduce additional control edges

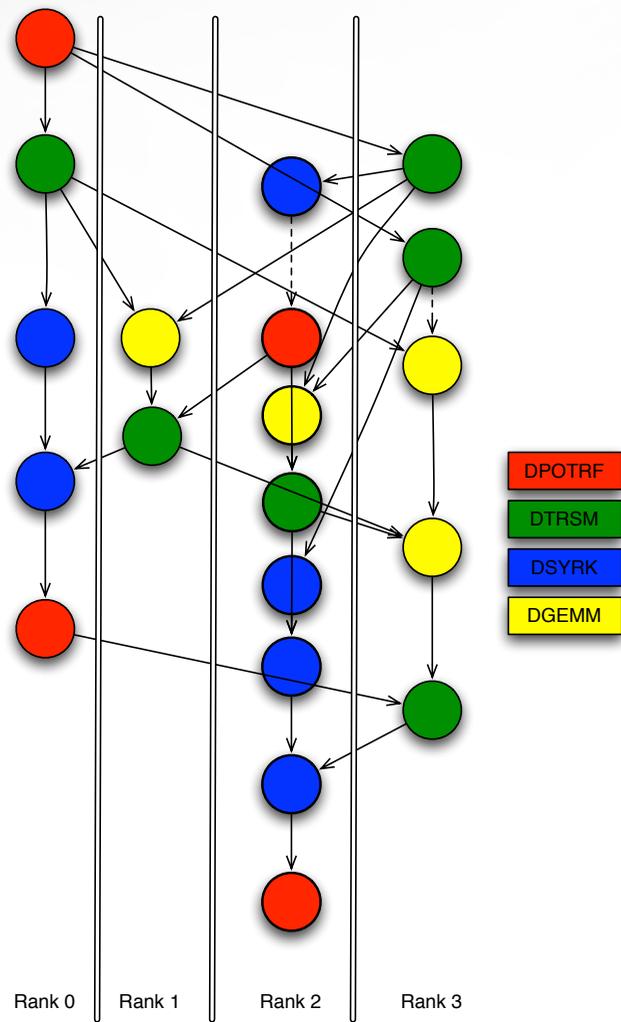


Runtime DAG Representation



- Every process has the algebraic DAG rep.
- Dist. Scheduling based on remote completion notifications
- NUMA / Cache aware Scheduling
- Work Stealing and sharing based on memory hierarchies

Runtime DAGuE Engine



- Data Distribution (and data/task affinity) imposes a task location
- On each node, the full DAG algebraic representation is available
- Each computing unit (core, GPU, etc.) runs its own instance of the DAGuE scheduler
- An additional communication thread sends completion notifications and data when necessary

Scheduling in DAGuE

- Based on Work Stealing
 - Shared data structures with atomic access operations
 - Uniform scheduler: all scheduler run with the global view of the DAG and the local view of progress (plus remote notifications)
 - Fully Distributed scheduler: all threads alternate between scheduling and work
- Main heuristic: data locality
 - DAGuE engine tracks data usage, and targets to improve data reuse
 - NUMA aware hierarchical bounded buffers to implement work stealing
- Users hints: tasks with “high priority”; Algebraic expressions for priorities
 - Insertion in waiting queue abides to priority, but work stealing can alter this ordering
- Communications heuristics
 - Communications inherits priority of destination task

Example: RTT

Sequential code:

```
B = ping(A[0]);  
for(k = 1; k < NT-1; k++) {  
    C = pong(B);  
    B = ping(C);  
}  
A[0] = B;
```

- Read data into B using ping
- Call pong on B to make C
- Call ping on C to make B
- Iterate NT-1 times
- Save data

Example RTT: JDF

```
1  PING(k)
2  k = 0 .. NT // Execution space
3  : A[0] // Parallel partitioning
4  T ← (k == 0) ? A(0) : I PONG(k-1) [ATYPE]
5  → (k == NT) ? A(0) : I PONG(k) [ATYPE]
6
7  PONG(k)
8  k = 0 .. NT-1 // Execution space
9  : A[1] // Parallel partitioning
10 I ← T PING(k) [ATYPE]
11 → T PING(k+1) [ATYPE]
12
```

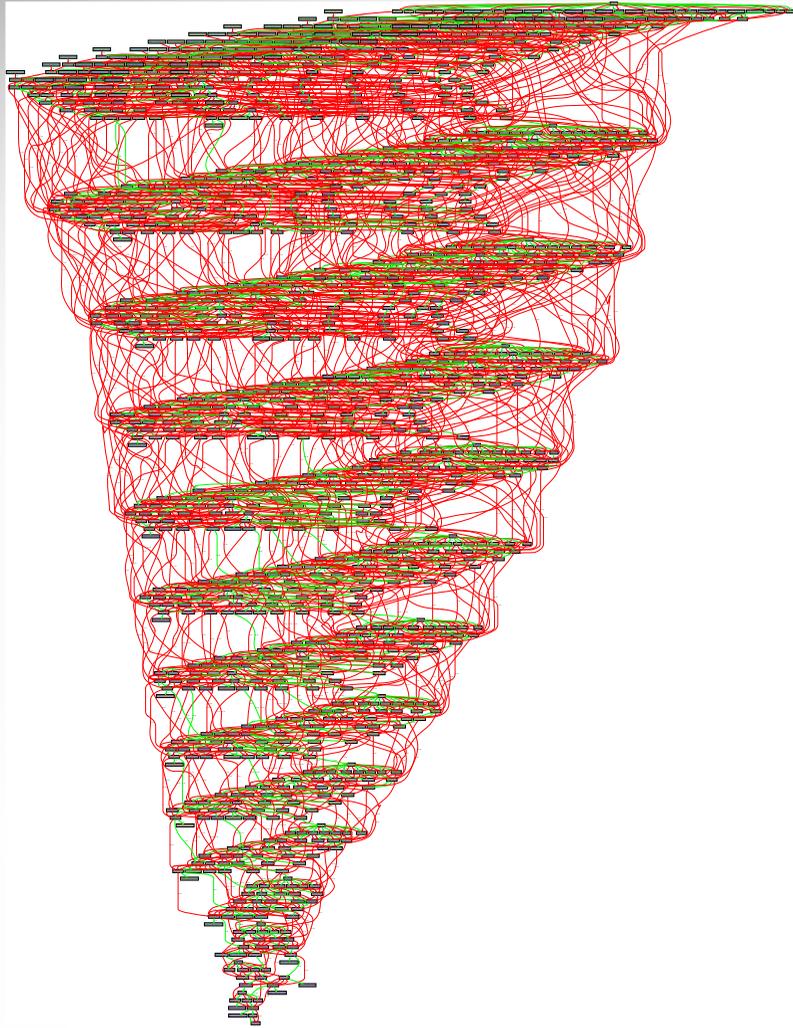
- The JDF Translator will create a stub (rtt.c / rtt.h) which includes two visible functions:
 - rtt_new(A, NT) creates a DAG generator for a specific data A
 - Data format includes accessors to discover the data distribution
 - rtt_destroy() frees the resources allocated by the new

Example RTT: main program

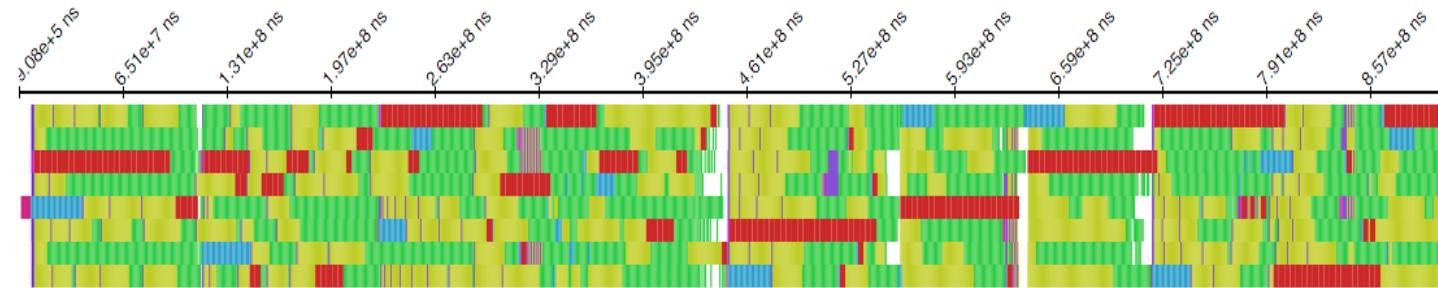
```
#include "ping_pong.h"
int main( int argc, char** argv) {
    int size, rank; /* MPI_COMM_WORLD size and rank */
    dague_object_t* rtt;
    dague_matrix_t* A = dague_matrix( 1000, [ATYPE],
                                      "cyclic", 1, size );

    dague_t* mydag = dague_init("-c 1");
    rtt = rtt_new( A, rank, 1000);
    dague_enqueue( my_dag, rtt );
    dague_progress( my_dag );
    dague_fini( &my_dag );
}
```

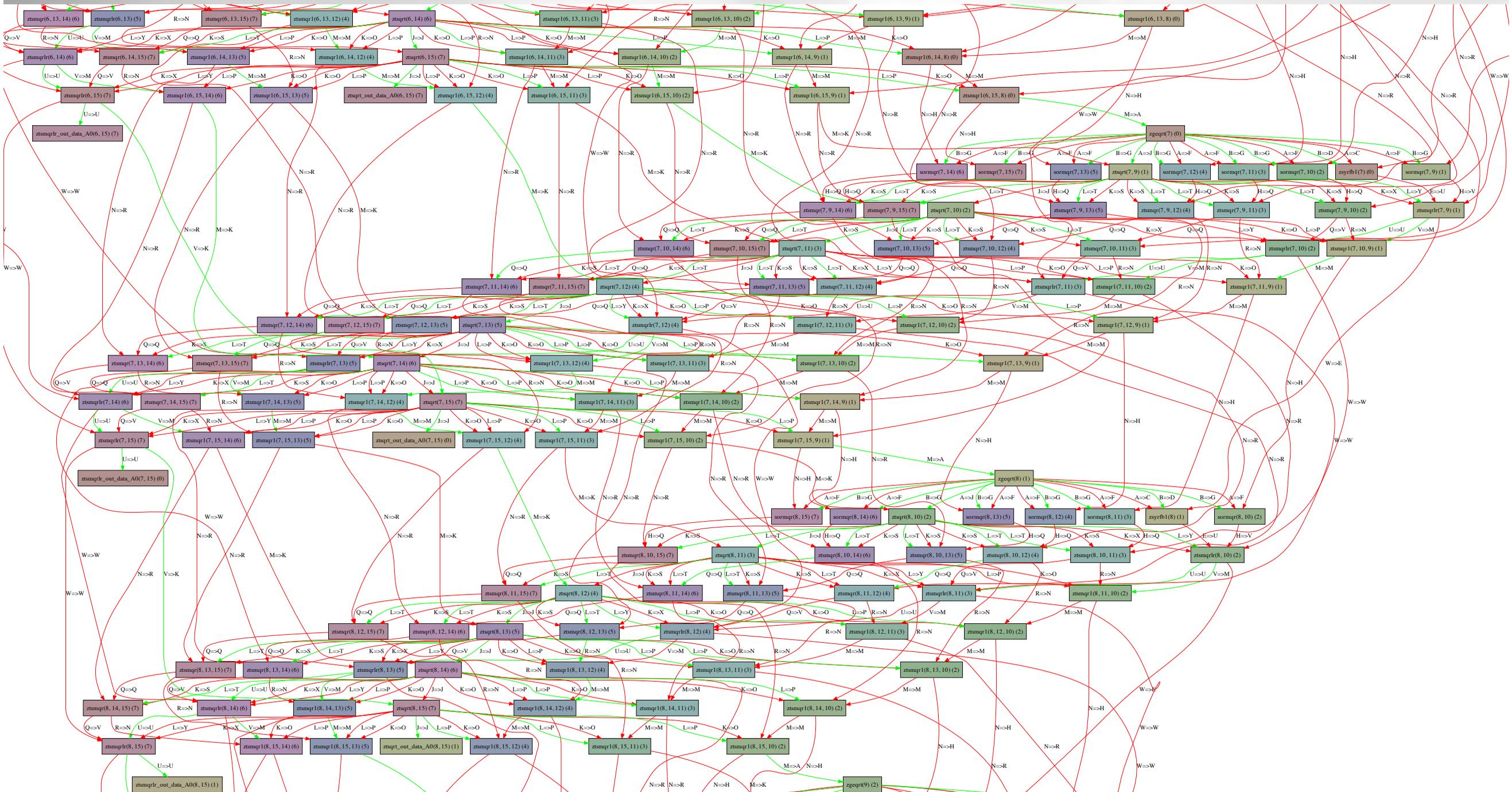
DAGuE: Analysis Tools



Proc 0: DAGuE Thread 0
Proc 0: DAGuE Thread 7
Proc 0: DAGuE Thread 6
Proc 0: DAGuE Thread 4
Proc 0: DAGuE Thread 1
Proc 0: DAGuE Thread 3
Proc 0: DAGuE Thread 5
Proc 0: DAGuE Thread 2

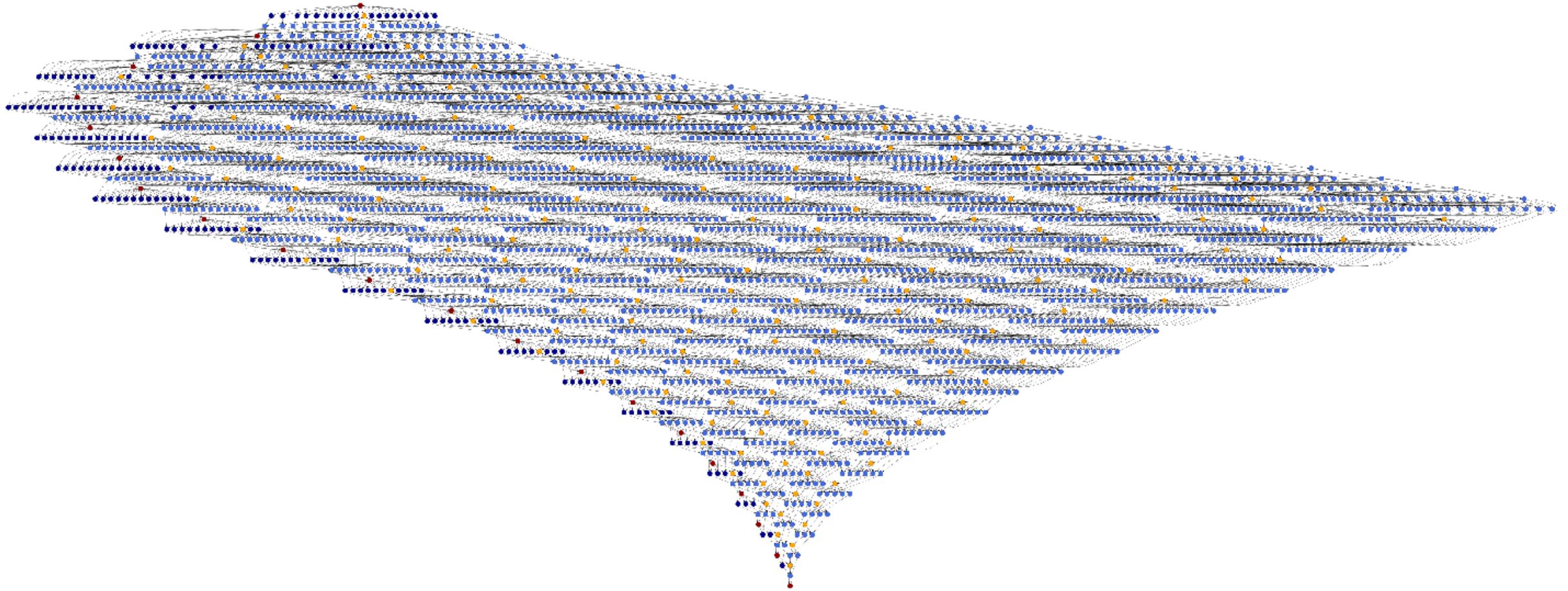


Hermitian Band Diagonal; 16x16 tiles



Performance Evaluation

Cholesky Factorization, QR, LU



Experimental Platform

Dancer @ UTK

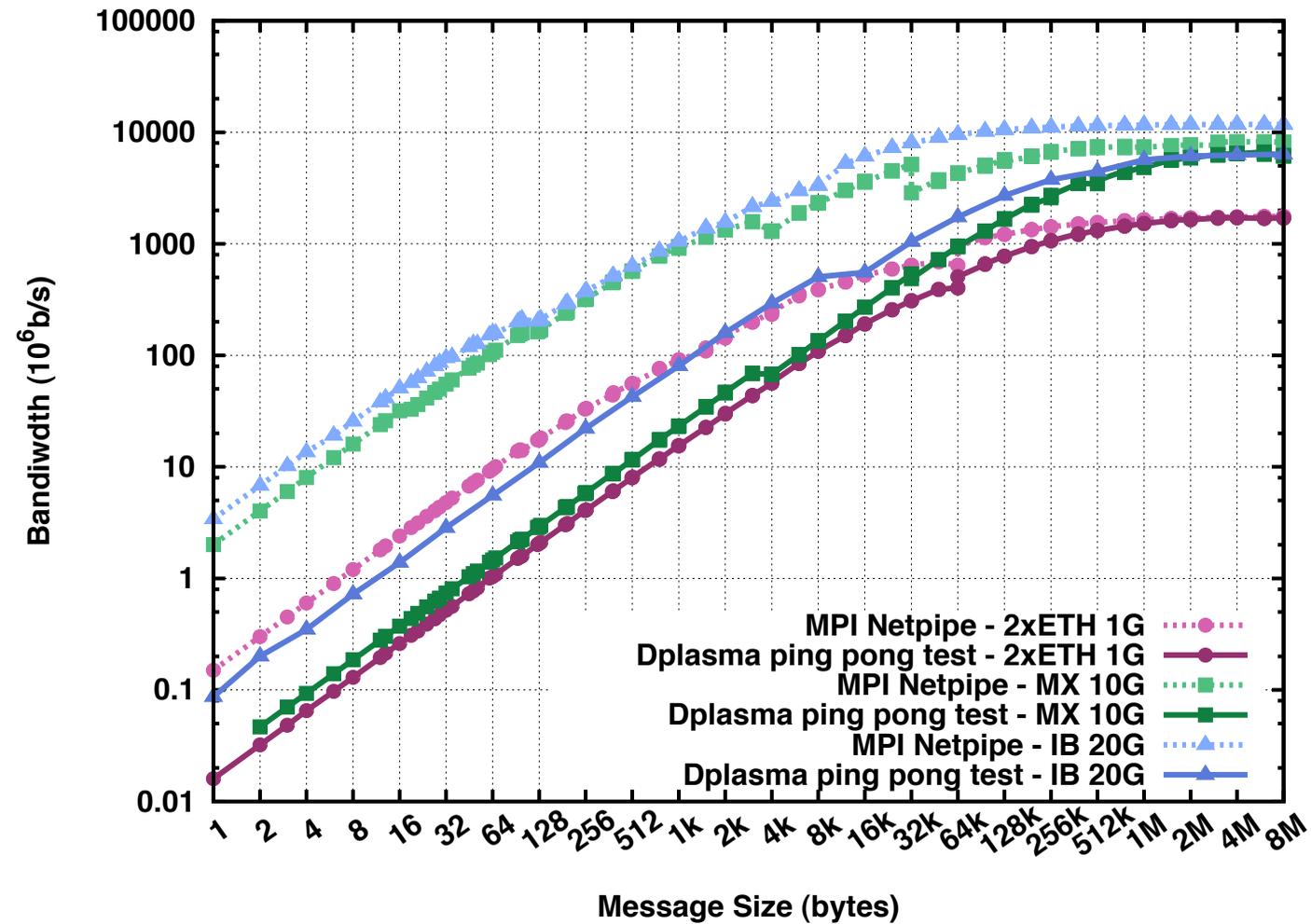
- 32 Cores (8 sockets)
- Intel Q9400 quad cores @ 2.5GHz
- 4GB RAM
- 2x 1GB/s ethernet
- 4 nodes with Fermi GPU
- 4 nodes with Tesla GPU

Griffon @ Grid'5000

- 648 Cores (81 dual sockets)
- Intel Xeon L5420 quad cores @ 2.5GHz
- 16GB RAM
- 20 GB/s Infiniband

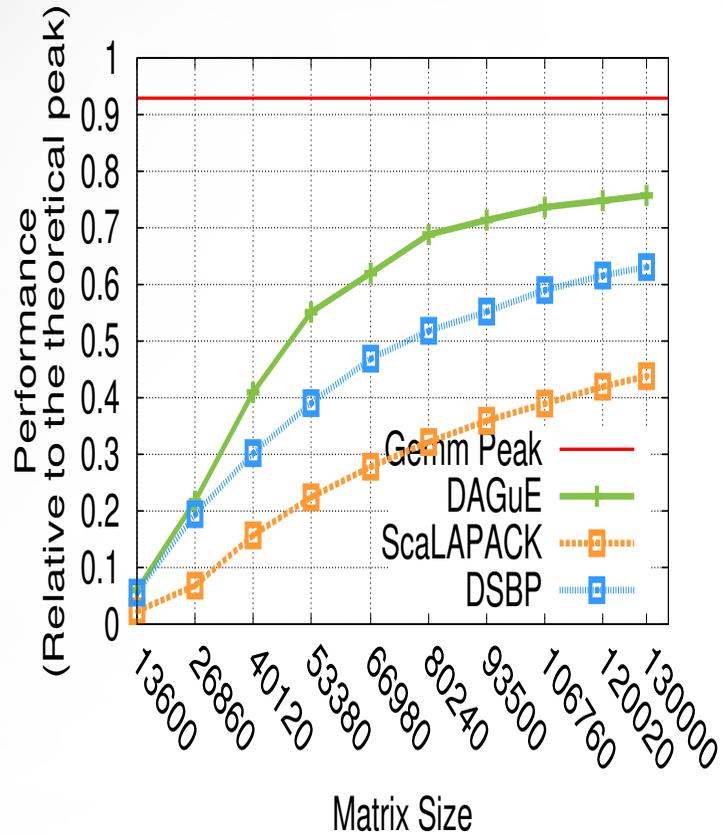
MKL-10.1.0.015 / gcc 4.4 / gfortran 4.4

Communications

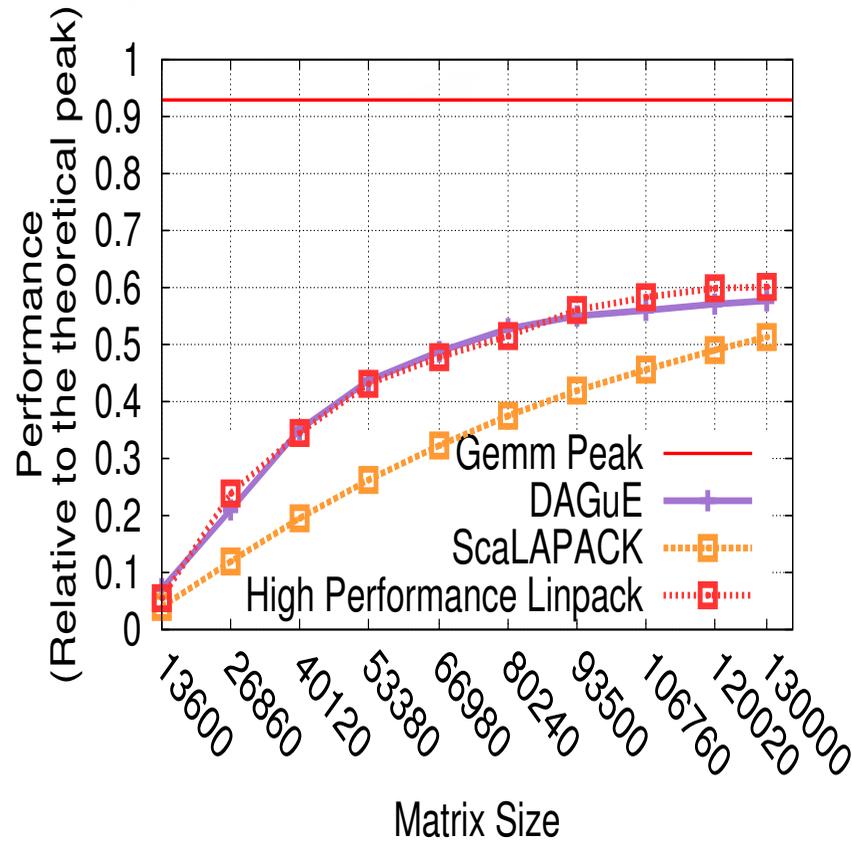


Problem Scaling

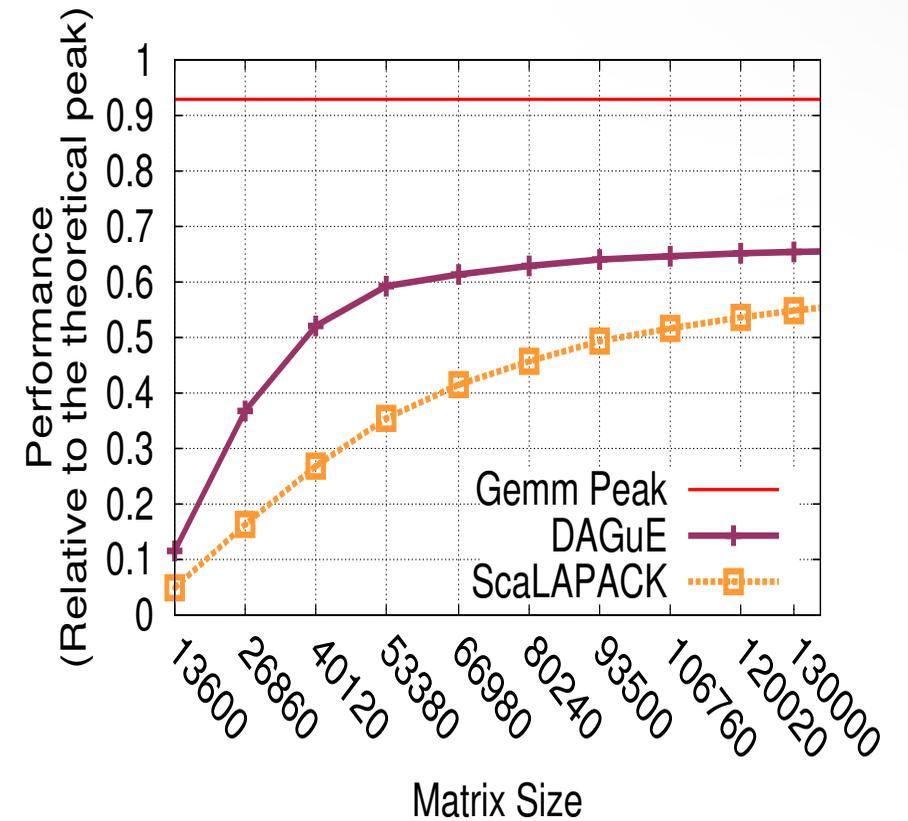
Cholesky



LU

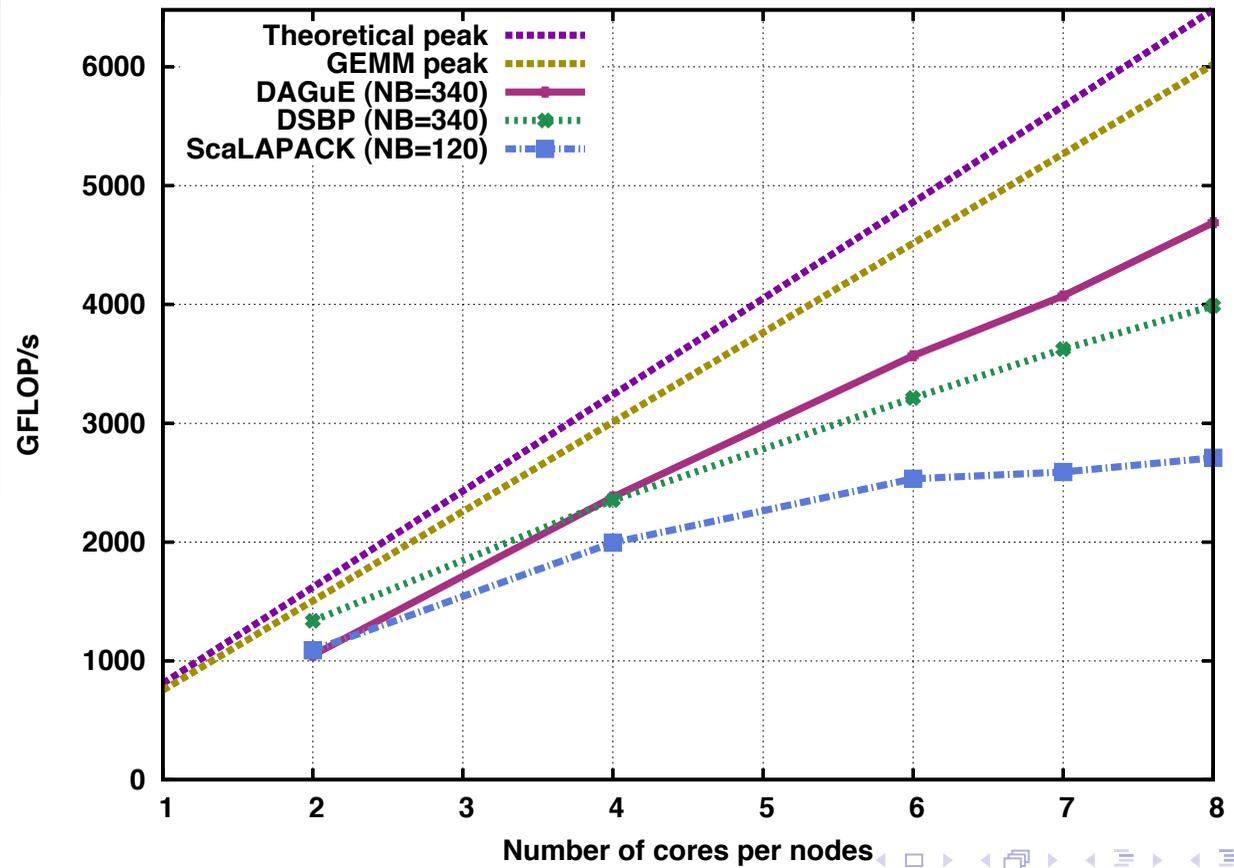


QR

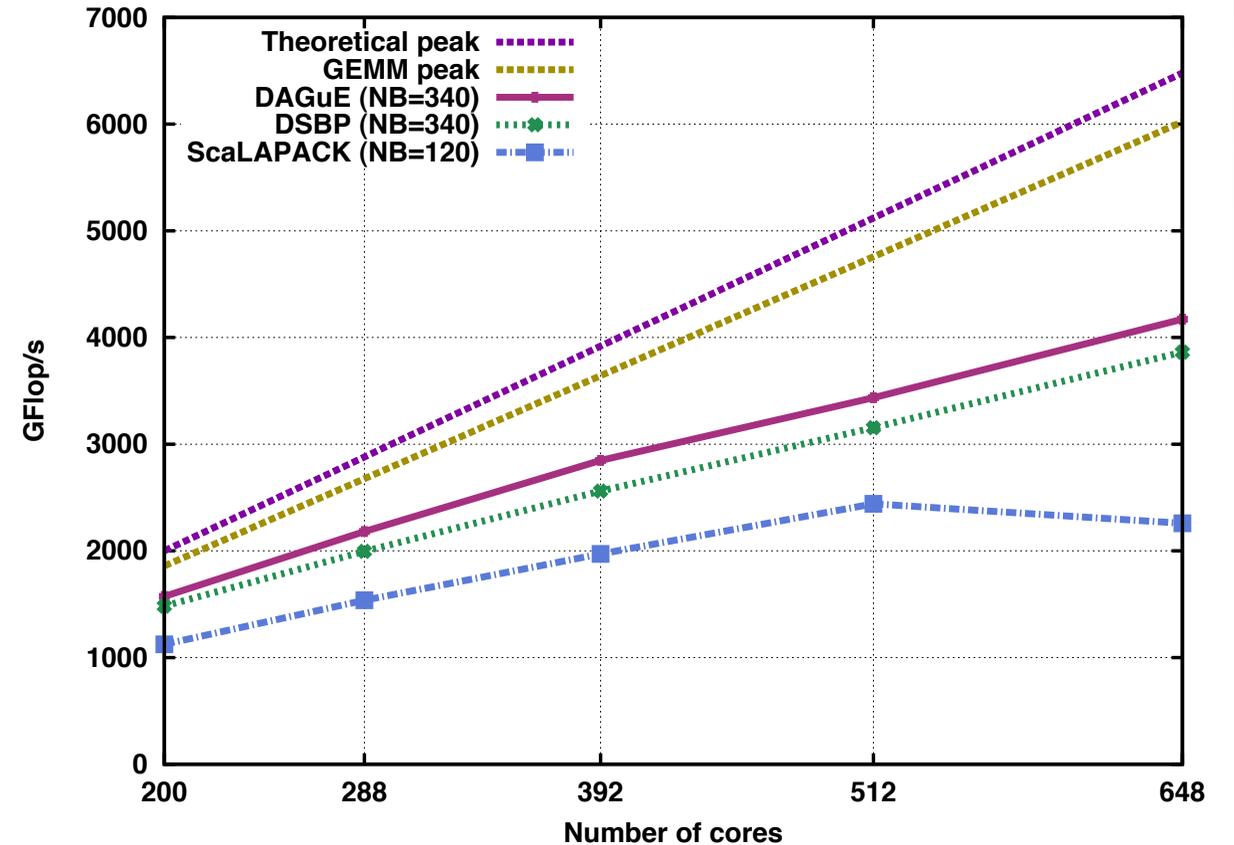


Cholesky Strong Scalability

Increase the number of cores

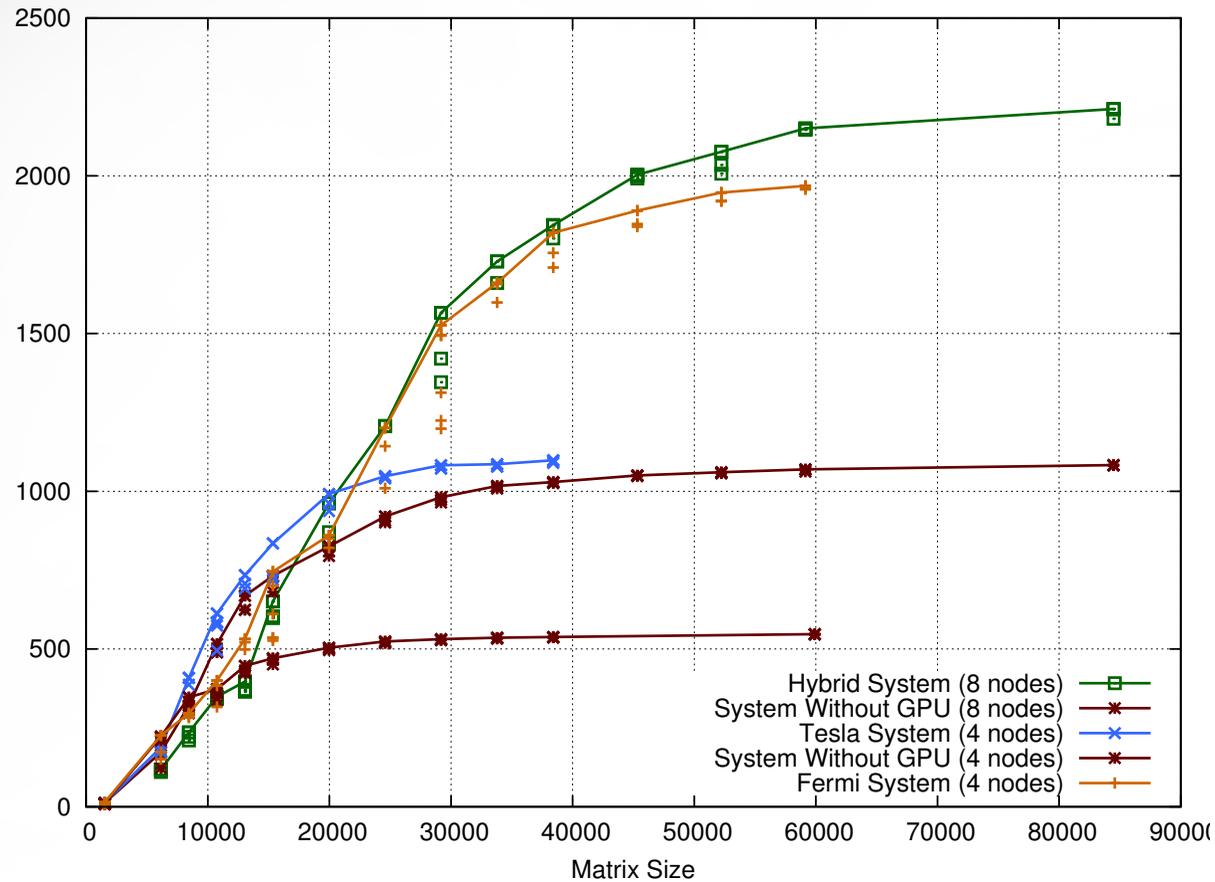


Increase the number of nodes

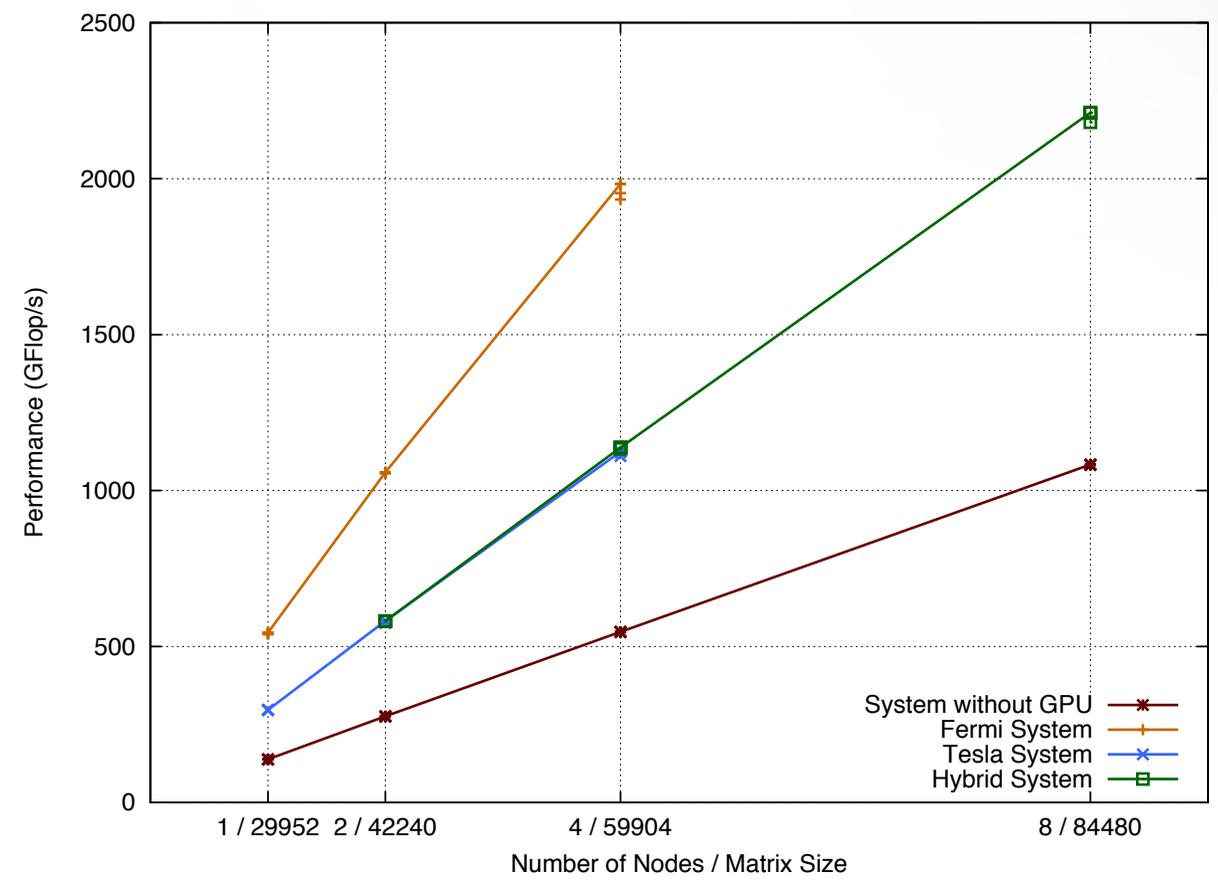


DAGuE: distributed multi-GPU

Problem Scaling



Weak Scaling



Conclusion

- Hybrid programming (of dense LA) made easy(ier)
 - Portability: inherently take advantage of all hardware capabilities
 - Efficiency: deliver the best performance on tested algorithms
- Works well with Dense Linear Algebra with Direct Method
 - Sparse?
 - Branch and Bound?
 - Iterative Method?
- Let different people focus on different problems
 - Application developers on their algorithms
 - System developers on system issues

Related Works

- YML Like: read the DAG representation; unroll it (completely); schedule it (centrally)
- PLASMA / MAGMA / T-BLAS / STARPU like: seq. code -> new tasks + dep.; tasks window; sched. is dyn/stat
- PTG like: use a concise DAG representation to discover new tasks. Have a bounded ready list.

DAGuE uses a concise representation of the DAG, instantiates dynamically the tasks, scheduling is fully distributed and dynamic