2012 Scheduling Workshop, Pittsburgh, PA

DAGuE

http://icl.utk.edu/dague

George Bosilca, Aurelien Bouteiller, Anthony Danalis,
Mathieu Faverge, Thomas Herault, Jack Dongarra

INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

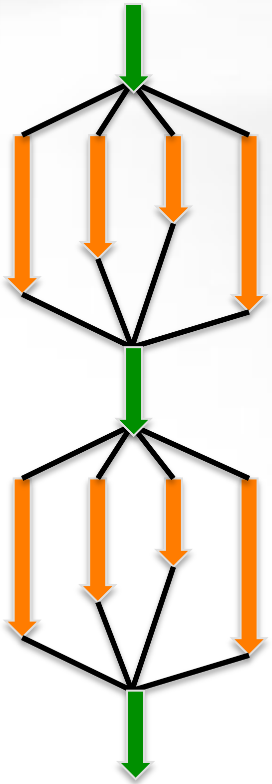
DAGuE

(the Prague Astronomical Clock was first installed in 1410, making it the third-oldest astronomical clock in the world and the oldest one still working. – Wikipedia Notice)

- DAGuE [dag] (like in Prague [prag])
  - Not DAGuE like ragout [rágoo̅]
  - Not DAGuE like vague [väg]
- Innovative Computing Laboratory, University of Tennessee, Knoxville
- Task / Data Flow Computation Framework
  - Dynamic Scheduling
  - Symbolic DAG representation
  - Distributed Memory
  - Many-core / Accelerators

Motivation

- Today software developers face systems with
  - ~1 TFLOP of compute power per node
  - 32+ of cores, 100+ hardware threads
  - Highly heterogeneous architectures (cores + specialized cores + accelerators/coprocessors)
  - Deep memory hierarchies
  - Today, we deal with thousands of them (plan to deal with millions)
  - ➔ systemic load imbalance / decreasing use of the resources
- How to harness these devices productively?
  - SPMD produces choke points, wasted wait times
  - We need to improve efficiency, power and reliability

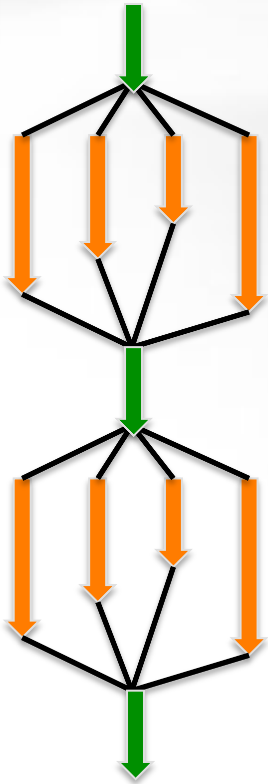


How to Program

- Threads & synchronization | Processes & Messages
  - Hand written Pthreads, compiler-based OpenMP, Chapel, UPC, MPI, hybrid
- Very challenging to find parallelism, to debug, to maintain and to get good performance
  - *Portably*
  - *With reasonable development efforts*

When is it time to redesign a software?

- Increasing gaps between the capabilities of today's programming environments, the requirements of emerging applications, and the challenges of future parallel architectures



Goals

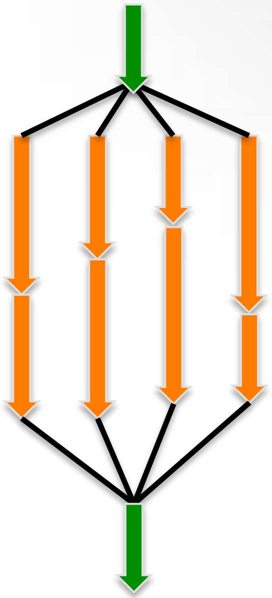
Language

Decouple "System issues" from Algorithm

- 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
  - Parallel applications without parallel programming

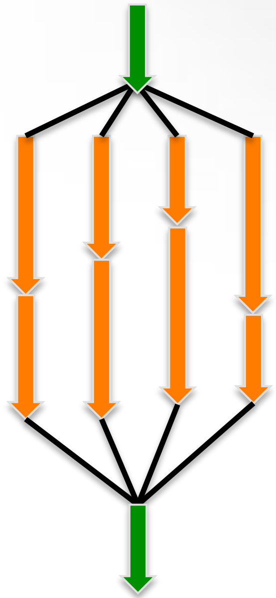
System

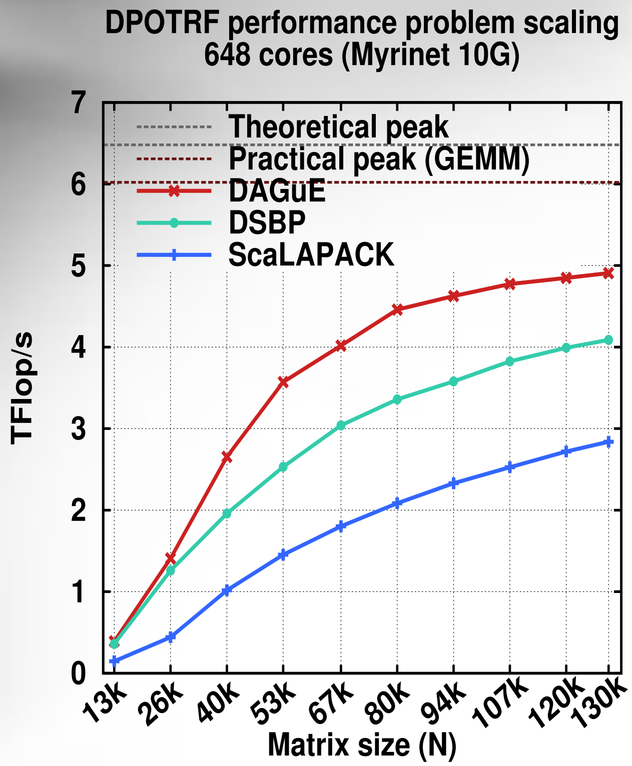
- Portability / Efficiency
  - Use all available hardware; overlap data movements / computation
  - Find something to do when imbalance arise

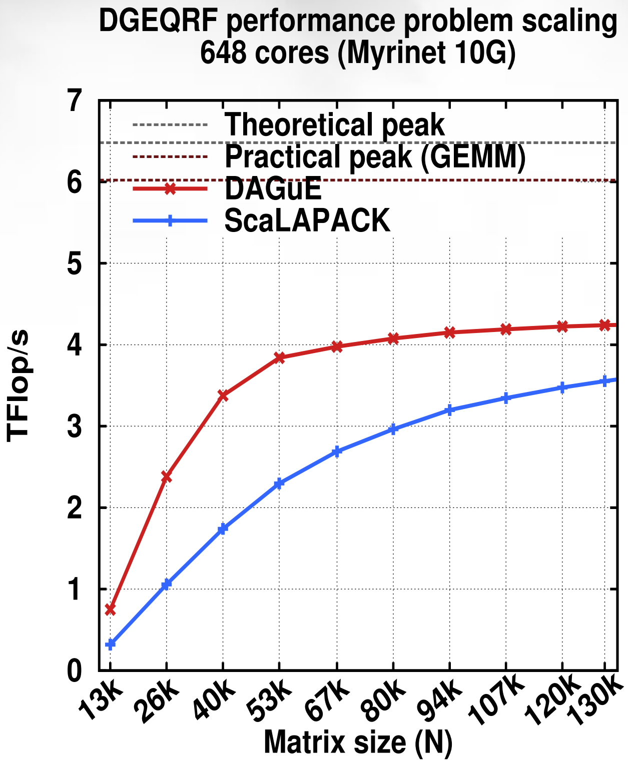


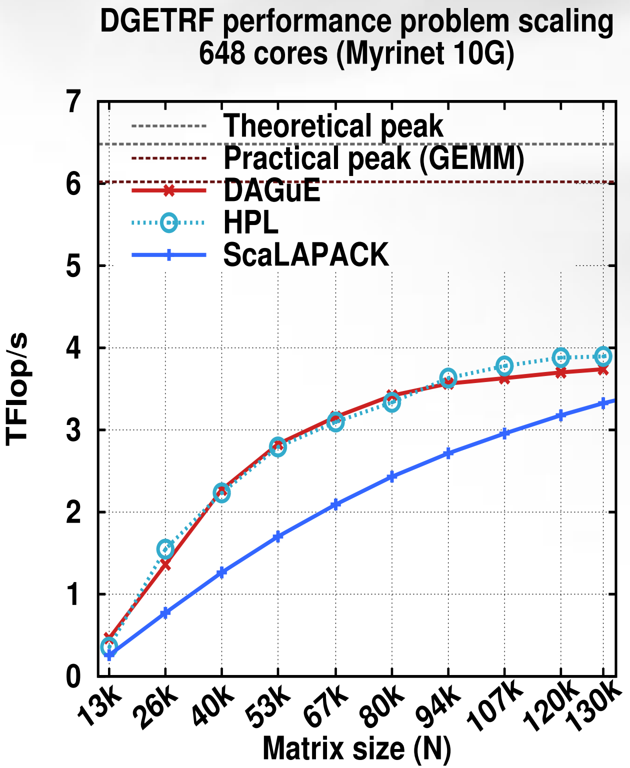
Dataflow with Runtime scheduling

- Algorithms expect help to abstract
  - *Hardware specificities*: a runtime can provide portability, performance, scheduling heuristics, heterogeneity management, data movement, …
  - *Scalability*: maximize parallelism extraction, but avoid centralized scheduling or entire DAG representation: dynamic and independent discovery of the relevant portions during the execution
  - *Jitter resilience*: Do not support explicit communications, instead make them implicit and schedule to maximize overlap and load balance
- ➔express the algorithms differently





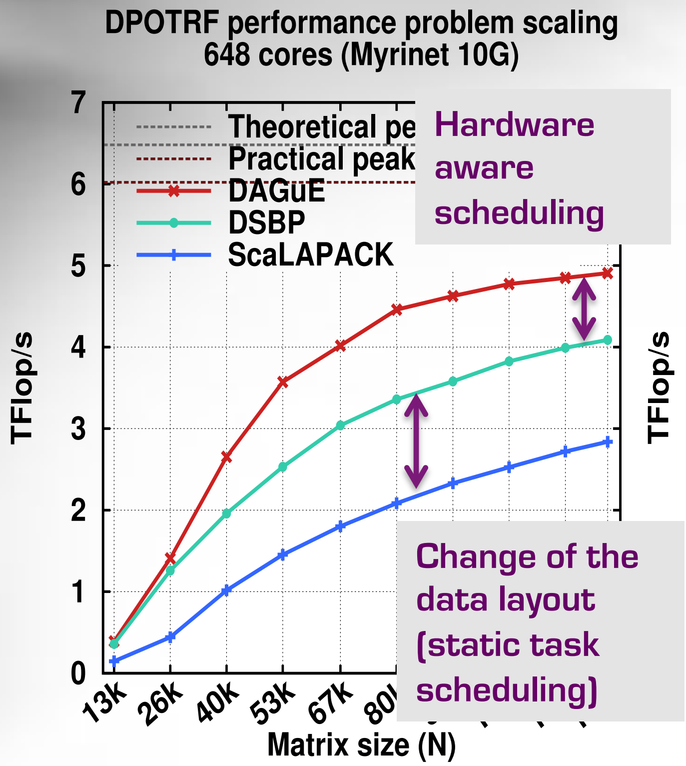


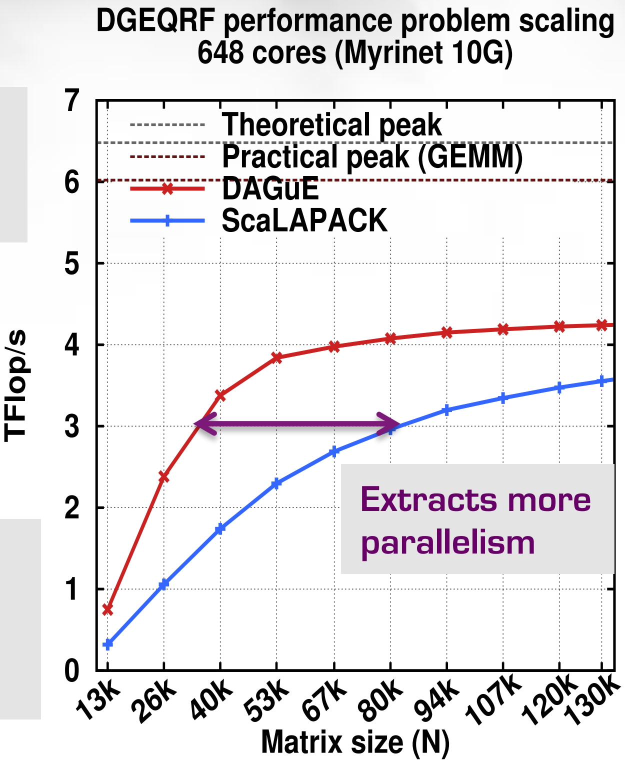


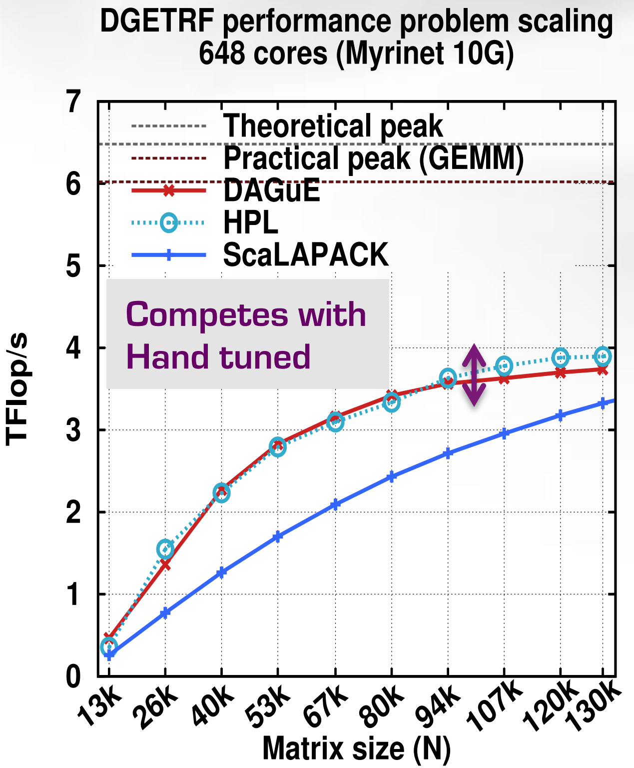
DSBP

[22] F. G. Gustavson, L. Karlsson, and B. Kågström. Distributed SBP cholesky factorization algorithms with near-optimal scheduling. *ACM Trans. Math. Softw.*, 36(2):1–25, 2009. ISSN 0098-3500. DOI: 10.1145/1499096.1499100.

81 dual Intel Xeon L5420@2.5GHz
(2x4 cores/node) ➔ 648 cores
MX 10Gbs, Intel MKL, Scalapack





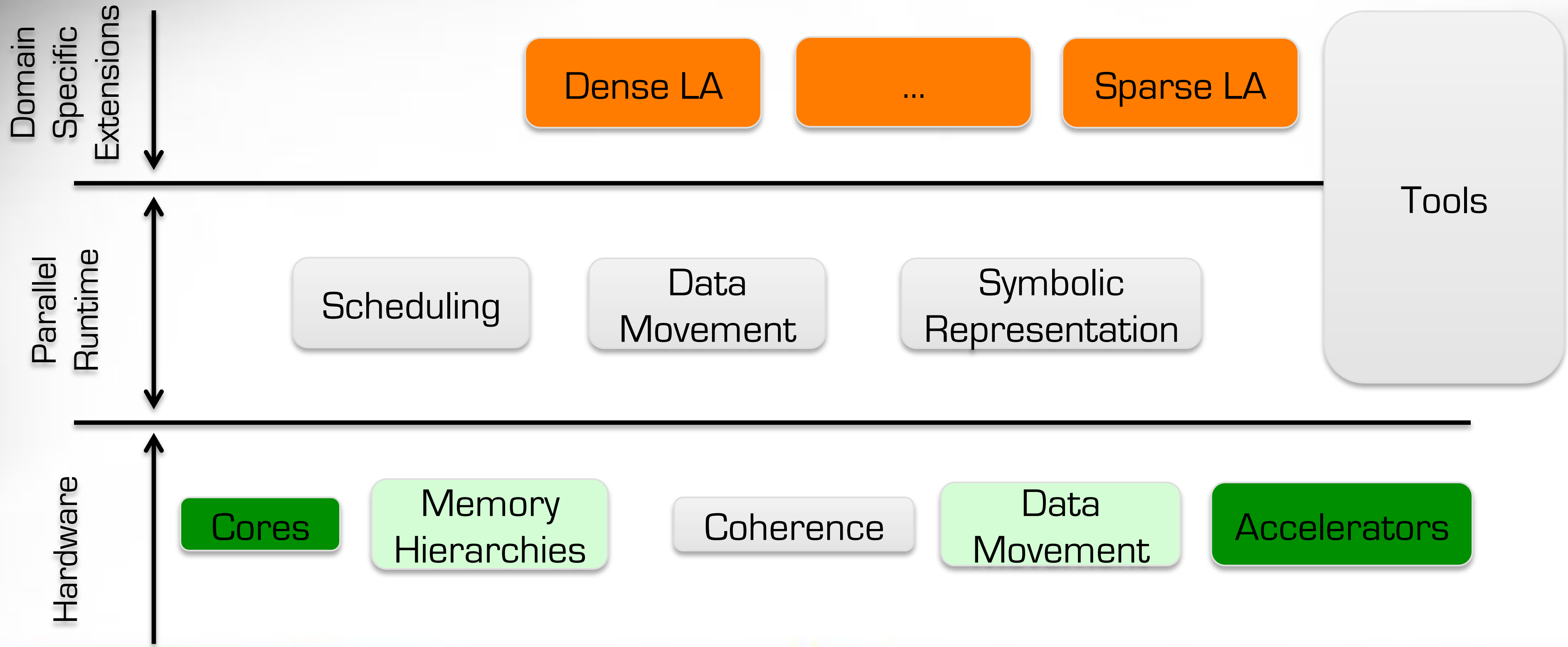


DSBP

[22] F. G. Gustavson, L. Karlsson, and B. Kågström. Distributed SBP cholesky factorization algorithms with near-optimal scheduling. *ACM Trans. Math. Softw.*, 36(2):1–25, 2009. ISSN 0098-3500. DOI: 10.1145/1499096.1499100.

81 dual Intel Xeon L5420@2.5GHz
(2x4 cores/node) ➔ 648 cores
MX 10Gbs, Intel MKL, Scalapack

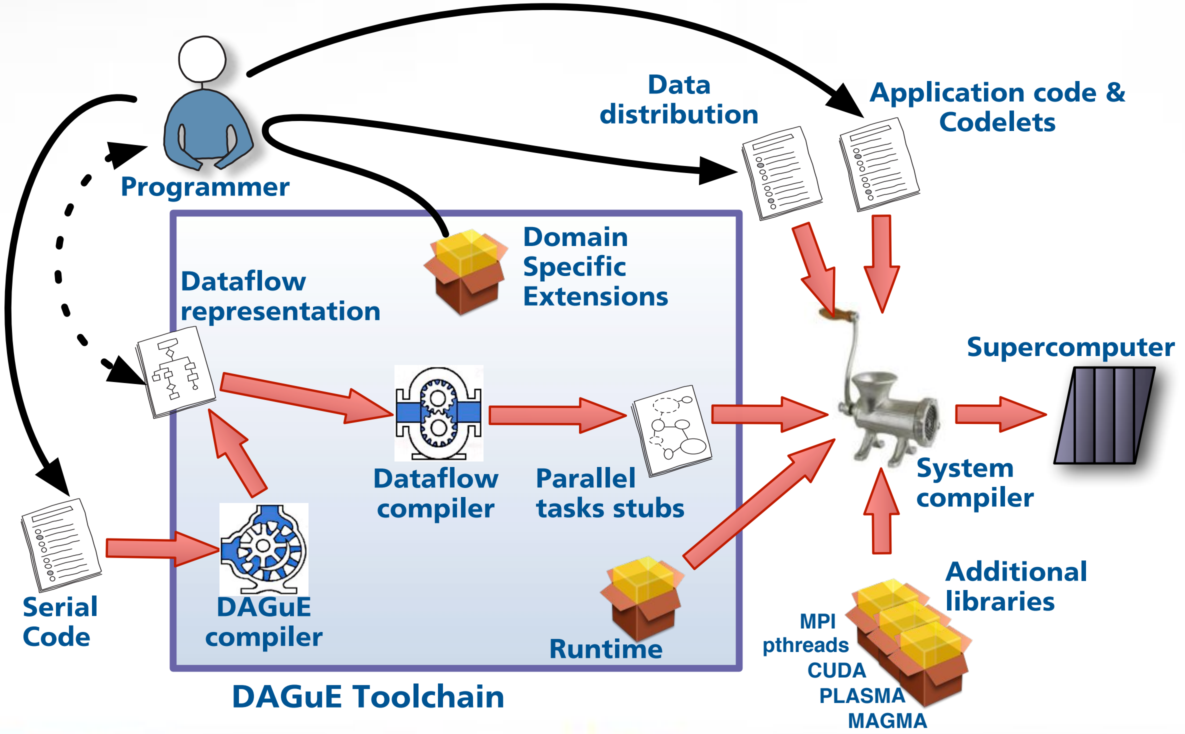
The DAGuE framework

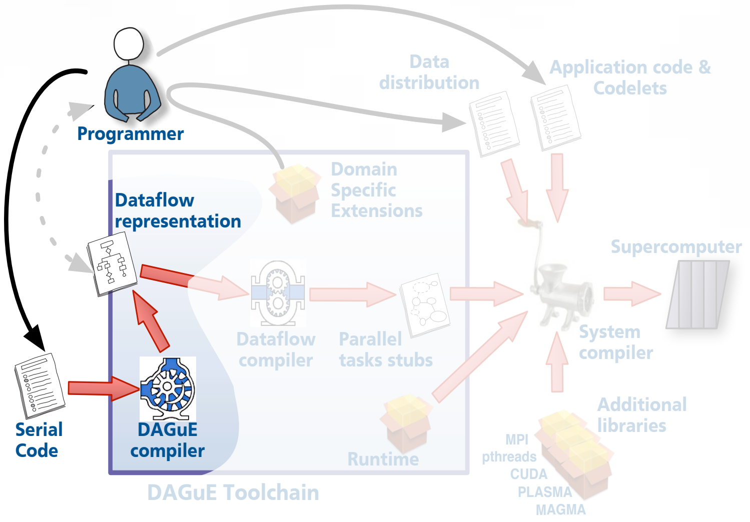


Domain Specific Extensions

- DSEs $\Rightarrow$ higher productivity for developers
  - High-level data types & ops tailored to domain
    - E.g., relations, matrices, triangles, …
  - Prototyping / Meta-Programming
- Portable and scalable specification of parallelism
  - Automatically adjust data structures, mapping, and scheduling as systems scale up
  - Toolkit of classical data distributions, etc

DAGuE toolchain



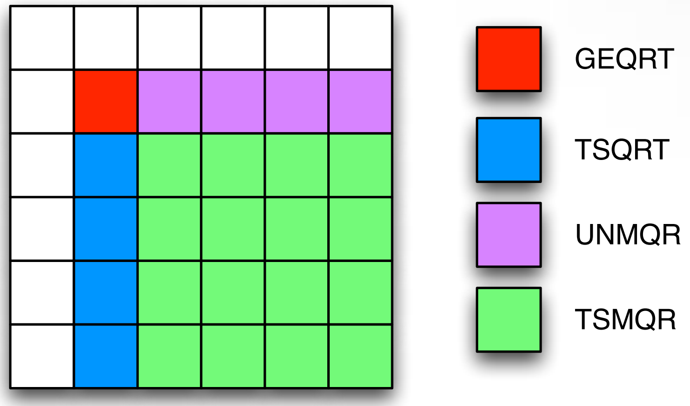


DAGuE Compiler

Serial Code to Dataflow Representation

Example: QR Factorization

```
FOR k = 0 .. SIZE - 1

    A[k][k], T[k][k]  <-  GEQRT( A[k][k] )

    FOR m = k+1 .. SIZE - 1

        A[k][k]|Up, A[m][k], T[m][k]  <-
            TSQRT( A[k][k]|Up, A[m][k], T[m][k] )

    FOR n = k+1 .. SIZE - 1

        A[k][n] <- UNMQR( A[k][k]|Low, T[k][k], A[k][n] )

        FOR m = k+1 .. SIZE - 1

            A[k][n], A[m][n] <-
                TSMQR( A[m][k], T[m][k], A[k][n], A[m][n] )
```



Input Format – Quark (PLASMA)

```
for (k = 0; k < A.mt; k++) {
  Insert_Task( zgeqrt,  A[k][k], INOUT,
                        T[k][k], OUTPUT);
  for (m = k+1; m < A.mt; m++) {
    Insert_Task( ztsqrt,  A[k][k], INOUT | REGION_D|REGION_U,
                          A[m][k], INOUT | LOCALITY,
                          T[m][k], OUTPUT);
  }
  for (n = k+1; n < A.nt; n++) {
    Insert_Task( zunmqr,  A[k][k], INPUT | REGION_L,
                          T[k][k], INPUT,
                          A[k][m], INOUT);
    for (m = k+1; m < A.mt; m++) {
      Insert_Task( ztsmqr, A[k][n], INOUT,
                           A[m][n], INOUT | LOCALITY,
                           A[m][k], INPUT,
                           T[m][k], INPUT);

    }
  }
}
```

- Sequential C code
- Annotated through QUARK-specific syntax
  - Insert_Task
  - INOUT, OUTPUT, INPUT
  - REGION_L, REGION_U, REGION_D, …
  - LOCALITY

Dataflow Analysis

MEM

Incoming Data

Outgoing Data

k = SIZE-1

k = 0

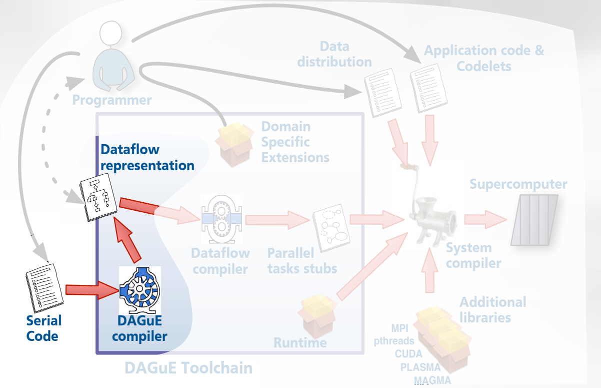
```
FOR k = 0 .. SIZE - 1

  A[k][k], T[k][k]  <-  GEQRT( A[k][k] )

  FOR m = k+1 .. SIZE - 1

    A[k][k]|Up, A[m][k], T[m][k]  <-
      TSQRT( A[k][k]|Up, A[m][k], T[m][k] )

  FOR n = k+1 .. SIZE - 1

    A[k][n]  <-  UNMQR( A[k][k]|Low, T[k][k], A[k][n] )

    FOR m = k+1 .. SIZE - 1

      A[k][n], A[m][n]  <-
        TSMQR( A[m][k], T[m][k], A[k][n], A[m][n] )
```

UPPER

LOWER

n = k+1

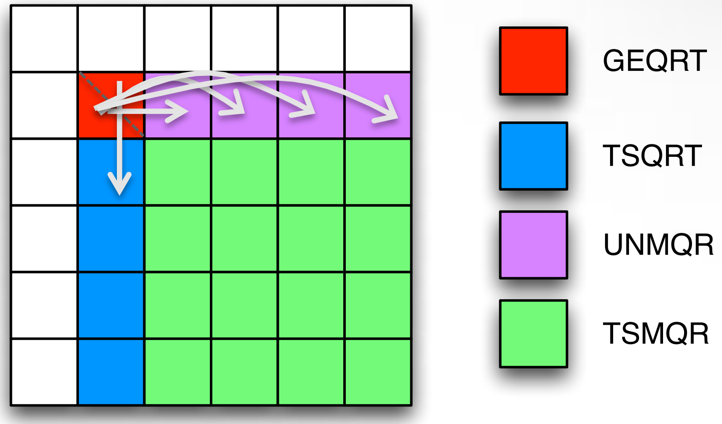
m = k+1



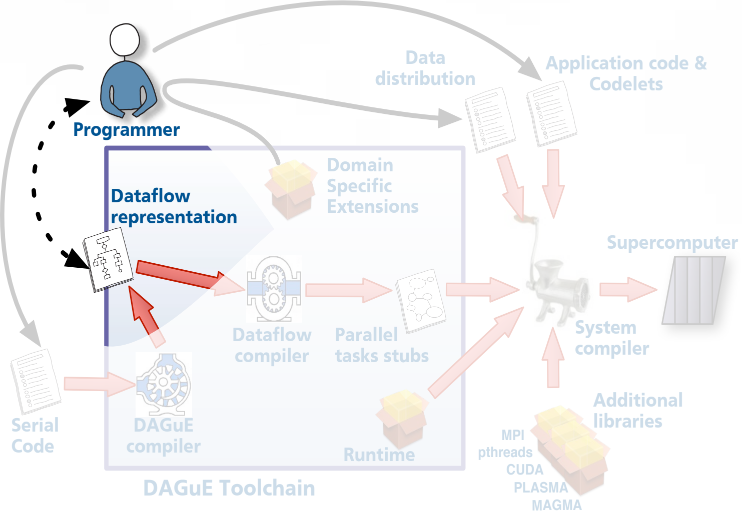
- data flow analysis
  - Example on task DGEQRT of QR
  - Polyhedral Analysis through Omega Test
  - Compute algebraic expressions for:
    - Source and destination tasks
    - Necessary conditions for that data flow to exist

Intermediate Representation: Job Data Flow

```
GEQRT(k)
 /* Execution space */
 k = 0..( MT < NT ) ? MT-1 : NT-1 )
 /* Locality */
 : A(k, k)
 RW     A <- (k == 0)     ? A(k, k)
                          : A1 TSMQR(k-1, k, k)
        -> (k < NT-1)  ? A UNMQR(k, k+1 .. NT-1)   [type = LOWER]
        -> (k < MT-1)  ? A1 TSQRT(k, k+1)          [type = UPPER]
        -> (k == MT-1) ? A(k, k)                   [type = UPPER]
 WRITE T <- T(k, k)
        -> T(k, k)
        -> (k <  NT-1) ? T UNMQR(k, k+1 .. NT-1)
 /* Priority */
 ;(NT-k)*(NT-k)*(NT-k)

BODY
   zgeqrt( A, T )
END
```



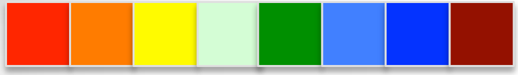
Control flow is eliminated, therefore maximum parallelism is possible



JDF

Dataflow Representation

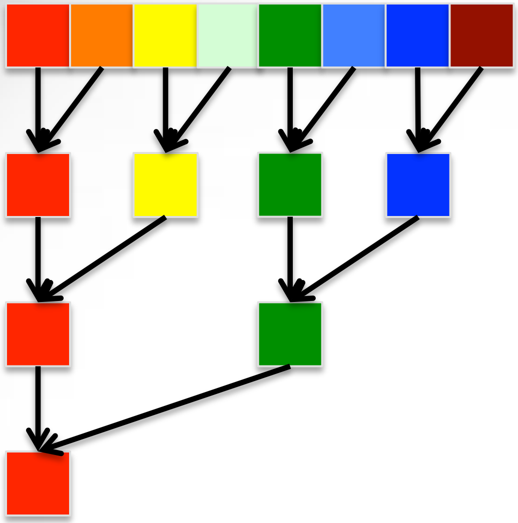
Example: Reduction Operation



- Reduction: apply a user defined operator on each data and store the result in a single location.

  (Suppose the operator is associative and commutative)

Example: Reduction Operation



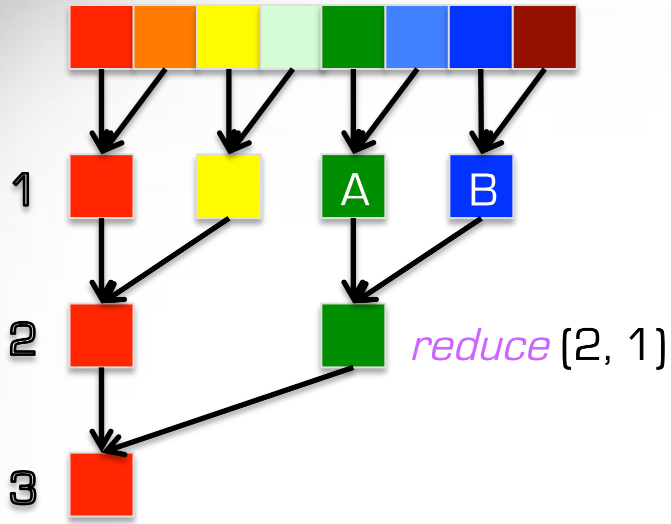
- Reduction: apply a user defined operator on each data and store the result in a single location.

  (Suppose the operator is associative and commutative)

```
for(s = 1; s < N/2; s = 2*s)
  for(i = 0; i < N-s; i += s)
    operator(V[i], V[i+s])
```

Issue: Non-affine loops lead to non-polyhedral array accessing

Example: Reduction Operation

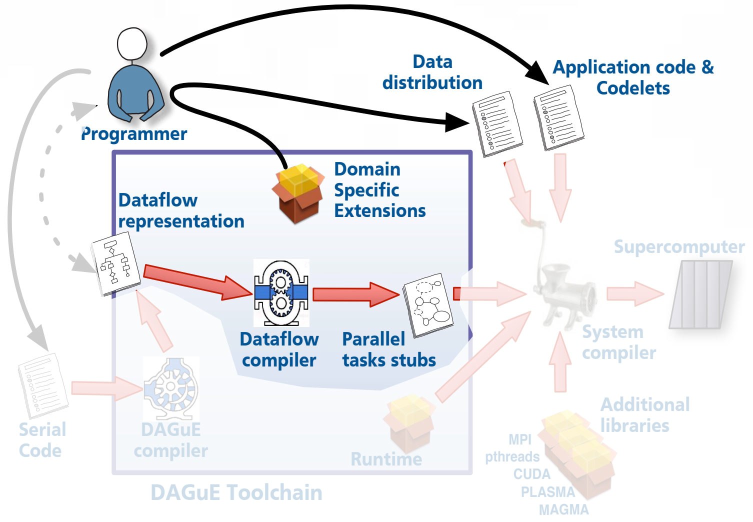


Solution: Hand-writing of the data dependency using the intermediate Data Flow representation

```
reduce(l, p)
l = 1 .. depth
p = 0 .. (MT / (1<<l))

: V(p * (1<<l))

RW   A <- (1 == l) ? V(2*p)
                      : A reduce( l-1, 2*p )
          -> (depth == l) ? V(0)
          -> (0 == (p%2)) ? A reduce(l+1, p/2)
                          : B reduce(l+1, p/2)
READ B <- (1 == l) ? V(2*p+1)
                      :  A reduce( l-1, p*2+1 )
BODY
  operator(A, B);
END
```

Data Flow Compiler

Integration



Data Flow Compiler

- Produces functions to instantiate the DAG object
  - At runtime a DAG object is still problem-size independent, it is just a set of functions to obtain successors or predecessors of tasks, compute the set of initial tasks.

```
dague_object_t *reduce_create(
        dague_ddesc_t *V,
        int MT,
        int depth);

void reduce_destroy(dague_object_t *o);
```

```
reduce(l, p)
l = 1 .. depth
p = 0 .. (MT / (1<<l))

: V(p * (1<<l))
```

Data Distribution

- Flexible data distribution
  - Decoupled from the algorithm
    - But can be exposed
  - Expressed as a user-defined function
  - Only limitation: must evaluate uniformly across all nodes
- Common distributions provided in DSEs
  - 1D cyclic, 2D cyclic, etc.
  - Symbol Matrix for sparse direct solvers

```
dague_ddesc_t *V;
V  = dague_onedim_bc(
        PTR,
        DAGUE_FLOAT,
        worldsize,
        M);
```

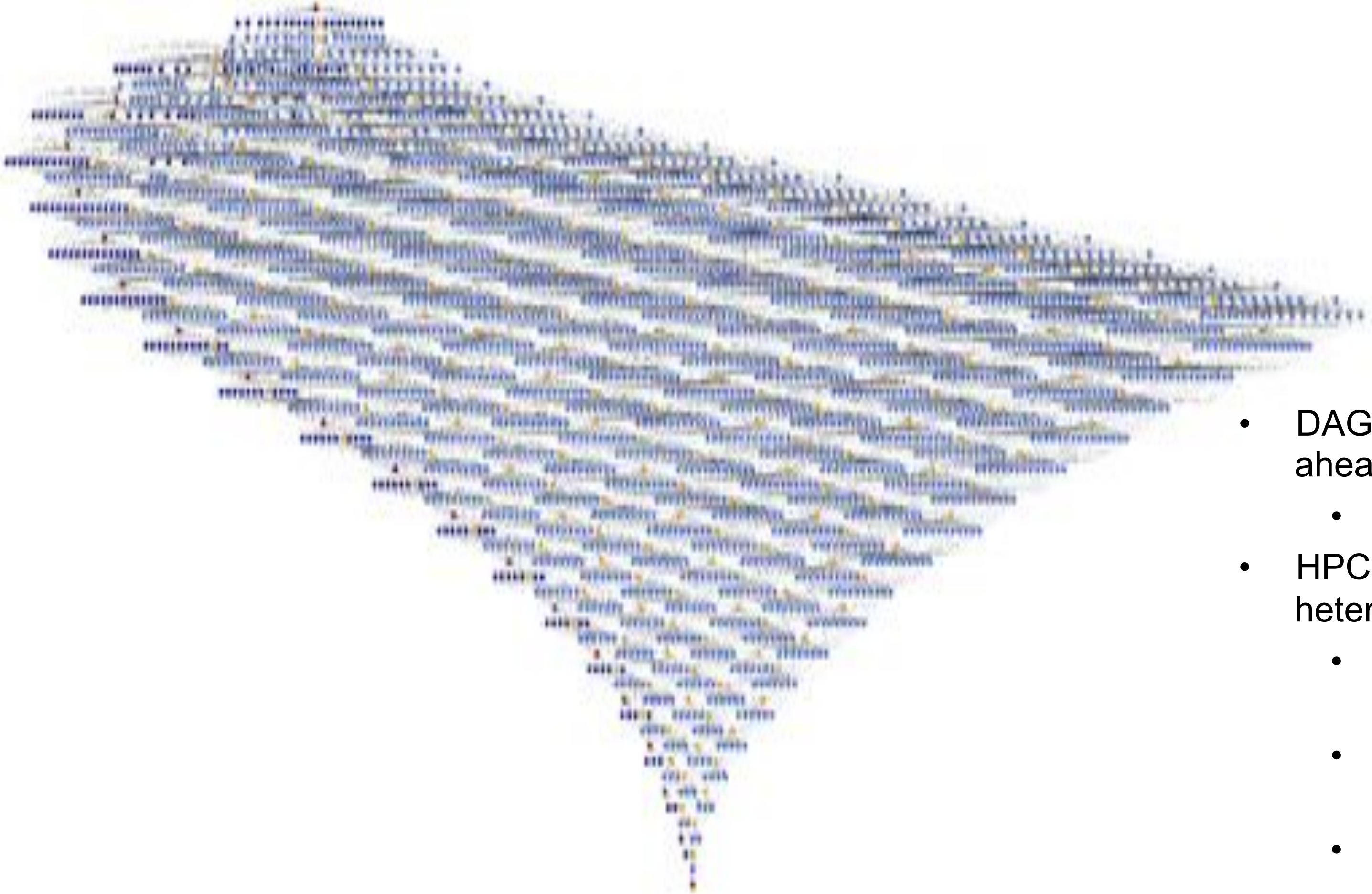
Main Program

- Traditional MPI program
- Initialize / Finalize DAGuE runtime
- Create DAGuE data descriptors
- Instantiate DAGuE DAG objects with parameters and descriptors
- Enqueue them
- Wait for completion
  - During this time, no MPI call can be issued

```
int main(...) {
  MPI_Init(...);
  dague_init(cores, worldsize, ...);
  dague_ddesc_t * V = ...;
  dague_object_t * r =
            reduce_create(V, ...);
  dague_enqueue(r);
  dague_wait(r);
  reduce_destroy(r);
  dague_fini();
  MPI_Finalize();
}
```

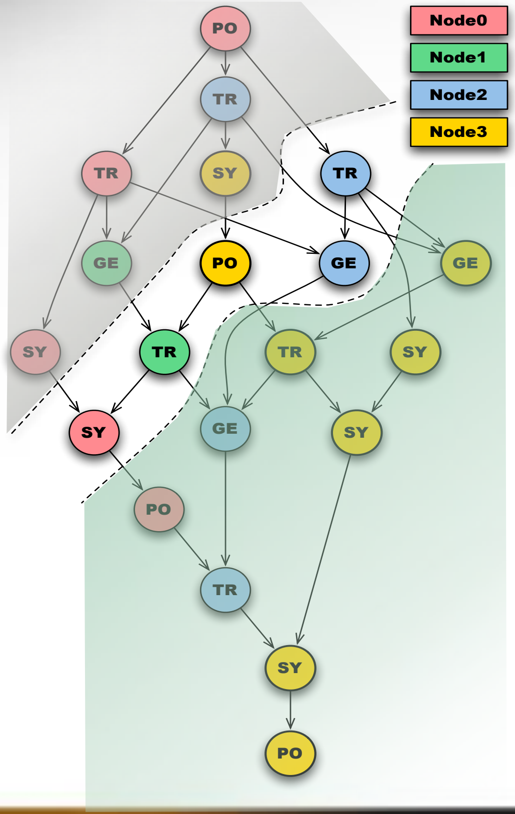
Algorithm is now expressed as a Parameterized DAG

Parallel Runtime



- DAG too large to be generated ahead of time
  - Generate it dynamically
- HPC is about distributed heterogeneous resources
  - Have to get involved in message passing
  - Distributed management of the scheduling
  - Dynamically deal with heterogeneity

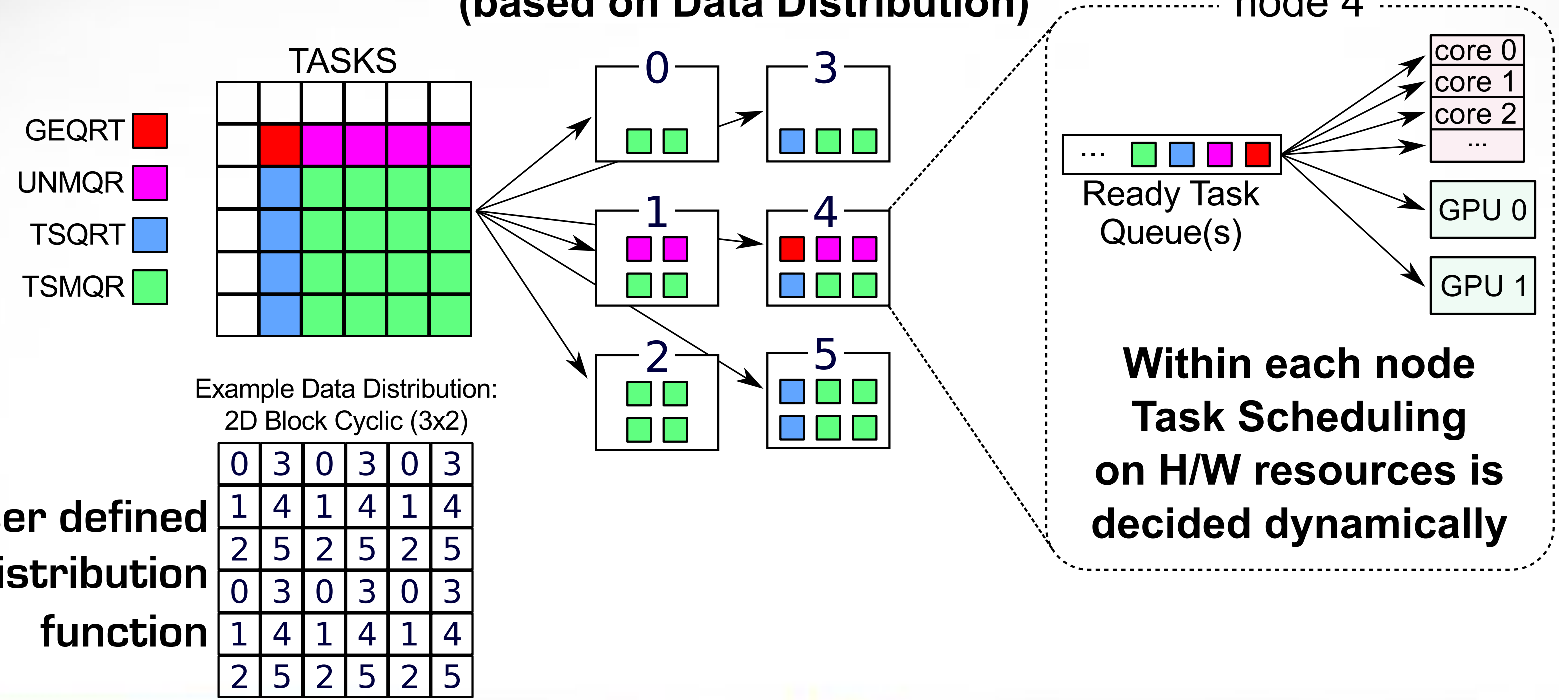
Runtime DAG scheduling



- Every process has the symbolic DAG representation
  - Only the (node local) frontier of the DAG is considered
  - Distributed Scheduling based on remote completion notifications
- Background remote data transfer automatic with overlap
- NUMA / Cache aware Scheduling
  - Work Stealing and sharing based on memory hierarchies

Dynamic / Static

Task Affinity to nodes (based on Data Distribution)



User defined data distribution function

Within each node Task Scheduling on H/W resources is decided dynamically

Scheduling Heuristics in DAGuE

- Manages parallelism & locality
  - Achieve efficient execution (performance, power, …)
  - Handles specifics of HW system (hyper-threading, NUMA, …)
- Per-object capabilities
  - Read-only or write-only, output data, private, relaxed coherence
  - DAGuE engine tracks data usage, and targets to improve data reuse
  - NUMA aware hierarchical bounded buffers to implement work stealing
- Users hints: expressions for distance to critical path
  - Selection from local waiting queue abides to priority, but work stealing can alter this ordering due to locality
- Communications heuristics
  - Communications inherits priority of destination task
- Algorithm defined scheduling

PTG vs DAG Unrolling

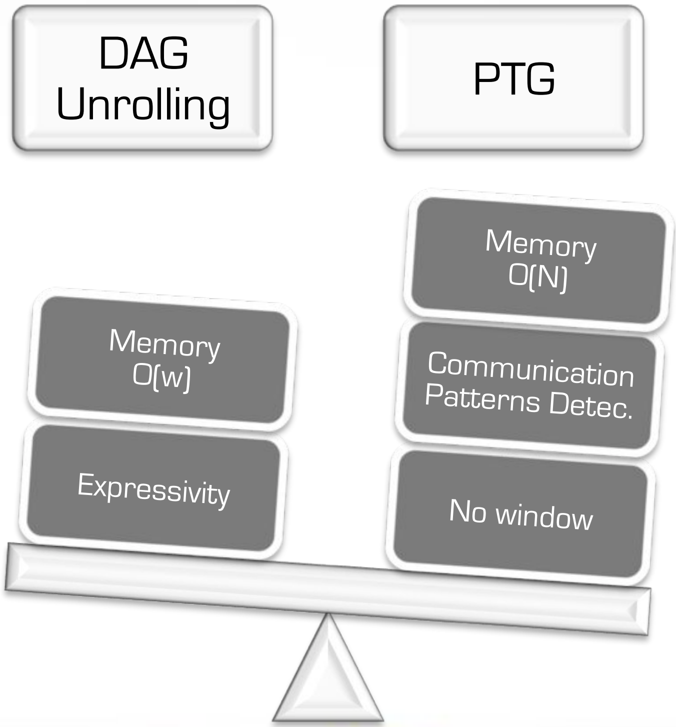
DAG Unrolling

- Discover the DAG while unrolling a sequential code
- StarPU, SMP*, PLASMA, … popular approach
- Window-Based (the DAG is huge)

Parameterized Task Graph

- DAGuE, PTG approach
- Problem-size independent object represents the whole DAG
- Given a task (the parameters of a terminated task), can compute the successors in O(d) (O(1))
- From these successors, can keep only the local & ready ones

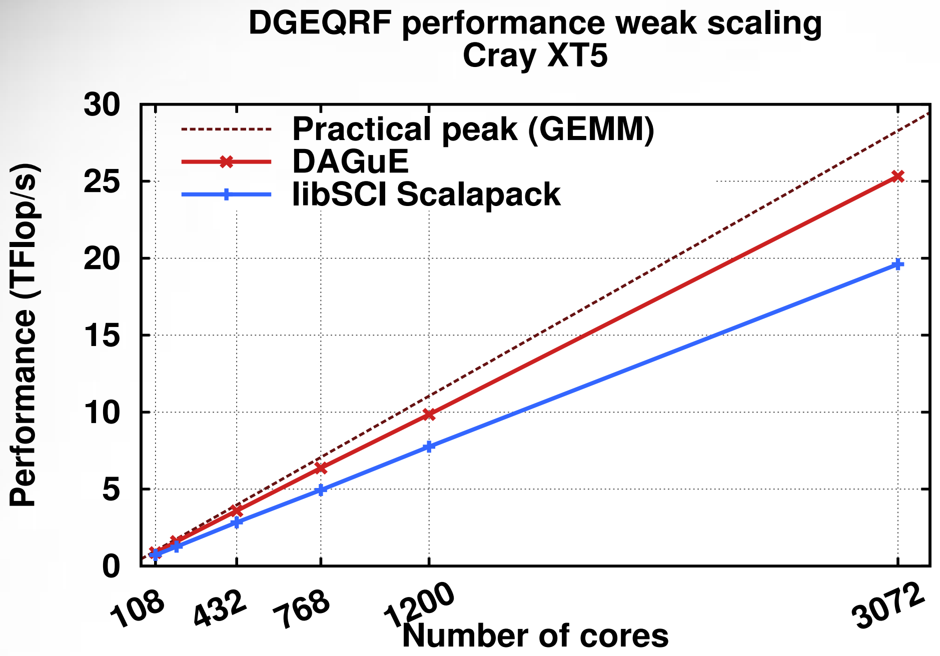
PTG vs DAG Unrolling (2)



Performance; Ongoing Work

Performance; Ongoing Work

Scalability in Distributed Memory



- Parameterized Task Graph representation
- Independent distributed scheduling

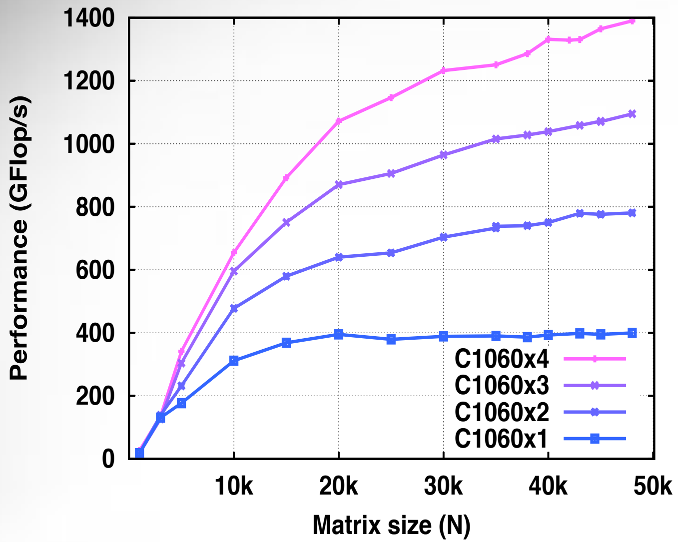
➡ Scales well

Heterogeneity Support

```
/* POTRF Lower case */
GEMM(k, m, n)

// Execution space
k = 0   .. MT-3
m = k+2 .. MT-1
n = k+1 .. m-1

// Parallel partitioning
: A(m, n)

// Parameters
READ  A <- C TRSM(m, k)
READ  B <- C TRSM(n, k)
RW    C <- (k == 0)   ? A(m, n)      : C GEMM(k-1, m, n)
        -> (n == k+1) ? C TRSM(m, n) : C GEMM(k+1, m, n)

BODY [CPU, CUDA, MIC, *]
```

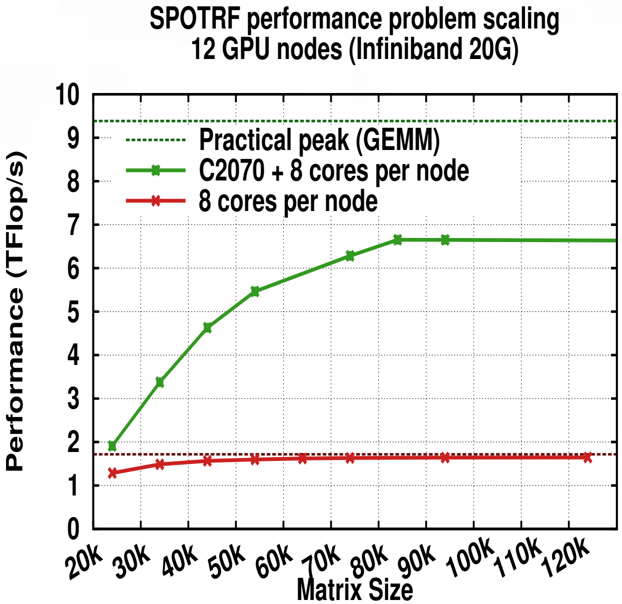
- A BODY is a task on a specific device (codelet)
- Currently the system supports CUDA and cores
- A CUDA device is considered as one additional memory level
- Data locality and data versioning define the transfers to and from the GPU/Co-processors

- Multi GPU – single node



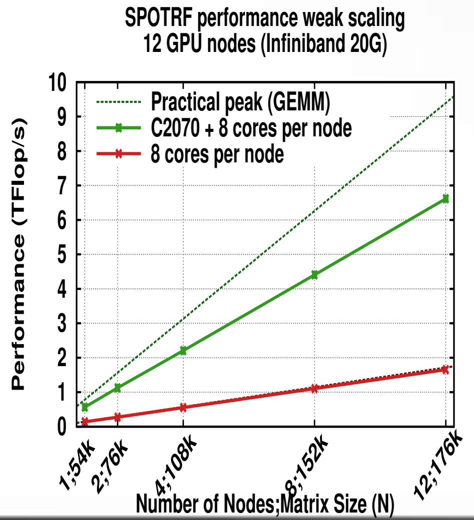
- Single node
- 4xTesla (C1060)
- 16 cores (AMD opteron)

- Multi GPU - distributed

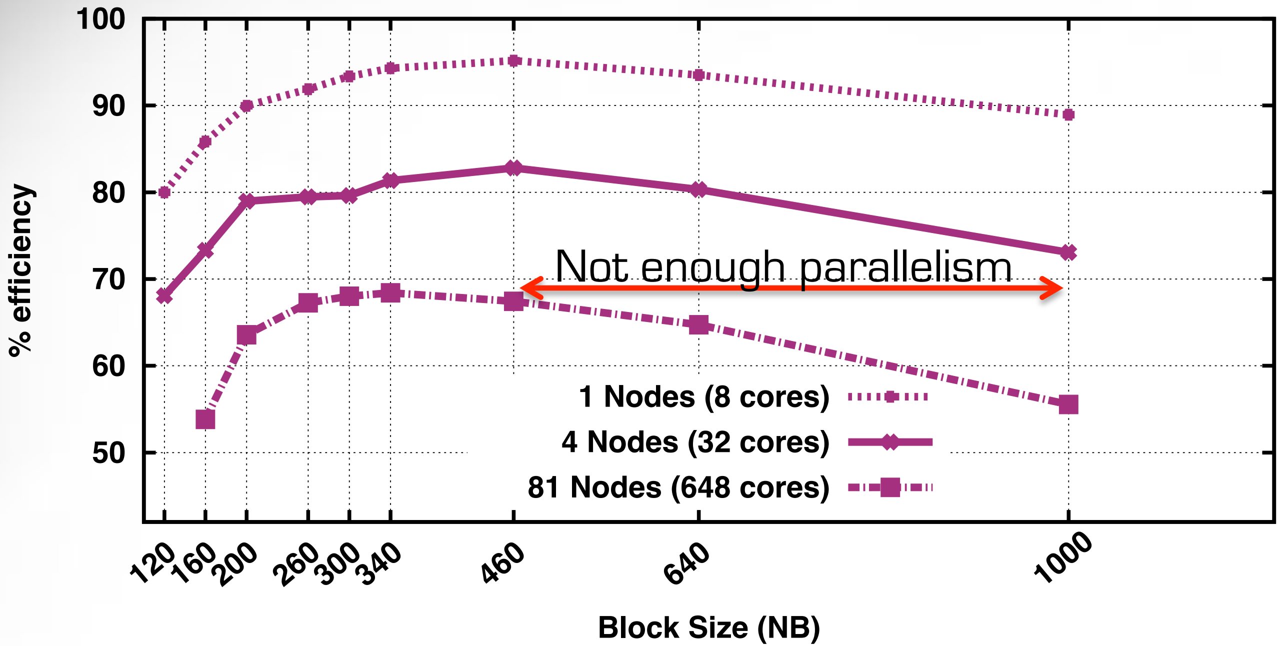


- 12 nodes
- 12xFermi (C2070)
- 8 cores/node (Intel core2)

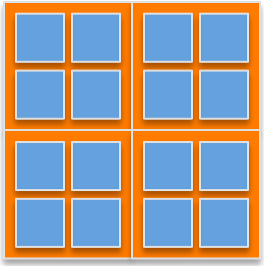
Scalability



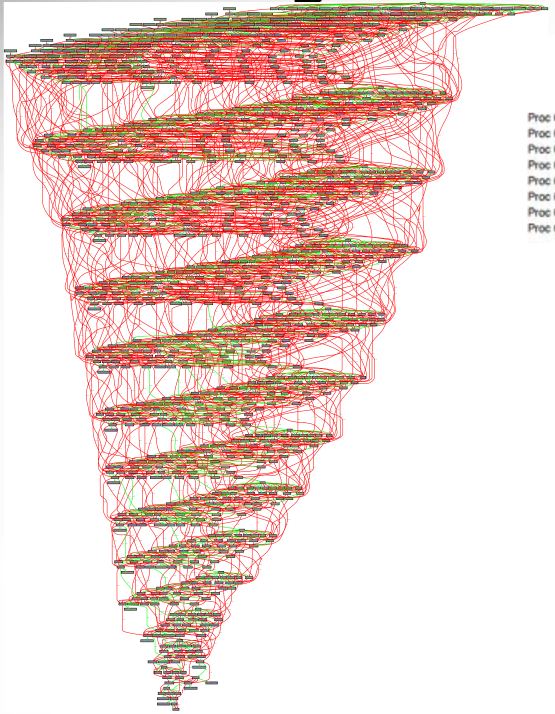
Auto-tuning



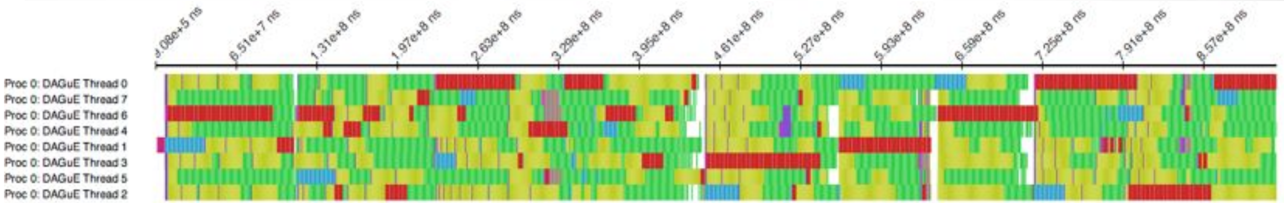
- Multi-level tuning
  - Tune the kernels based on local architecture
  - Then tune the algorithm
- Depends on the network, type and number of cores
- For a fixed size matrix increasing the task duration (or the tile size) decrease parallelism
- For best performance: auto-tune per system

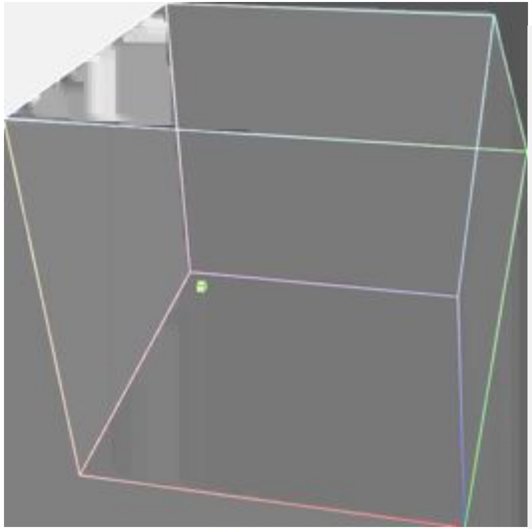


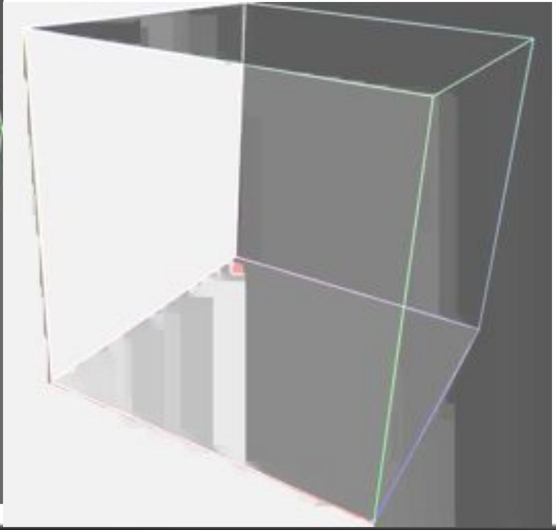
Analysis Tools



Hermitian Band Diagonal; 16x16 tiles

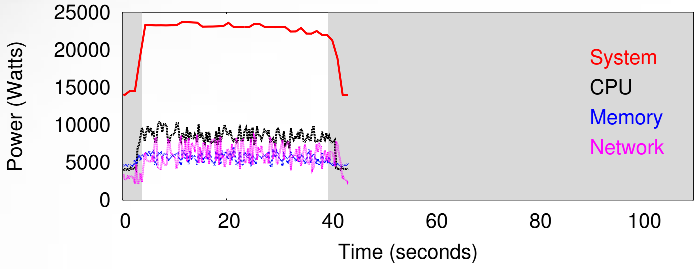




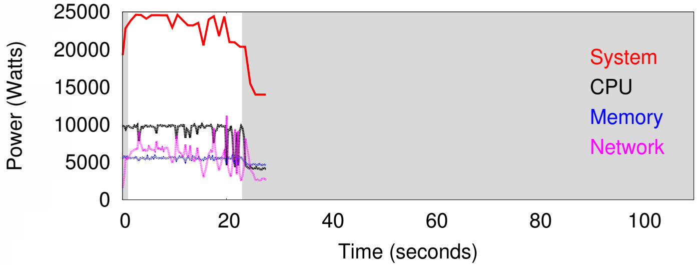


Energy efficiency

QR factorization (256 cores)



(a) ScaLAPACK.



(b) DPLASMA.

Total energy consumption

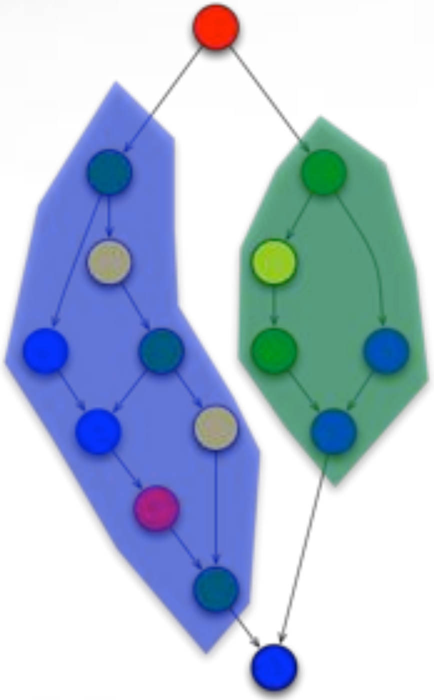
# Cores	Library	Cholesky	QR
128	ScaLAPACK	192000	672000
	DPLASMA	128000	540000
256	ScaLAPACK	240000	816000
	DPLASMA	96000	540000
512	ScaLAPACK	325000	1000000
	DPLASMA	125000	576000

Work in progress with Hatem Ltaief

- Energy used depending on the number of cores
- Up to 62% more energy efficient while using a high performance tuned scheduling
  - Power efficient scheduling

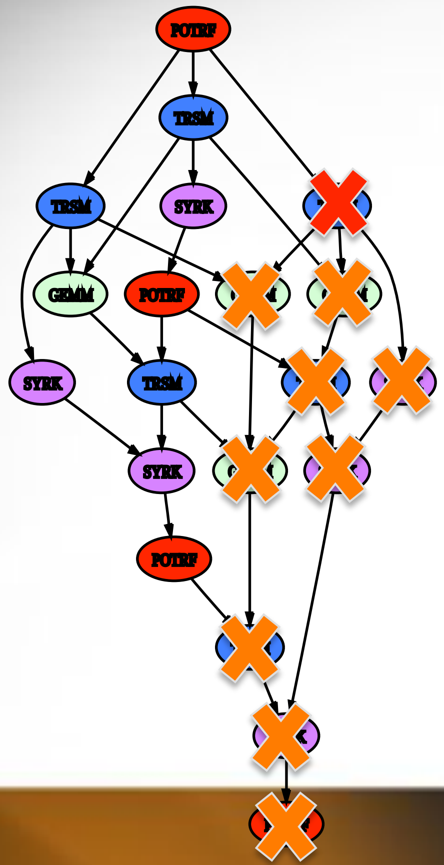
SystemG: Virginia Tech Energy Monitored cluster (ib40g, intel, 8cores/node)

(Runtime) Choice



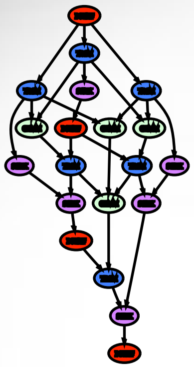
- Take one of the two branches
- “cancel” the branch that was not taken
  - Remember the choices in the dague_object
  - Broadcast the choices for distributed runs

Resilience



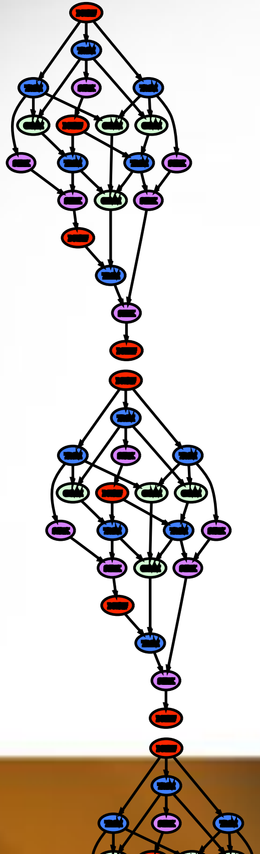
- The fault propagate in the system based on the data dependencies
- However, if the original data can be recovered, the execution complete without user interaction
- Automatic recovery made simple

Composition



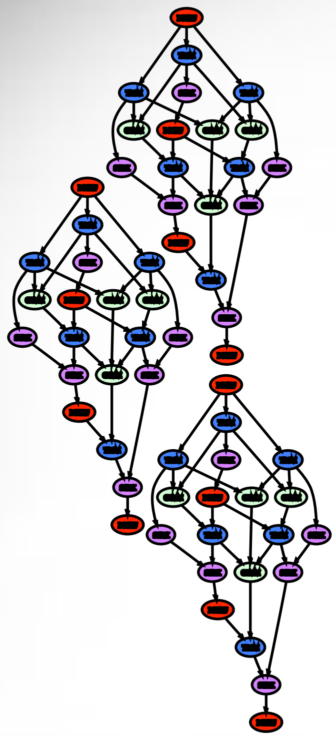
- An algorithm is a series of operations with data dependencies
- A sequential composition limit the parallelism due to strict synchronizations
  - Following the flow of data we can loosen the synchronizations and transform them in data dependencies

Composition



- An algorithm is a series of operations with data dependencies
- A sequential composition limit the parallelism due to strict synchronizations
  - Following the flow of data we can loosen the synchronizations and transform them in data dependencies

Composition



- An algorithm is a series of operations with data dependencies
- A sequential composition limit the parallelism due to strict synchronizations
  - Following the flow of data we can loosen the synchronizations and transform them in data dependencies

Related Work

Other Systems

	DAGuE	SMPss	StarPU	Charm ++	FLAME	QUARK	Tblas	PTG
Scheduling	Distr. (1/core)	Repl (1/node)	Repl (1/node)	Distr. (Actors)	w/ SuperMatrix	Repl (1/node)	Centr.	Centr.
Language	Internal or Seq. w/ Affine Loops	Seq. w/ add_task	Seq. w/ add_task	Msg-Driven Objects	Internal (LA DSL)	Seq. w/ add_task	Seq. w/ add_task	Internal
Accelerator	GPU	GPU	GPU		GPU	GPU		
Availability	Public	Public	Public	Public	Public	Public	Not Avail.	Not Avail.

Early stage: ParalleX
Non-academic: Swarm, MadLINQ, CnC

All projects support Distributed and Shared Memory
(QUARK with QUARKd; FLAME with Elemental)

History: Beginnings of Data Flow

- "*Design of a separable transition-diagram compiler*", M.E. Conway, Comm. ACM, 1963
  - Coroutines, flow of data between process
- J.B. Dennis, 60's
  - Data Flow representation of programs
  - Reasoning about parallelism, equivalence of programs, …
- "The semantics of a simple language for parallel programming", G. Kahn
  - Kahn Networks

Conclusion

- Programming made easy(ier)
  - Portability: inherently take advantage of all hardware capabilities
  - Efficiency: deliver the best performance on several families of algorithms
  - Higher-level programming:

    New languages should not strive to transform the easy into trivial, but to transform the impracticable into achievable.
- Computer scientists were spoiled by MPI
  - Now let's think about our users
- Let different people focus on different problems
  - Application developers on their algorithms
  - System developers on system issues

The end

Dague /däg/ (French): a short and sharp knife used for a wide variety of purposes