PaRSEC

Parallel Runtime Scheduling and Execution Control

http://icl.utk.edu/parsec

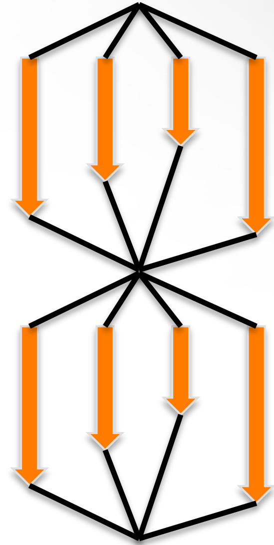
http://icl.utk.edu/dague

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

INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE UT

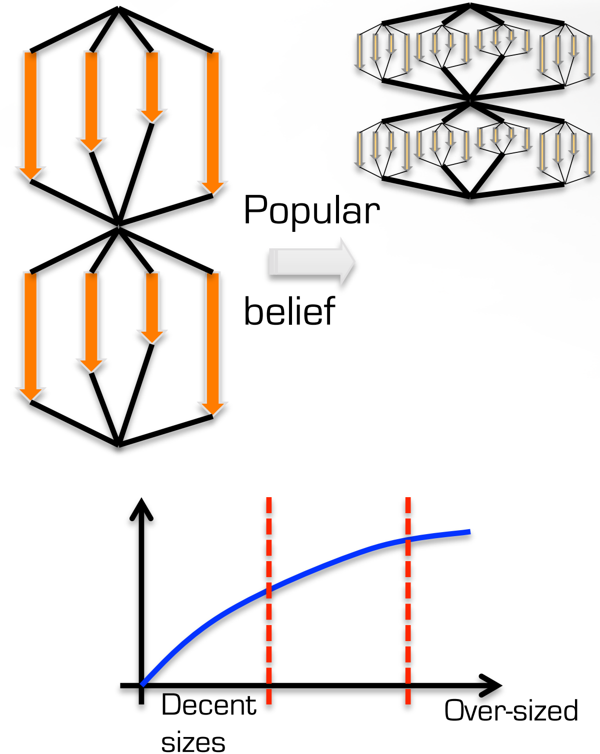
Where we are today

- 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
  - But wait there is more: They are distributed!
  - Consequence is systemic load imbalance
- And we still ask the same question: How to harness these devices productively?
  - SPMD produces choke points, wasted wait times
  - We need to improve efficiency, power and reliability



The missing parallelism

- Too difficult to find parallelism, to debug, maintain and get good performance *for everyone*
- Increasing gaps between the capabilities of today's programming environments, the requirements of emerging applications, and the challenges of future parallel architectures

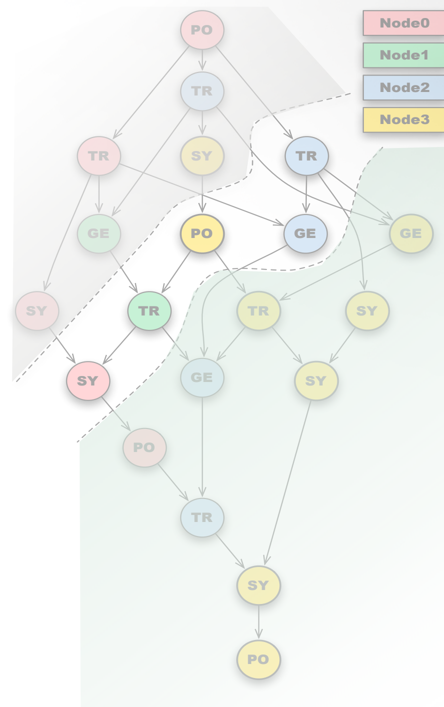


Decouple "System issues" from Algorithm

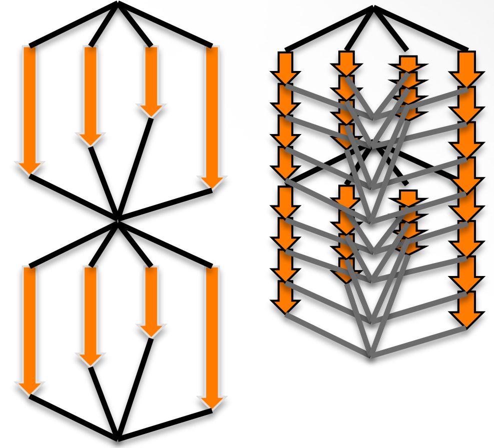
- Keep the algorithm simple
  - Depict only the flow of data between tasks
  - *Distributed Dataflow Environment based on Dynamic Scheduling of (Micro) Tasks*
- Programmability: layered approach
  - Algorithm / Data Distribution
  - Parallel applications without parallel programming
- Portability / Efficiency
  - Use all available hardware; overlap data movements / computation
  - Progress is still possible when imbalance arise

Dataflow with Runtime scheduling

- Algorithms need help to unleash their power
  - *Hardware specificities*: a runtime can provide portability, performance, scheduling heuristics, heterogeneity management, data movement, …
  - *Scalability*: maximize parallelism extraction, but no centralized scheduling or entire DAG unpacking: 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
- The need to express the algorithms differently

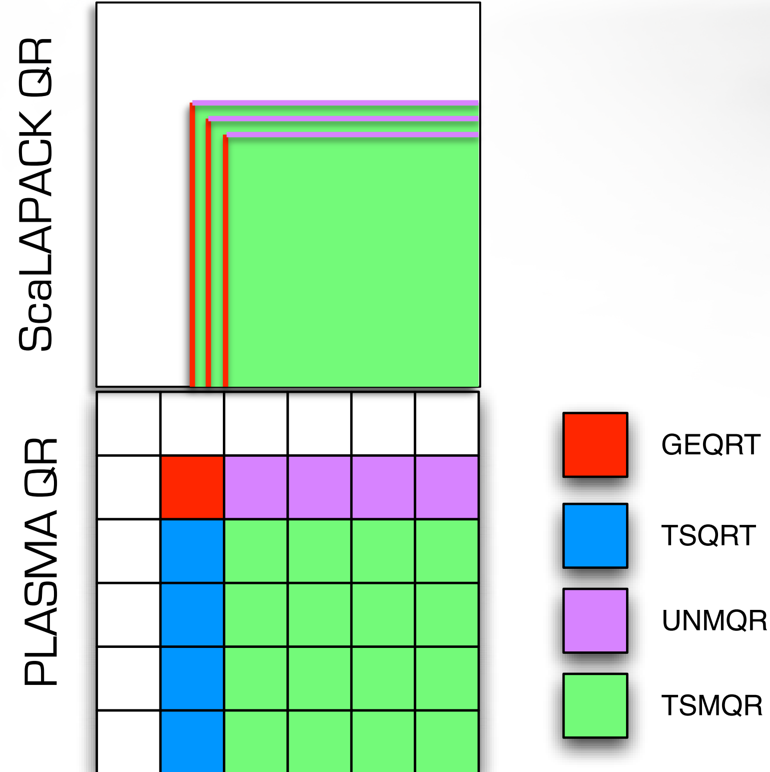


Divide-and-orchestrate

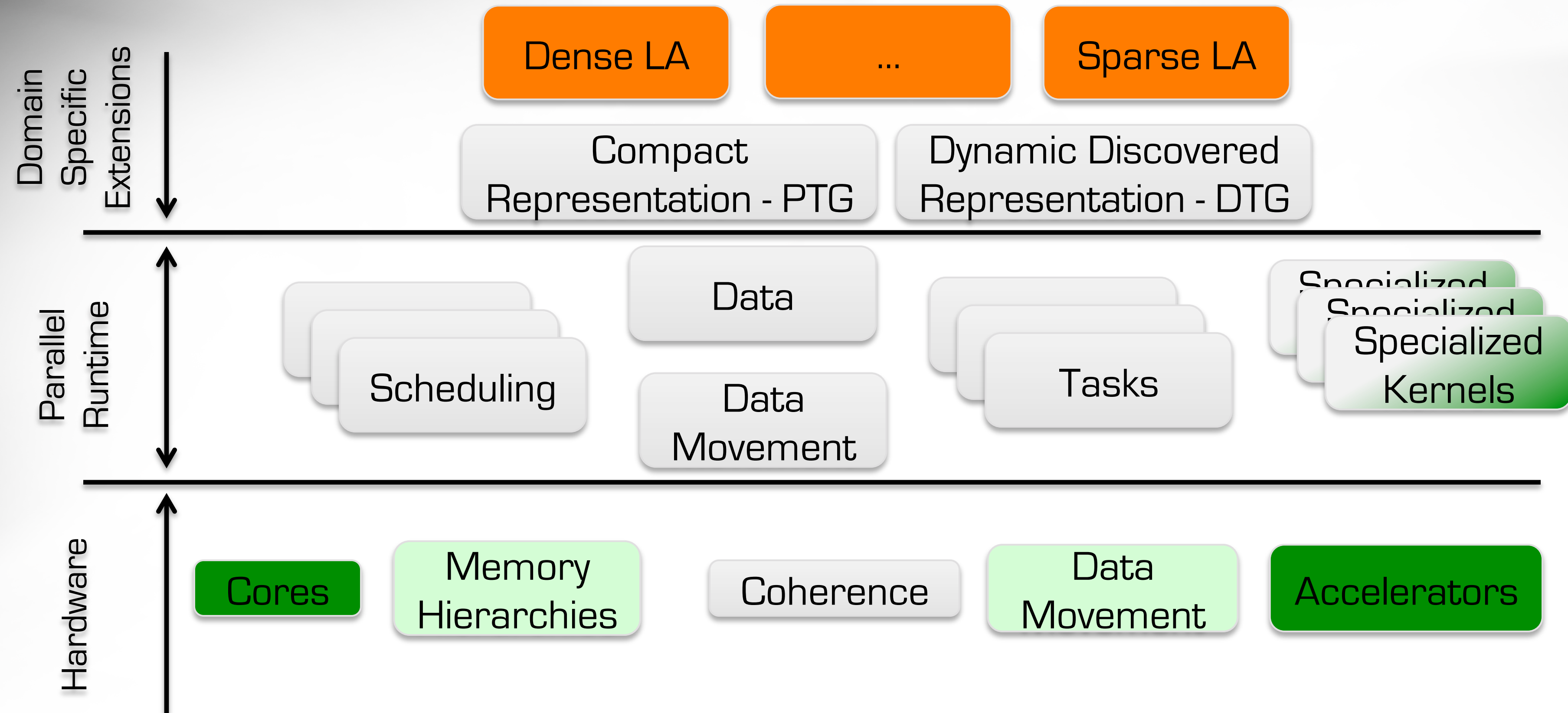


Divide-and-orchestrate

- Leave the optimization of each level to specialized entities
  - Compiler do marvels when the cost of memory accesses is known
- Think hierarchical super-scalar
  - Compiler focus on problems that fit in the cache
  - Humans focus on depicted the flow of data between tasks
  - And a runtime orchestrate the flows to maximize the throughput of the architecture



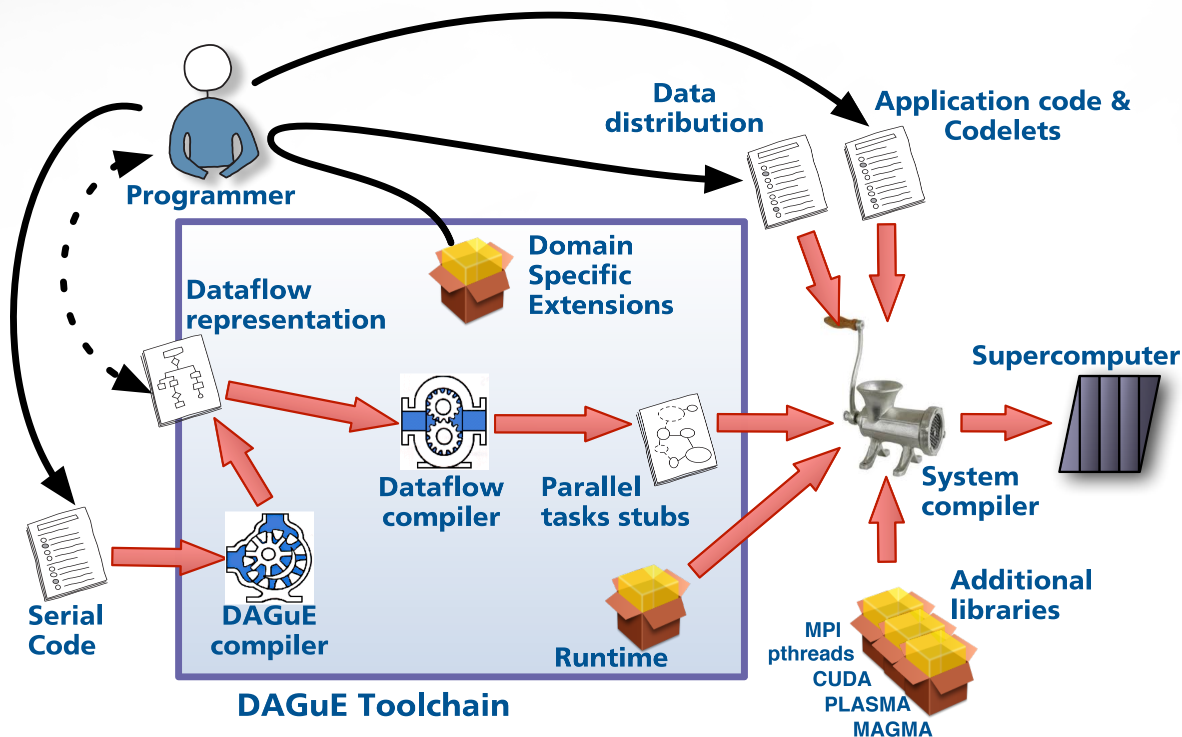
The PaRSEC framework

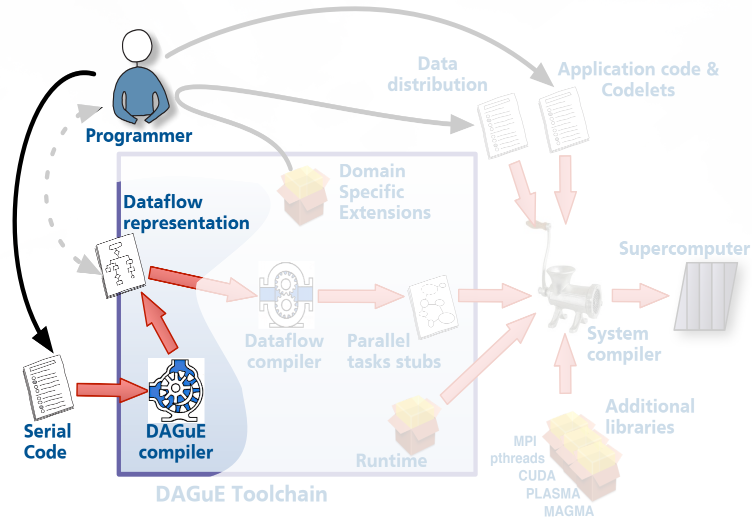


Domain Specific Extensions

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

PaRSEC toolchain



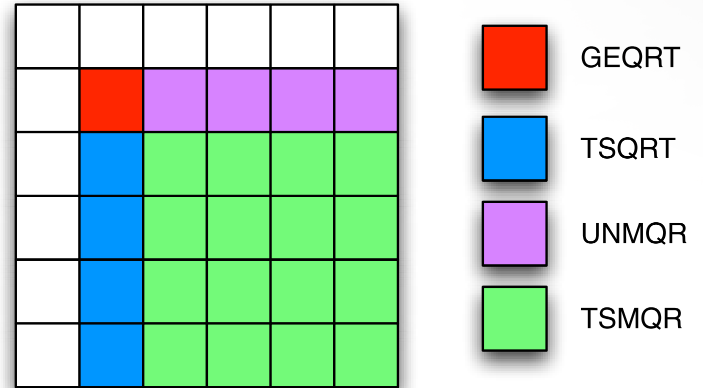


PaRSEC Compiler

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] )
```

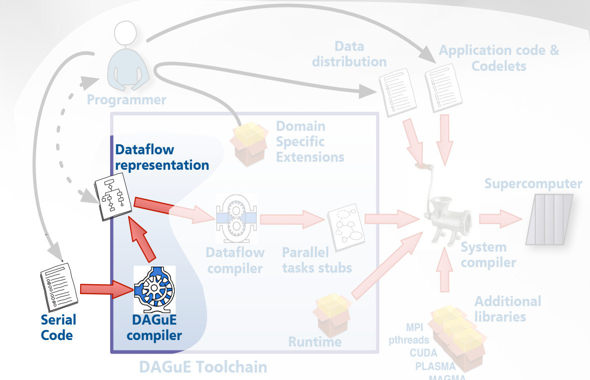


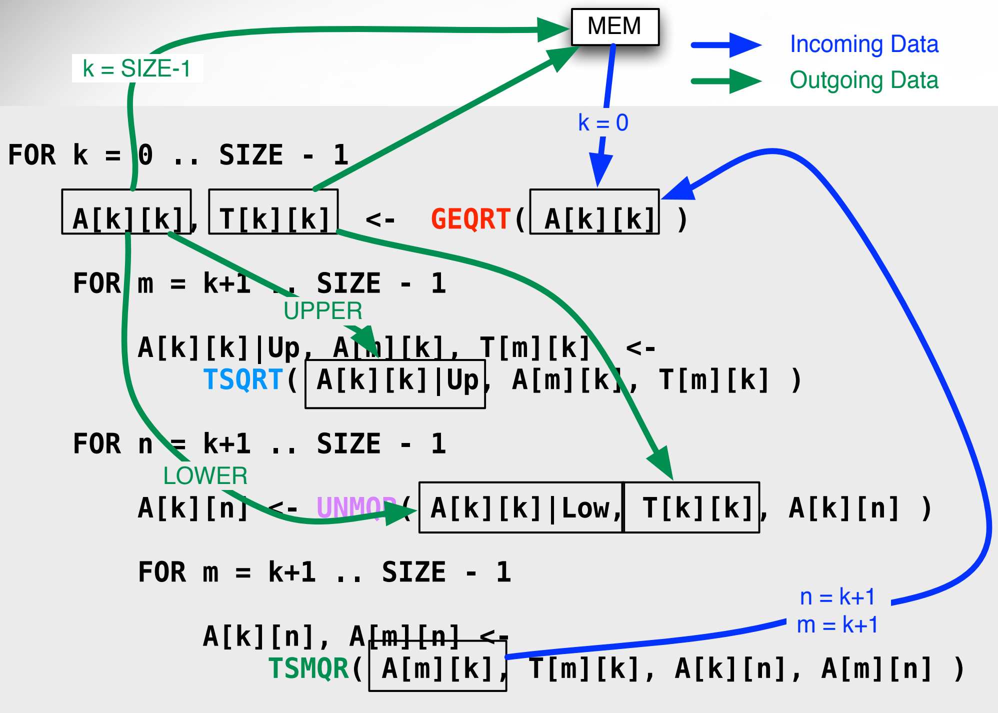
Input Format – Quark/StarPU/MORSE

```
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 some specific syntax
  - Insert_Task
  - INOUT, OUTPUT, INPUT
  - REGION_L, REGION_U, REGION_D, …
  - LOCALITY

Dataflow Analysis

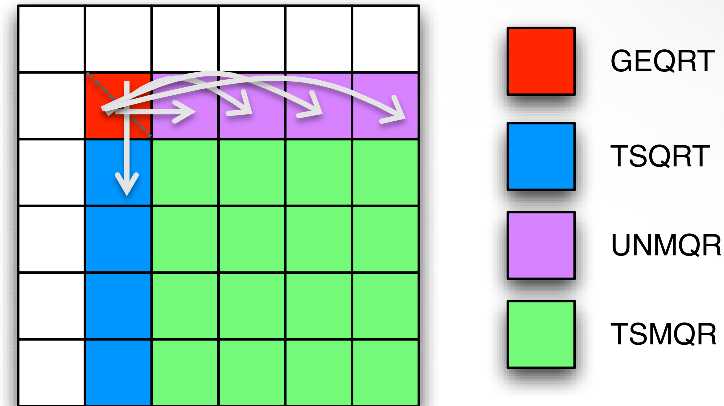




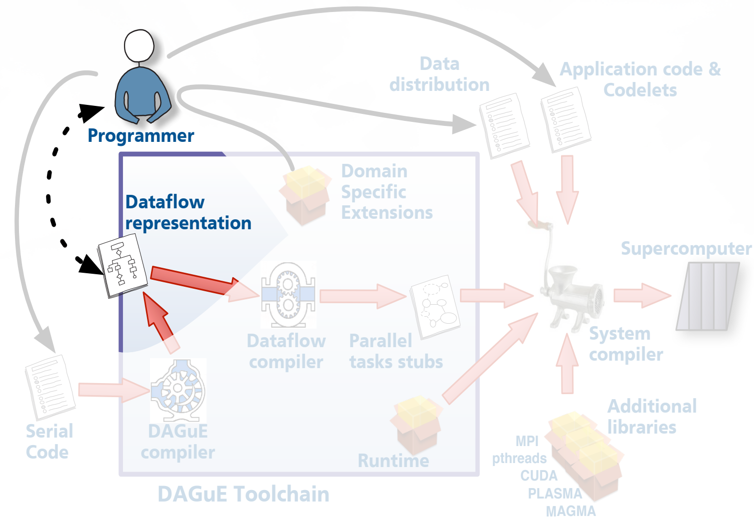
- data flow analysis
  - Example on task DGEQRT of QR
  - Polyhedral Analysis through Omega Test
  - Compute algebraic expressions for:
    - Source and destination tasks
    - Necessary conditions for that data flow to exist

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
```



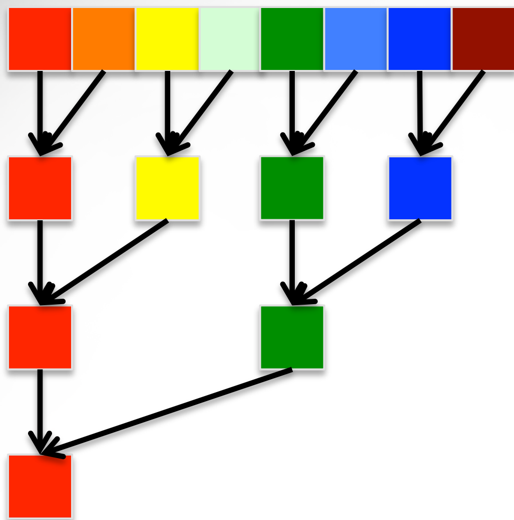
Control flow is eliminated, therefore maximum parallelism is possible



PaRSEC Compiler

Dataflow Representation

Example: Reduction Operation



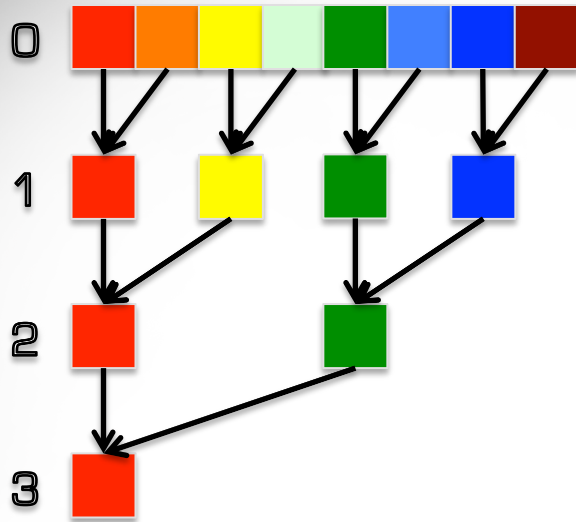
- Reduction: apply a user defined operator on each data and store the result in a single location.

  (Suppose the operator is associative and commutative)

```
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




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

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

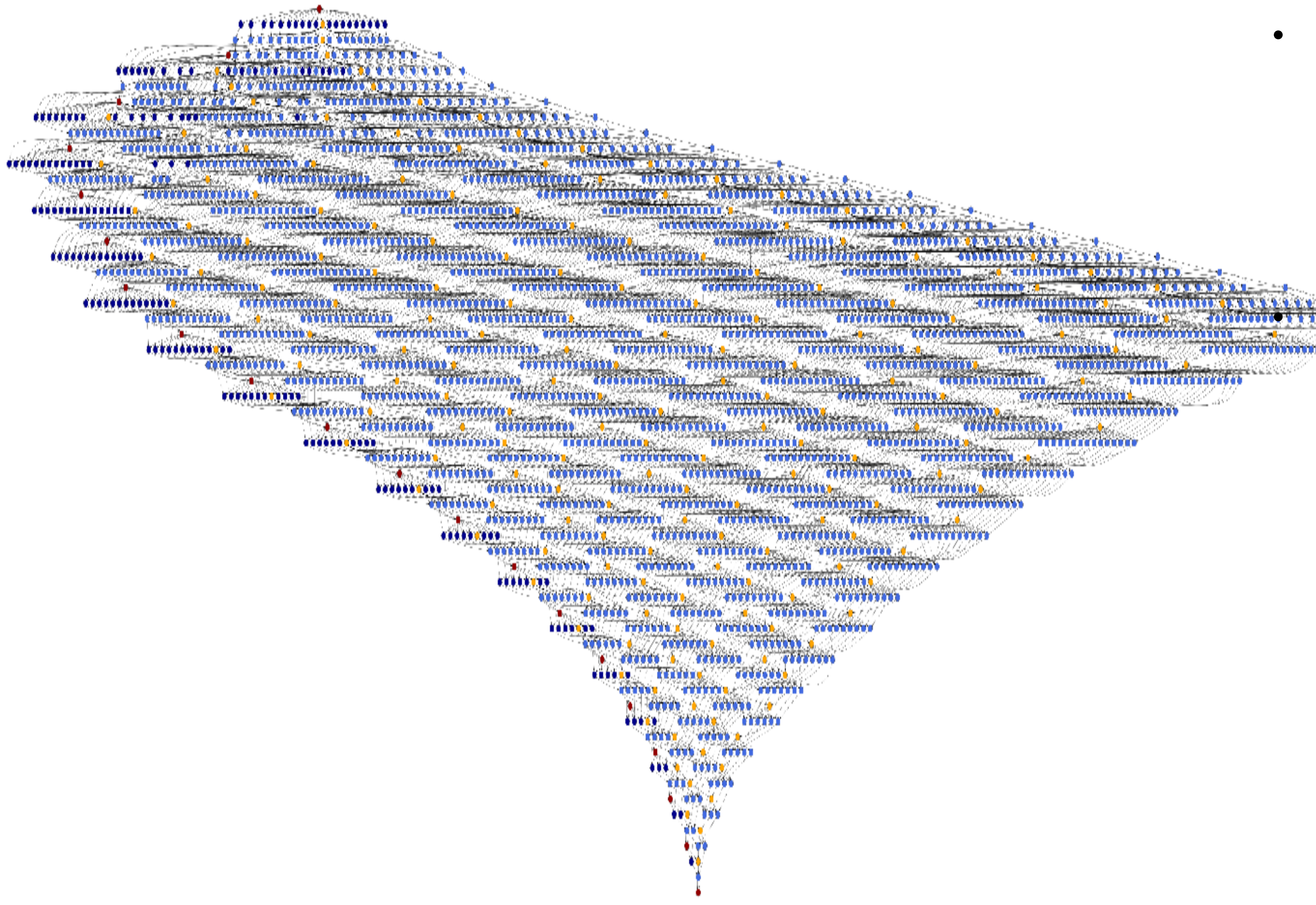
Solution: Hand-writing of the data dependency using the intermediate Data Flow representation

Data/Task Distribution

- Flexible data distribution
  - Decoupled from the algorithm
  - 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

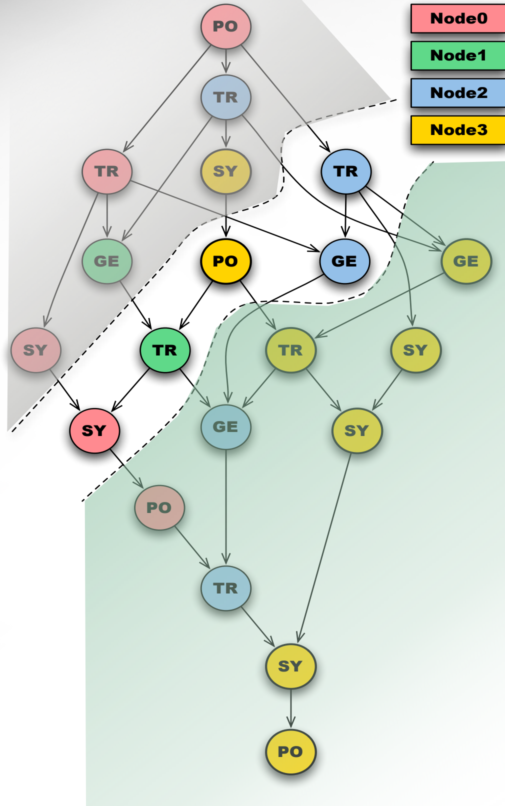
Algorithm is now expressed as a DAG (potentially Parameterized)

Parallel Runtime



- DAG too large to be generated ahead of time
  - Generate it dynamically
  - Merge parameterized DAGs with dynamically generated DAGs
- 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

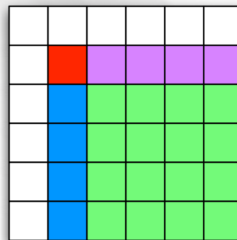
Scheduling Heuristics in PaRSEC

- 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
  - 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
  - Insertion in 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

Now we have a runtime and some algorithms

What’s next?

Auto-tuning



Weak-scaling

% efficiency

100
90
80
70
60
50

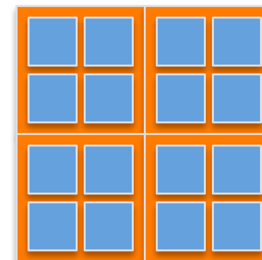
Not enough parallelism

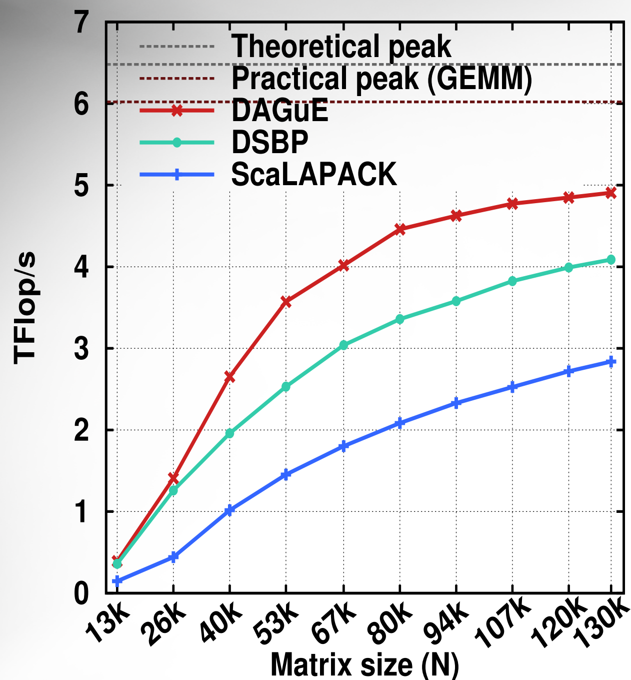
1 Nodes (8 cores)
4 Nodes (32 cores)
81 Nodes (648 cores)

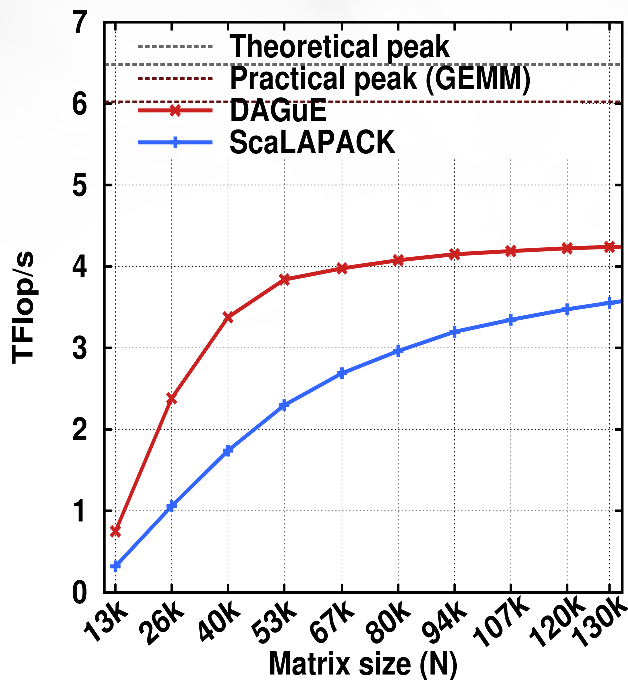
120 160 200 260 300 340 460 640 1000

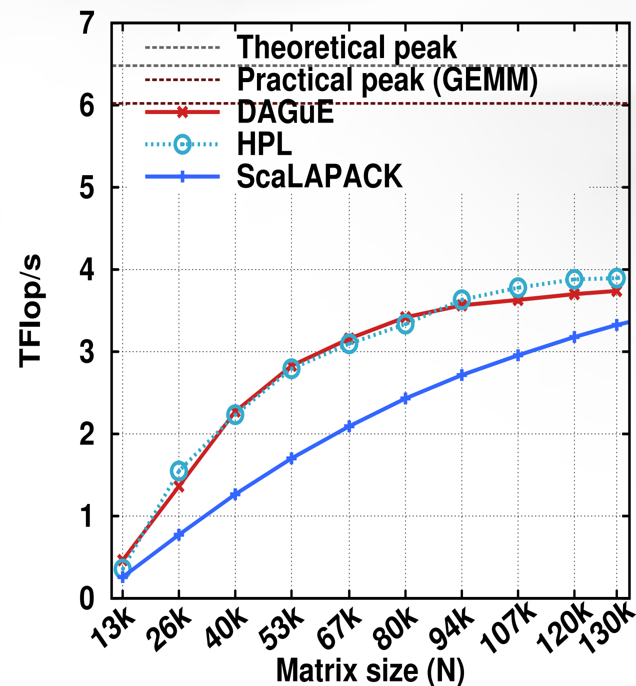
Block Size (NB)

- 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





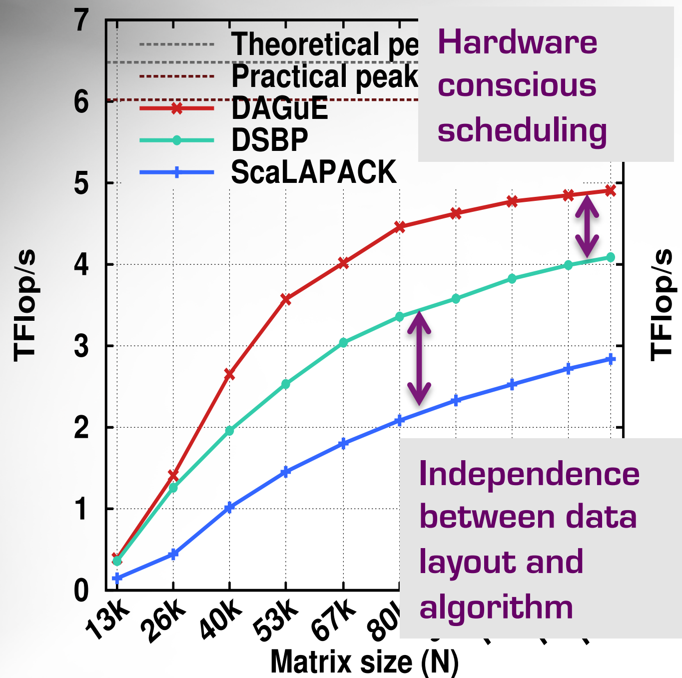


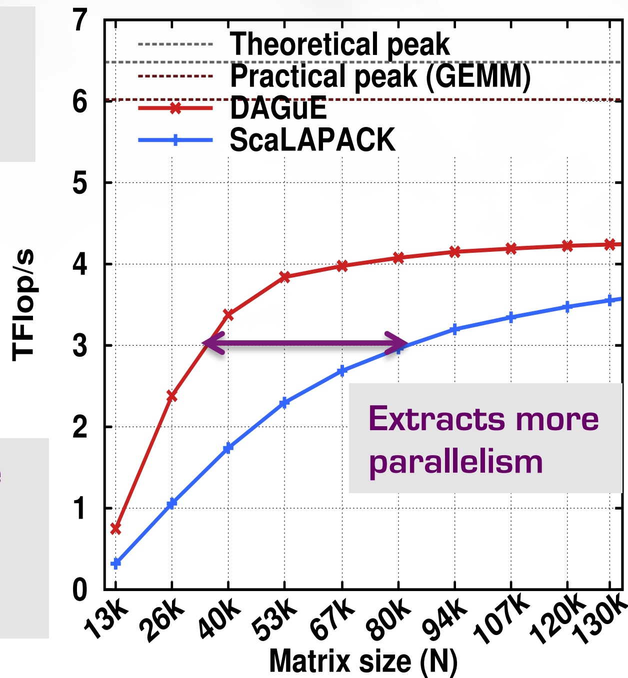


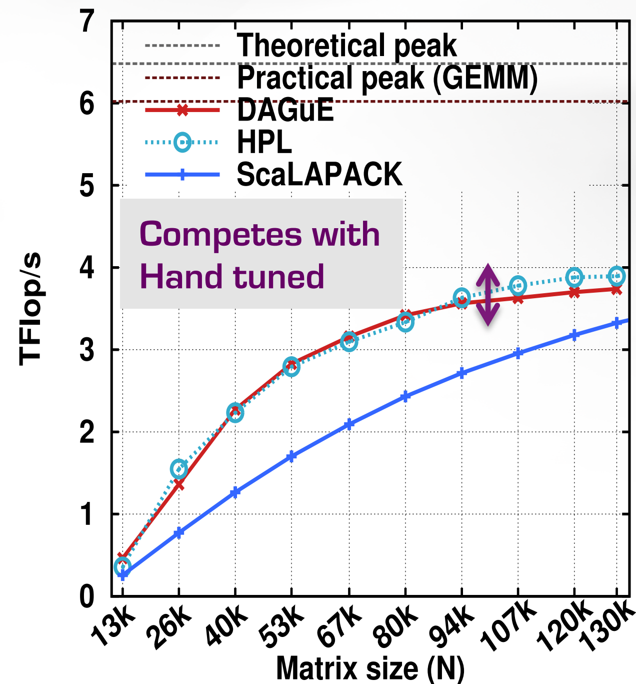
81 dual Intel Xeon L5420@2.5GHz
(2x4 cores/node) → 648 cores
MX 10Gbs, Intel MKL, Scalapack

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.





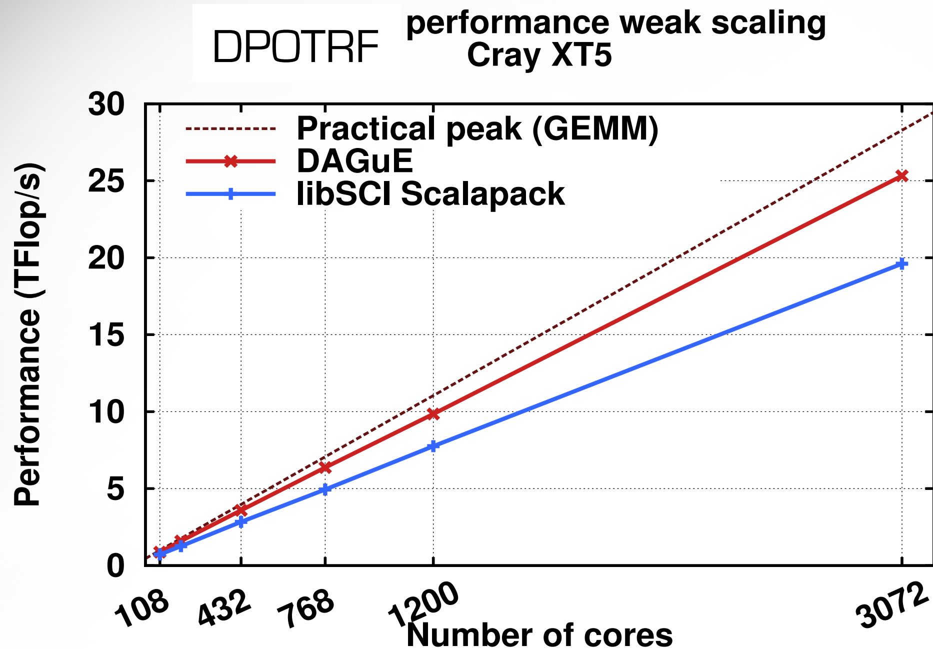


DSBP

[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

Scalability in Distributed Memory



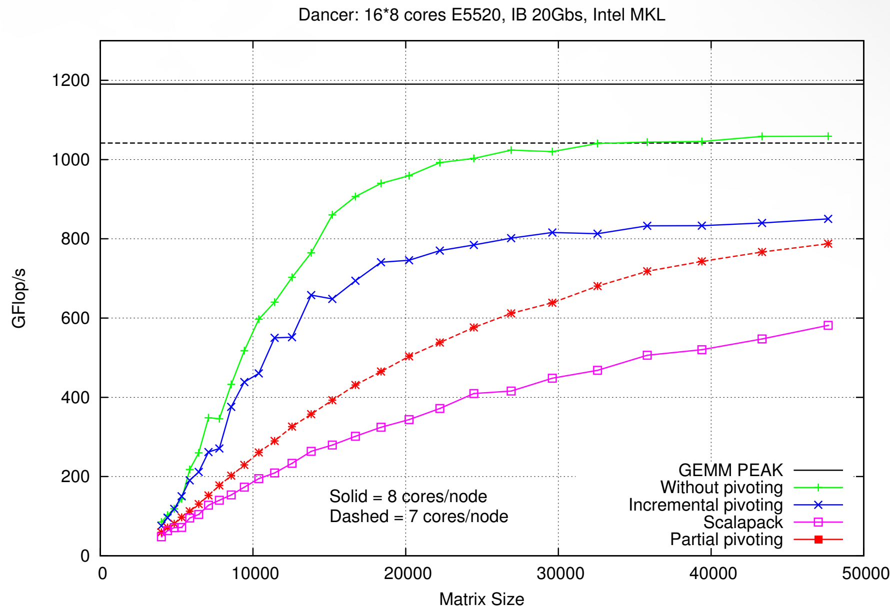
- Parameterized Task Graph representation
- Independent distributed scheduling

➡ Scalable

HPL or LU with Partial Pivoting

Inria Bordeaux, UTK

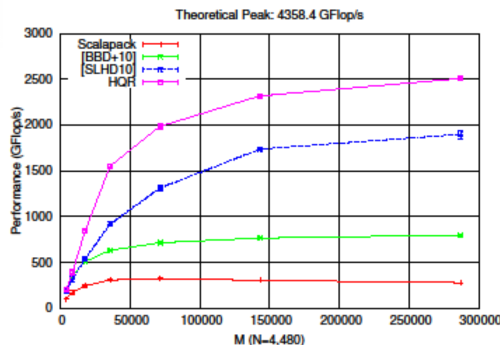
- Why LU is different? 2 reasons:
  - The panel is expressed using dataflow, generating many tiny tasks
    - Provides very poor performance
  - The DAG depends on the content of the data
    - The strict dataflow model we imposed on this exercise prevents message agregation
    - Our implementation is not message optimal
  - Explore all cases with as less tasks as possible

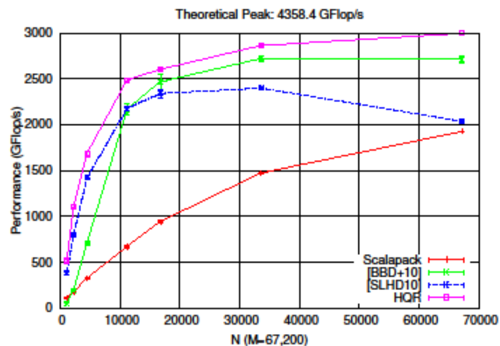


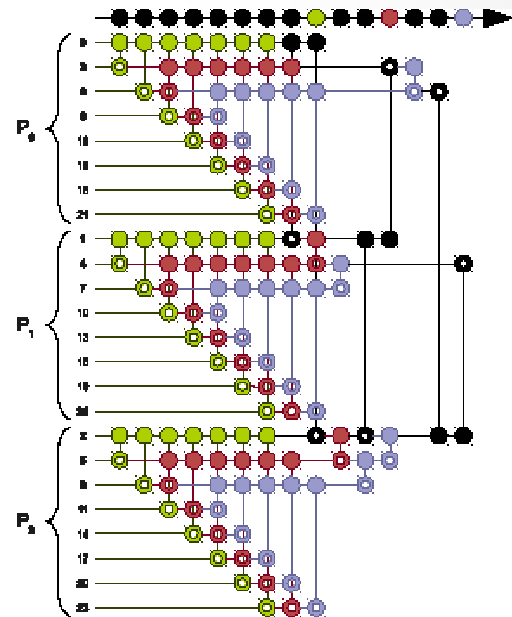
Hierarchical QR

ENS Lyon, Inria Bordeaux, UTK, UCD

- How to compose trees to get the best pipeline?
  - Flat, Binary, Fibonacci, Greedy, …
- Study on critical path lengths
- Surprisingly Flat trees are better for communications on square cases:
  - Less communications
  - Good pipeline



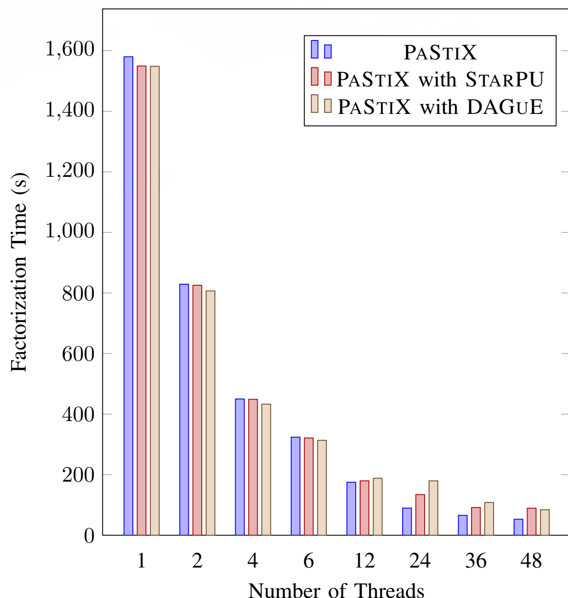


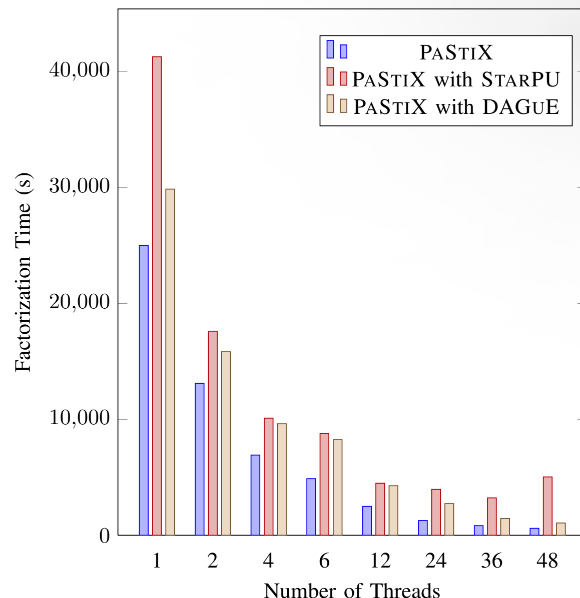


Sparse Direct Solvers

Inria Bordeaux, UTK

- Based on PaStiX solver
  - Super-nodal method as in SuperLU
  - Exploits an elimination tree => DAG
- LU, LL^t & LDL^t factorizations for shared memory with DAGuE/PaRSEC and StarPU
- GPU panel and update kernels
  - Based on blocked representation
- Problems:
  - Around 40 times more tasks than an equivalent dense factorization
  - Average task size can be very small (20x20)
  - Sizes are not regular





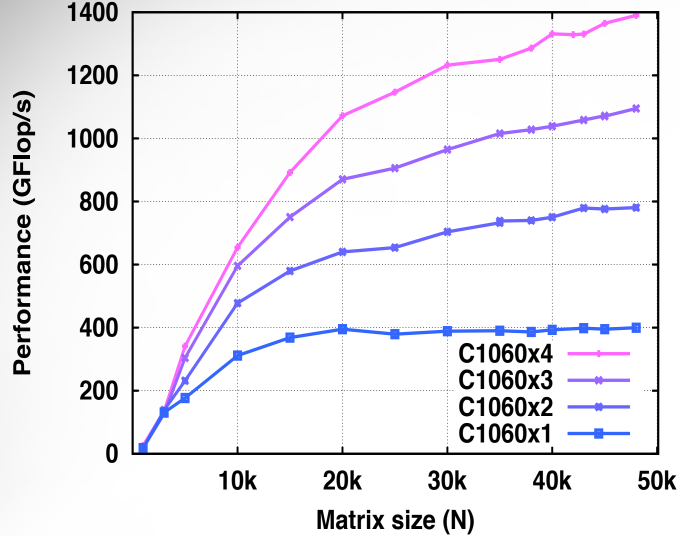
MHD	485,597	12,359,369	61.20	9.84e+12	Real	LU	University of Minnesota
Audi	943,695	39,297,771	31.28	5.23e+12	Real	LL^T	PARASOL Collection
10M	10,423737	89,072,871	75.66	1.72e+14	Complex	LDL^T	French CEA-Cesta

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, *]
```

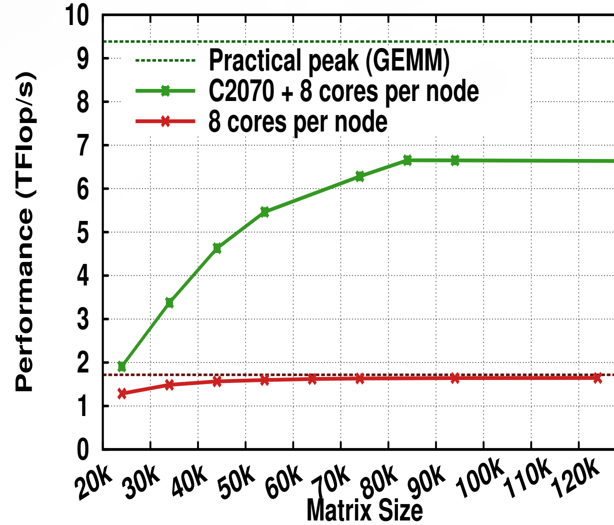
- A BODY is a task on a specific device (codelet)
- Currently the system supports CUDA and cores
- A CUDA device is considered as one additional memory level
- Data locality and data versioning define the transfers to and from the GPU/Co-processors

- Multi GPU – single node



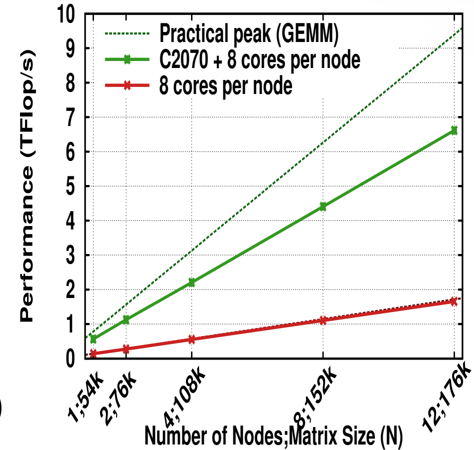
- Single node
- 4xTesla (C1060)
- 16 cores (AMD opteron)

- Multi GPU - distributed



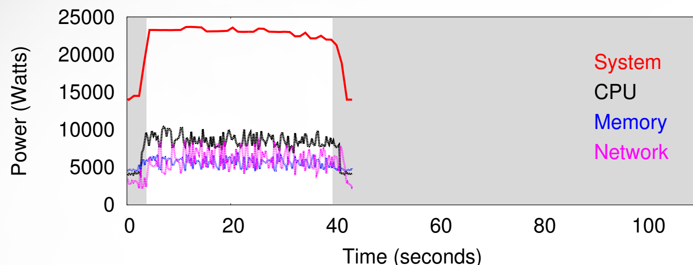
- 12 nodes
- 12xFermi (C2070)
- 8 cores/node (Intel core2)

Scalability

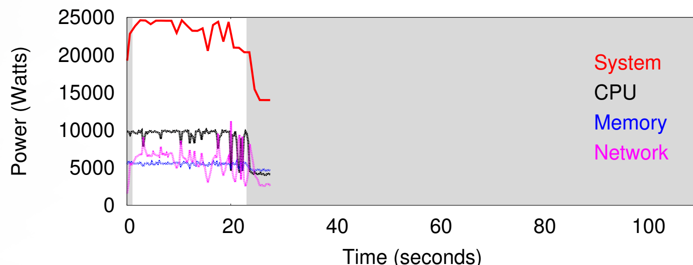


Energy efficiency

QR factorization (256 cores)



(a) ScaLAPACK.



(b) DPLASMA.

Total energy consumption

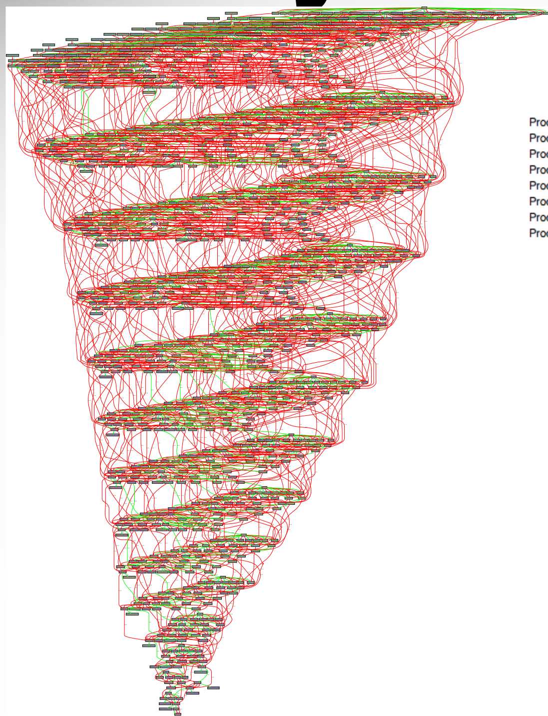
# Cores	Library	Cholesky	QR
128	ScaLAPACK	192000	672000
	DPLASMA	128000	540000
256	ScaLAPACK	240000	816000
	DPLASMA	96000	540000
512	ScaLAPACK	325000	1000000
	DPLASMA	125000	576000

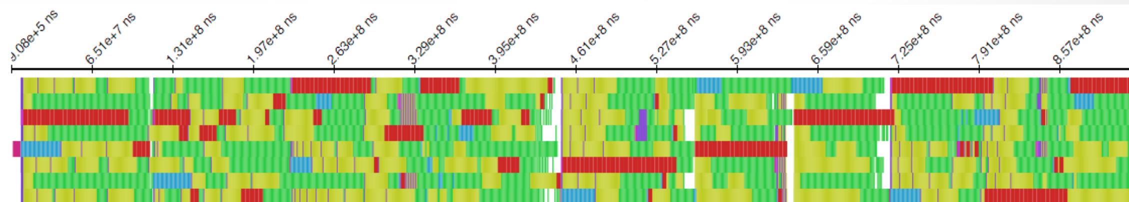
Work in progress with Hatem Ltaief

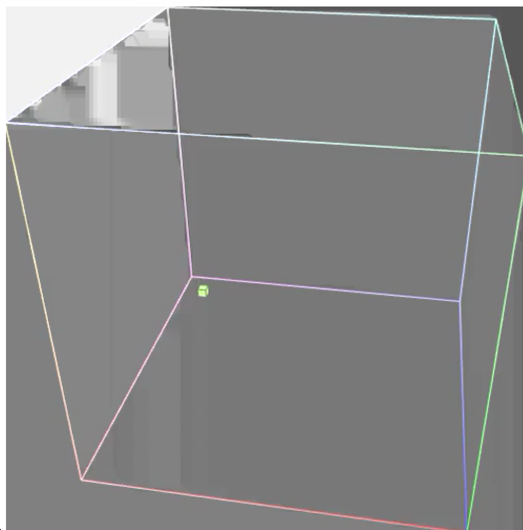
- Energy used depending on the number of cores
- Up to 62% more energy efficient while using a high performance tuned scheduling
  - Power efficient scheduling

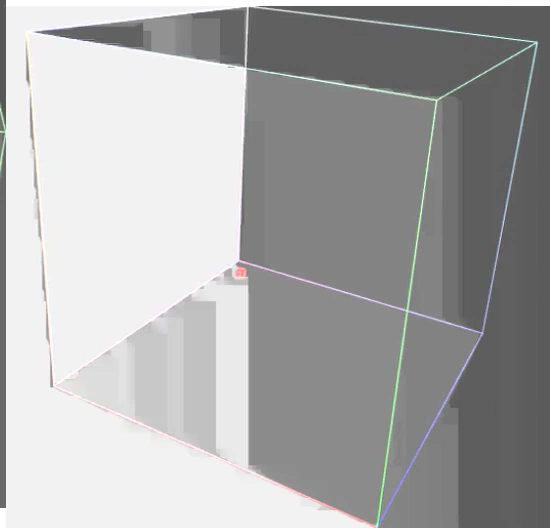
SystemG: Virginia Tech Energy Monitored cluster (ib40g, intel, 8cores/node)

Analysis Tools









Hermitian Band Diagonal; 16x16 tiles

Conclusion

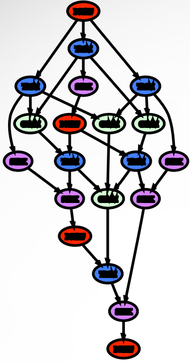
- Programming must be made easy(ier)
  - Portability: inherently take advantage of all hardware capabilities
  - Efficiency: deliver the best performance on several families of algorithms
- Computer scientists were spoiled by MPI
  - Now let’s think about our users
- Let different people focus on different problems
  - Application developers on their algorithms
  - System developers on system issues
  - Compilers on whatever they can

The end

Resilience

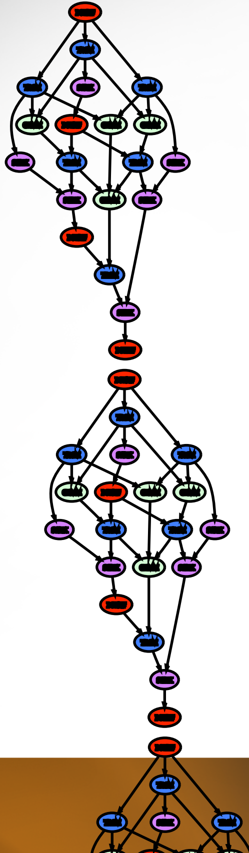
- The fault propagate in the system based on the data dependencies
- However, if the original data can be recovered, the execution complete without user interaction
- Automatic recovery made simple

Composition



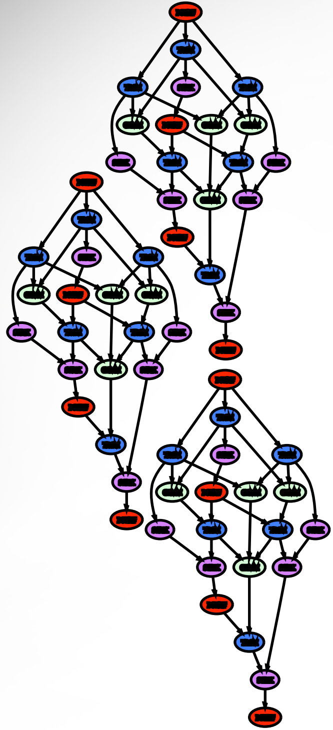
- An algorithm is a series of operations with data dependencies
- A sequential composition limit the parallelism due to strict synchronizations
  - Following the flow of data we can loosen the synchronizations and transform them in data dependencies

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

Other Systems

	PaRSEC	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