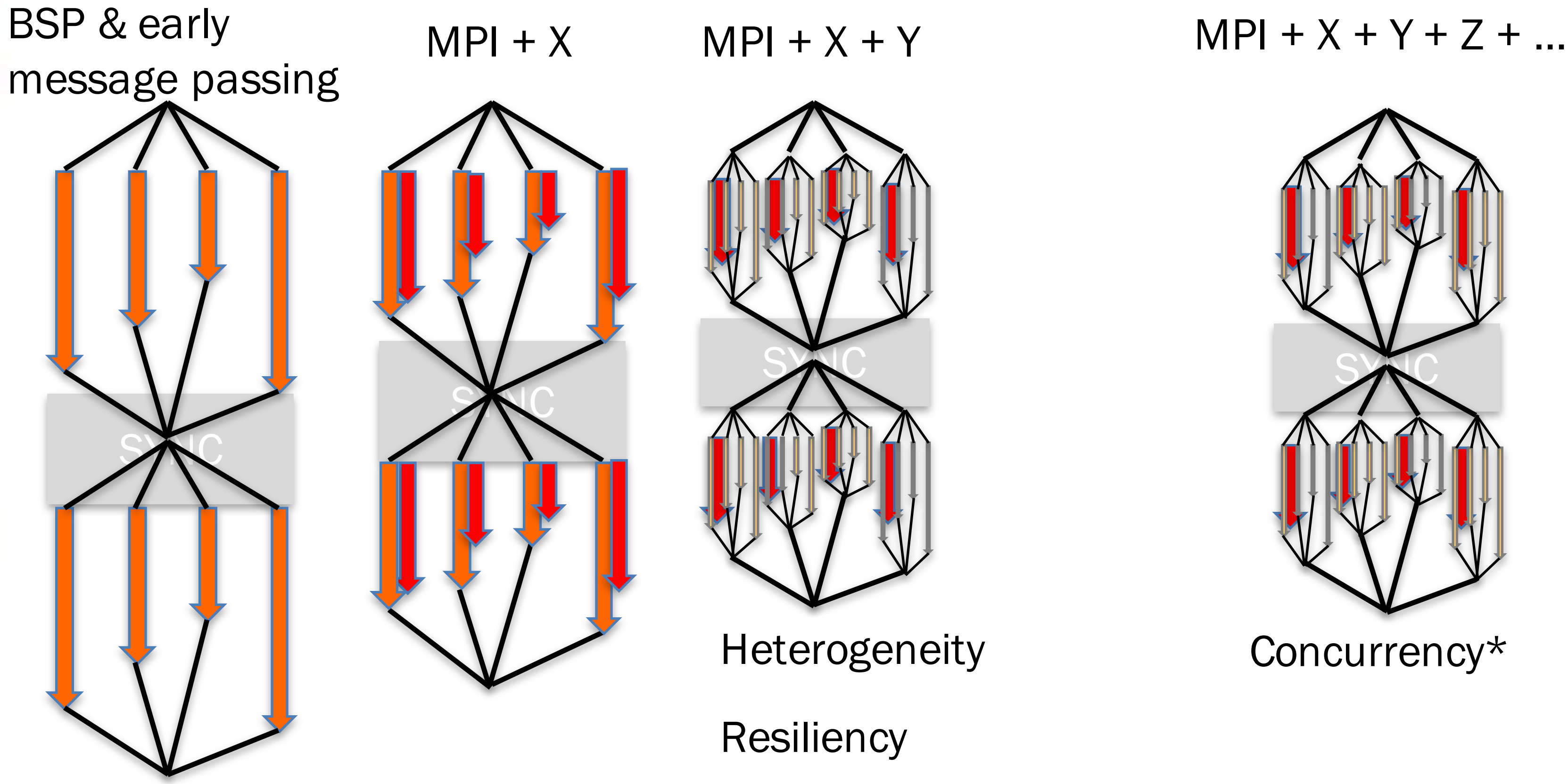
PaRSEC: Distributed task-based runtime for scalable applications

George Bosilca

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

A history of computing paradigms

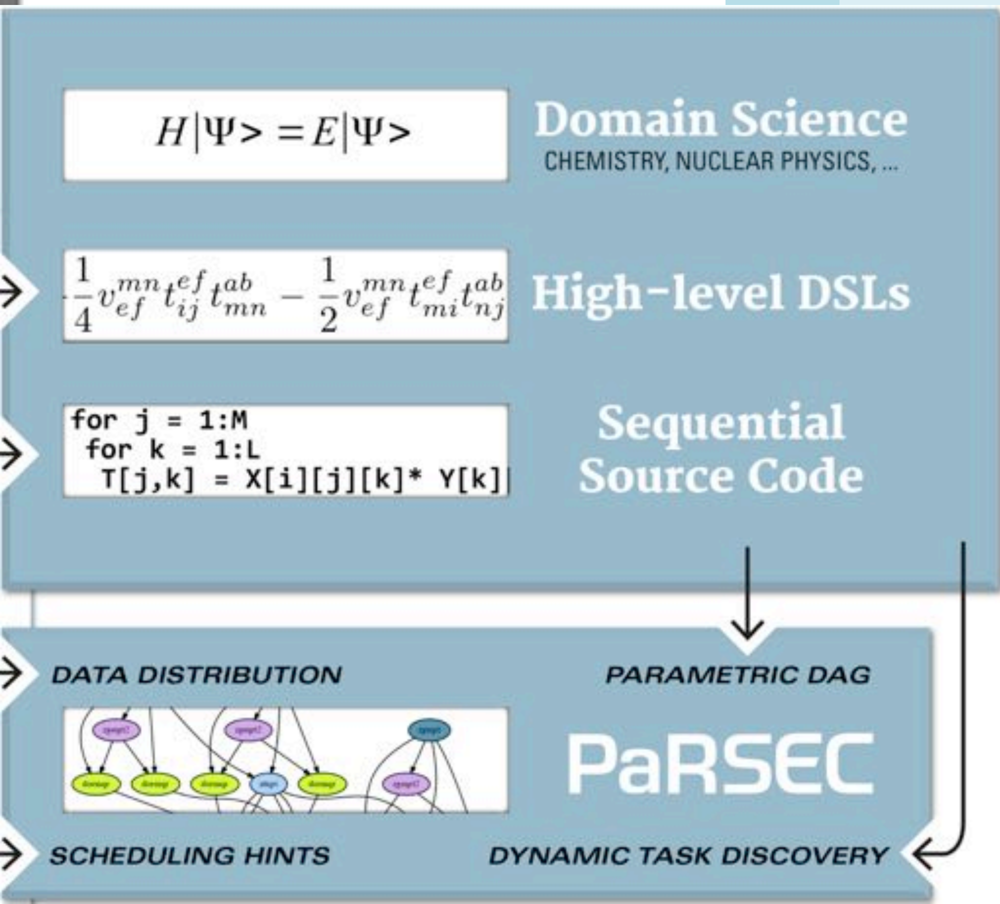


- Difficult to express the potential algorithmic parallelism
  - Control flow
  - Software became an amalgam of algorithm, data distribution and architecture characteristics
- Increasing gaps between the capabilities of today's programming environments, the requirements of emerging applications, and the challenges of future parallel architectures
- What is productivity ?

PaRSEC: a generic runtime system for asynchronous, architecture aware scheduling of fine-grained tasks on distributed many-core heterogeneous architectures

Concepts

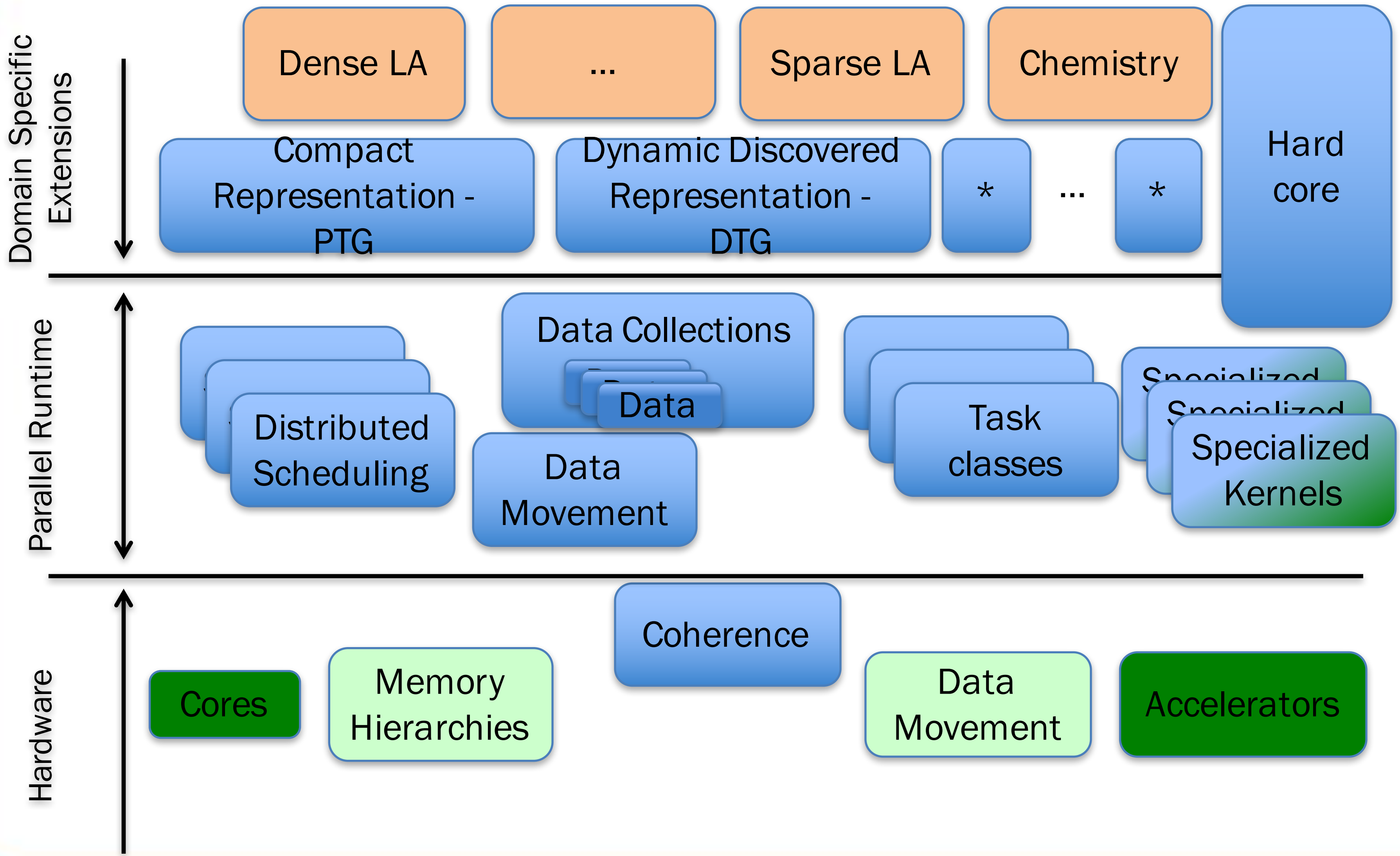
- Clear separation of concerns: compiler optimize each task class, developer describe dependencies between tasks, the runtime orchestrate the dynamic execution
- Interface with the application developers through specialized domain specific languages (PTG/JDF, Python, insert_task, fork/join, ...)
- Separate algorithms from data distribution
- Make control flow executions a relic



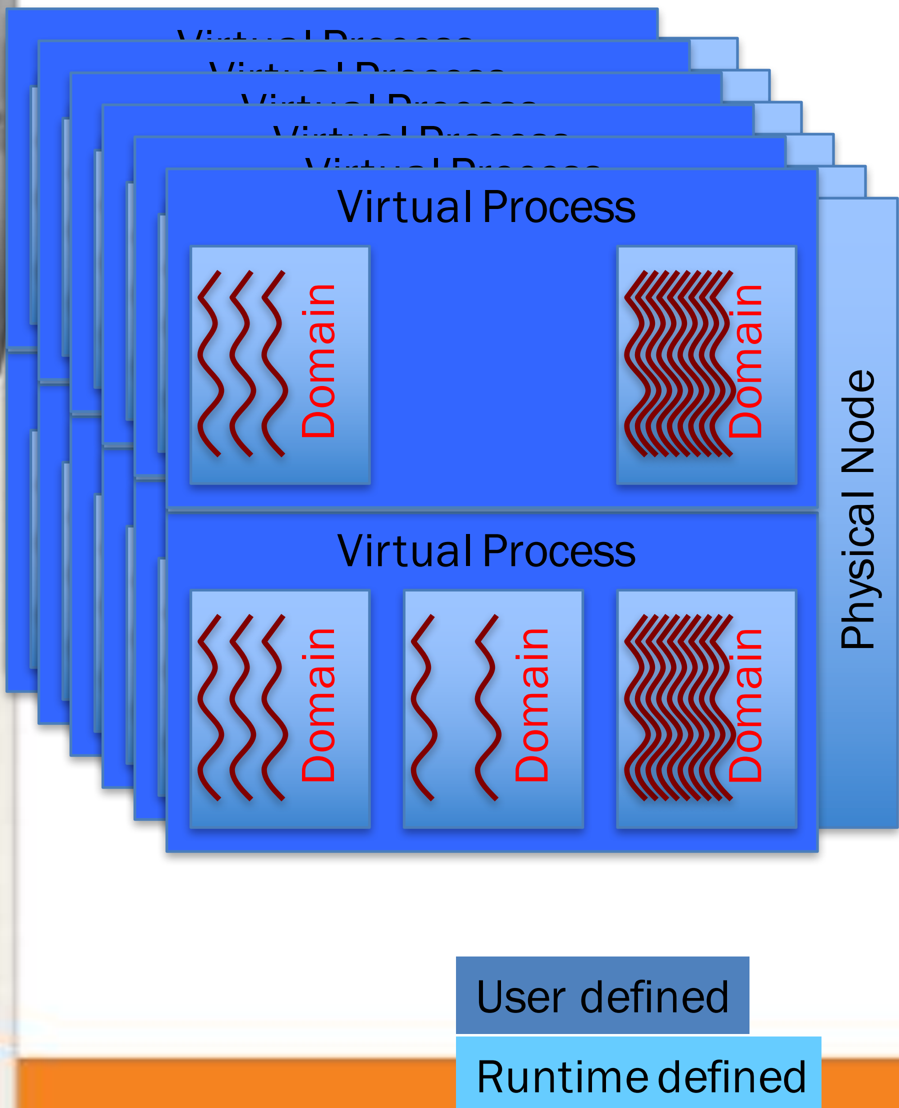
Runtime

- Permeable portability layer for heterogeneous architectures
- Scheduling policies adapt every execution to the hardware & ongoing system status
- Data movements between producers and consumers are inferred from dependencies. Communications/computations overlap naturally unfold
- Coherency protocols minimize data movements
- Memory hierarchies (including NVRAM and disk) integral part of the scheduling decisions

The PaRSEC framework



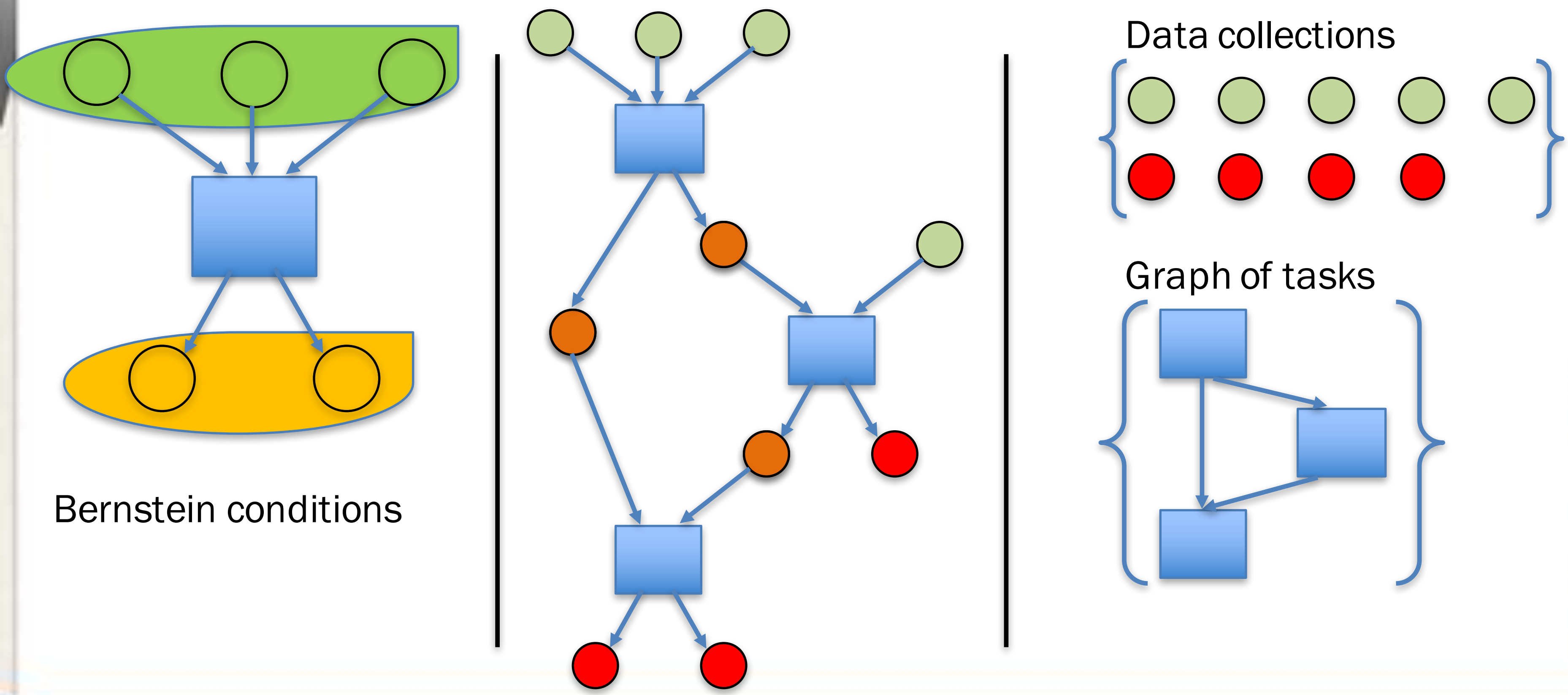
The PaRSEC machine model



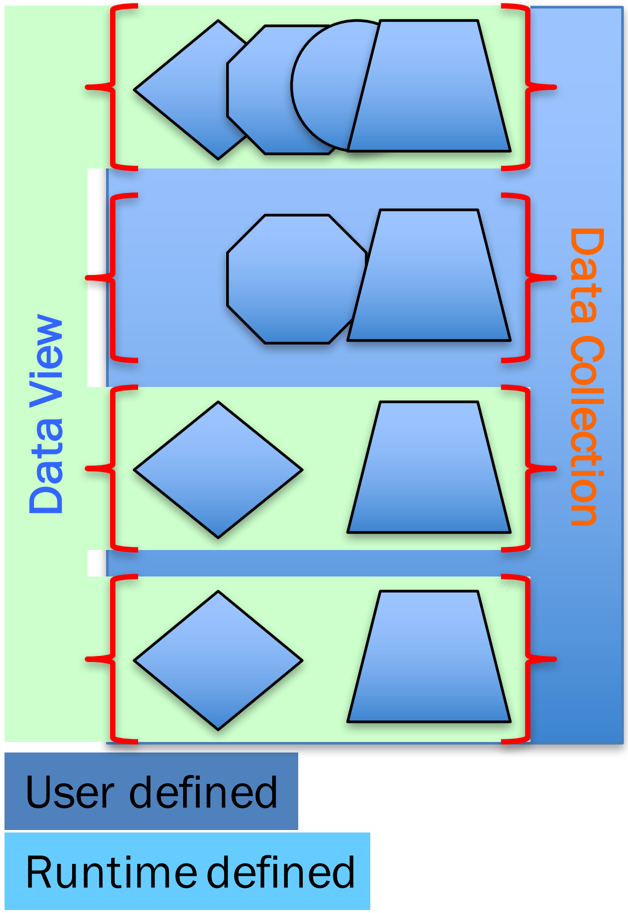
- Execution flow execute tasks sequentially
  - Can be bound to physical cores or can oversubscribe a resource
- A domain is a collection of execution flows with particular hardware properties: memory locality, similar computing capabilities, …
- A Virtual Process is a localization domain defining the scope of automatic migration/delocalization
  - Multiple VP can coexist on the same physical node
- Replicate over the total number of nodes

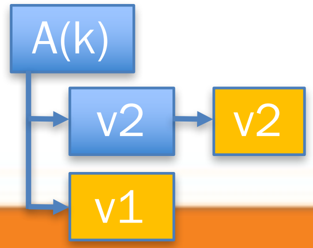
What is a task?

- An execution unit taking a set of input data and generating, upon completion, a different set of output data



The PaRSEC data



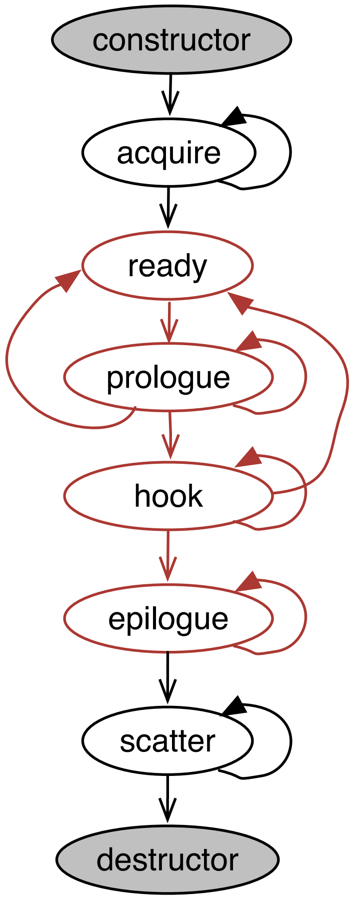


- A data is a manipulation token, the basic logical element used in the description of the dataflow
  - Location: have multiple coherent copies (remote node, device, checkpoint)
  - Shape: can have different memory layout
  - Visibility: only accessible via the most current version of the data
  - State: can be migrated / logged
- Data collections are ensemble of data distributed among the nodes
  - Can be regular (multi-dimensional matrices)
  - Or irregular (sparse data, graphs)
  - Can be regularly distributed (cyclic-k) or randomly
- Data View a subset of the data collection used in a particular algorithm (aka. submatrix, row, column,…)

- A data-copy is the practical unit of data
  - Has a memory layout (think MPI datatype)
  - Has a property of locality (device, NUMA domain, node)
  - Has a version associated with
  - Multiple instances can coexist

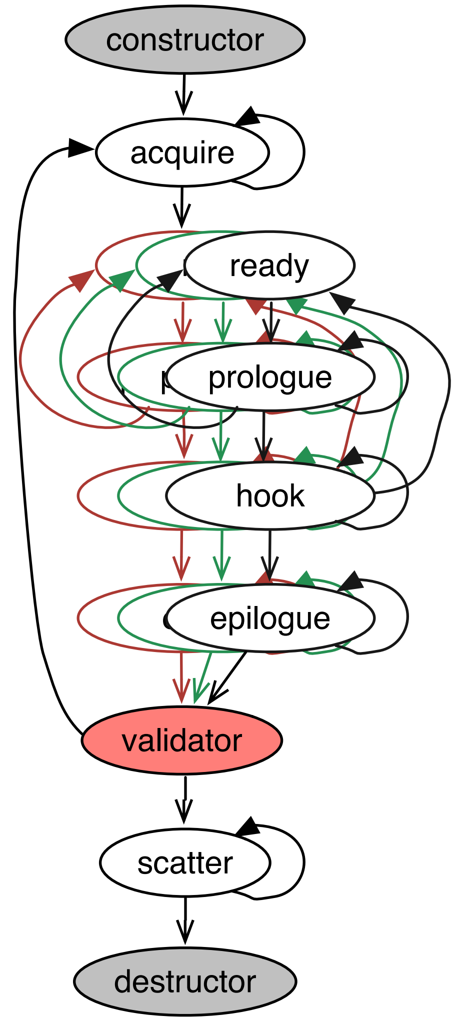
A PaRSEC task

- A task is a state machine
- The state machine is dynamic:
  - Can be altered by the runtime based on available resources
    - X and Y computing capability detected (CUDA, Xeon Phi, ...)
    - Resilient runtime
  - Or can be altered programmatically
- Changing states is based on the transition return code
  - Task delocalization to another (possibly external) execution domain
  - Task resubmission or reinitialization
  - Atomic tasks (and many more)

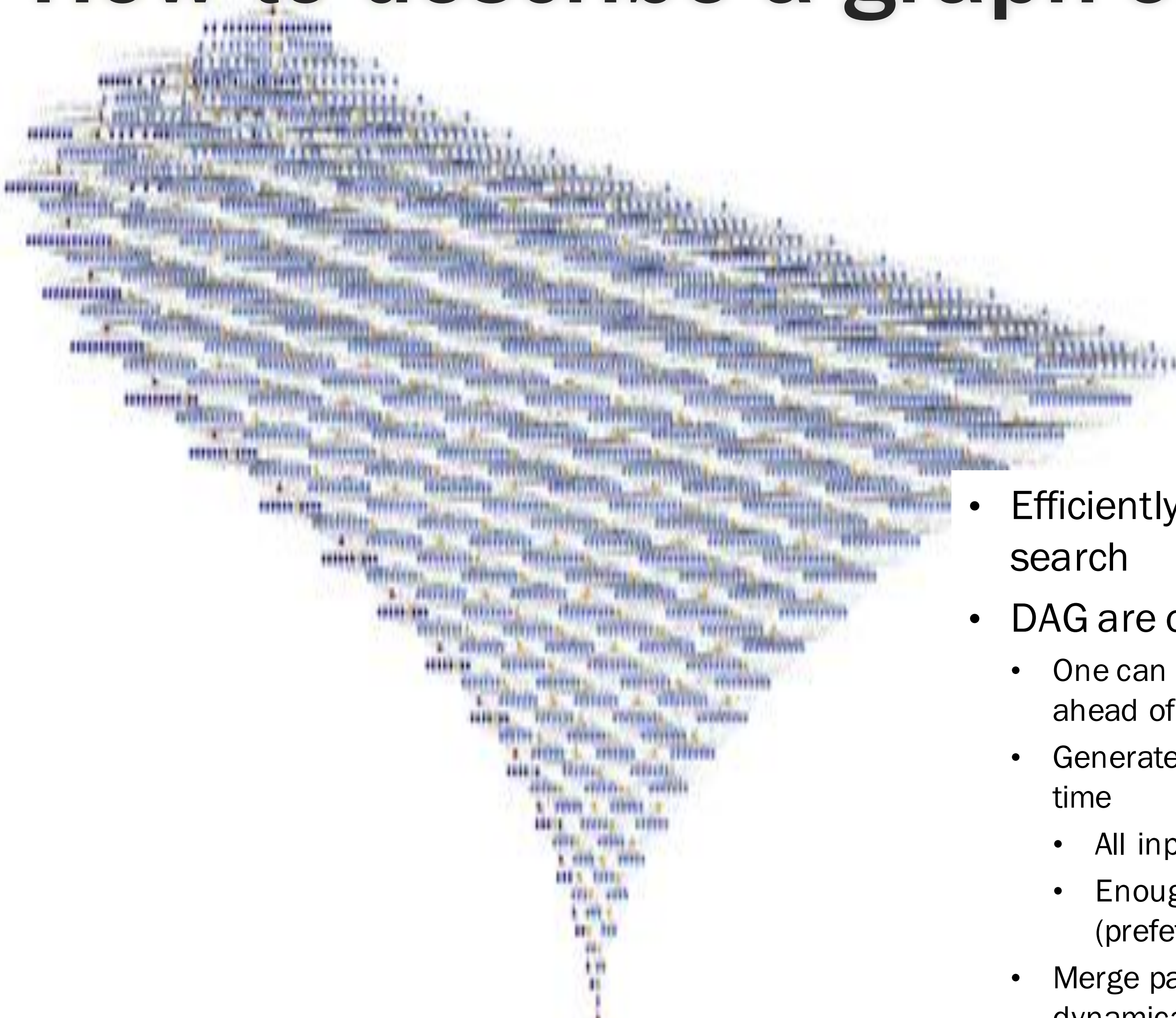


A PaRSEC task

- A task is a state machine
- The state machine is dynamic:
  - Can be altered by the runtime based on available resources
    - X and Y computing capability detected (CUDA, Xeon Phi, …)
    - Resilient runtime
  - Or can be altered programmatically
- Changing states is based on the transition return code
  - Task delocalization to another (possibly external) execution domain
  - Task resubmission or reinitialization
  - Atomic tasks (and many more)



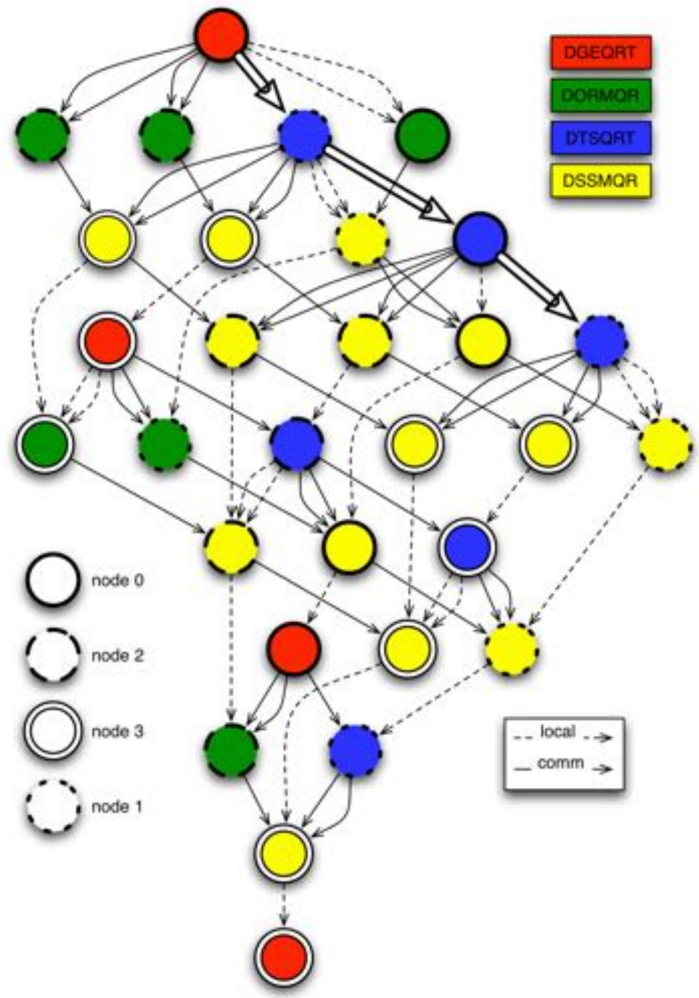
How to describe a graph of tasks ?



- Efficiently in terms of memory and search
- DAG are often large
  - One can hardly afford to generate them ahead of time
  - Generate it dynamically only when it is time
    - All input are available remotely
    - Enough inputs are available (prefetch)
  - Merge parameterized DAGs with dynamically generated DAGs

How to describe a graph of tasks ?

- Uncountable ways
  - Generic: Dagguer (Charm++), Legion, ParalleX, Parameterized Task Graph (PaRSEC), Dynamic Task Discovery (StarPU, StarSS), Yvette (XML), Fork/Join (spawn). CnC
  - Application specific: MADNESS
- PaRSEC runtime
  - The runtime is agnostic to the domain specific language (DSL)
  - Different DSL interoperate through the data collections
  - The DSL share
    - Distributed schedulers
    - Communication engine
    - Hardware resources
    - Data management (coherence, versioning, …)
  - They don't share
    - The task structure
    - The dataflow



The insert_task interface

Define a distributed collection of data (vector)

```
dague_vector_t dDATA;
dague_vector_init( &dDATA, matrix_Integer, matrix_Tile,
                   nodes, rank,
                   1, /* tile_size*/
                   N, /* Global vector size*/
                   0, /* starting point */
                   1 ); /* block size */
```

Start PaRSEC

```
dague_context_t* dague;
dague = dague_context_init(NULL, NULL); /* start the PaRSEC engine */
```

Create a tasks placeholder and associate it with the PaRSEC context

```
dague_dtd_handle_t* DAGUE_dtd_handle = dague_dtd_handle_new (dague);
dague_enqueue(dague, (dague_handle_t*) DAGUE_dtd_handle);
```

Keep adding tasks. A configurable window will limit the number of pending tasks

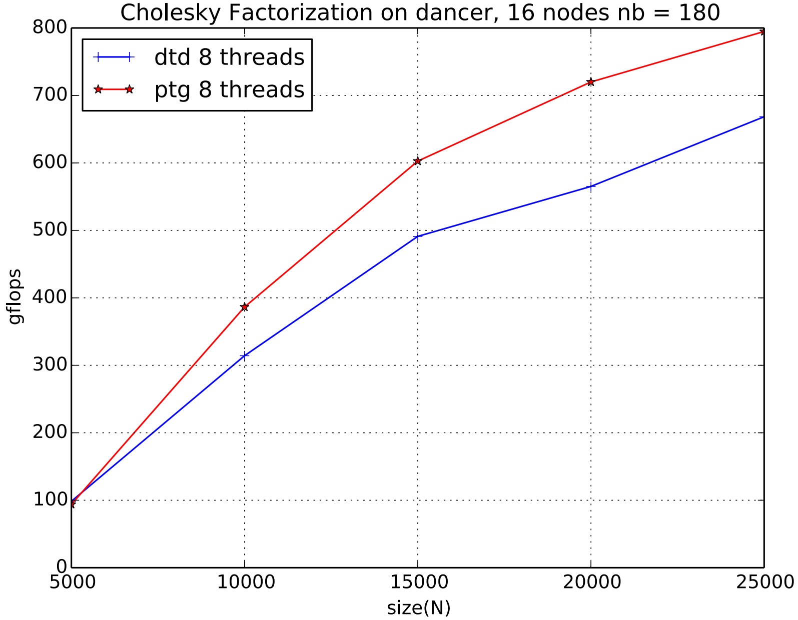
```
for( n = 0; n < N; n++ ) {
    dague_insert_task(
       DAGUE_dtd_handle,
       call_to_kernel_type_write,    "Task Name",
       PASSED_BY_REF,   DATA_AT(&dDATA, n),   INOUT | REGION_FULL,
       0 /* DONE */);
  for( k = 0; k < K; k++ ) {
    dague_insert_task(
       DAGUE_dtd_handle,
       call_to_kernel_type_read,    "Read_Task",
       PASSED_BY_REF,   DATA_AT(&dDATA, n),   INPUT | REGION_FULL,
       0  /* DONE */ );
    }
}
```

Wait 'till completion

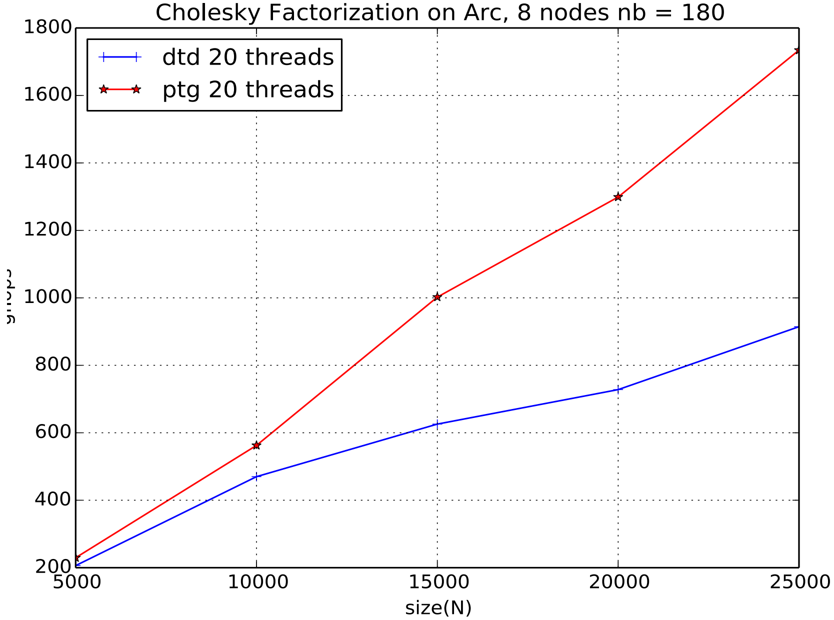
```
dague_handle_wait( DAGUE_dtd_handle );
```

The insert_task interface

16 nodes * 8 threads



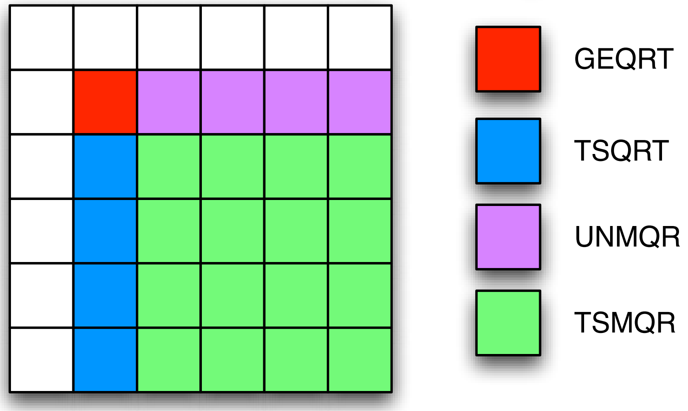
8 nodes * 20 threads



- Preliminary results
  - No collective pattern detection
  - No data cache

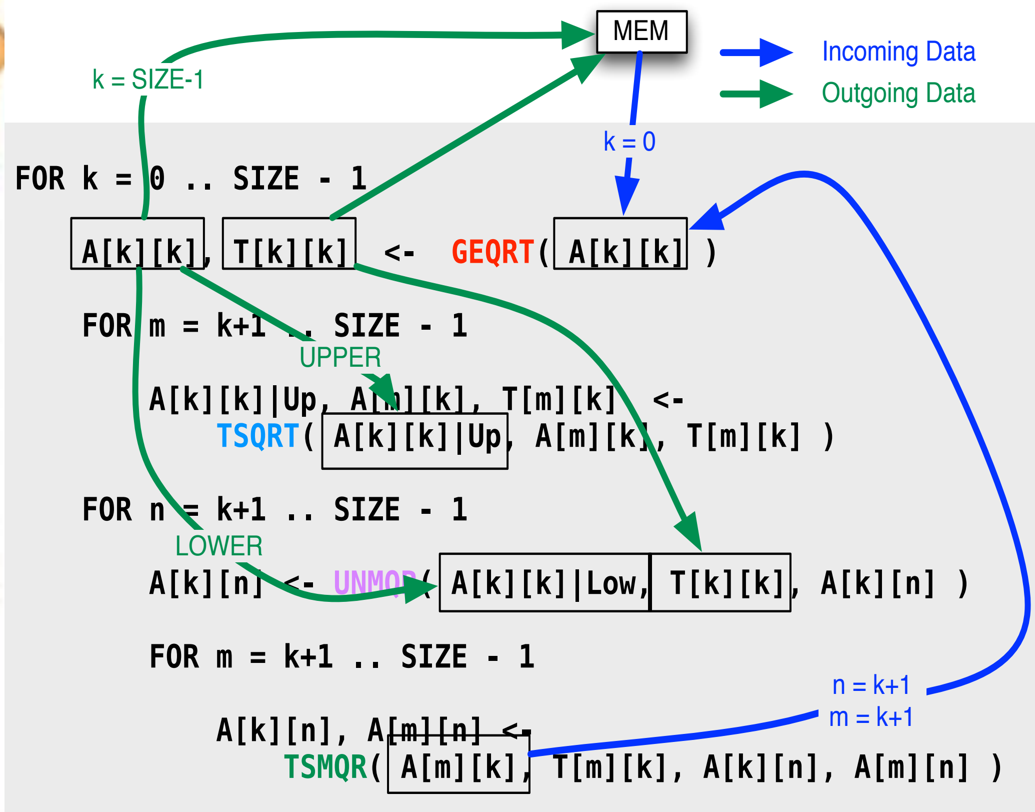
The Parameterized Task Graph (JDF)

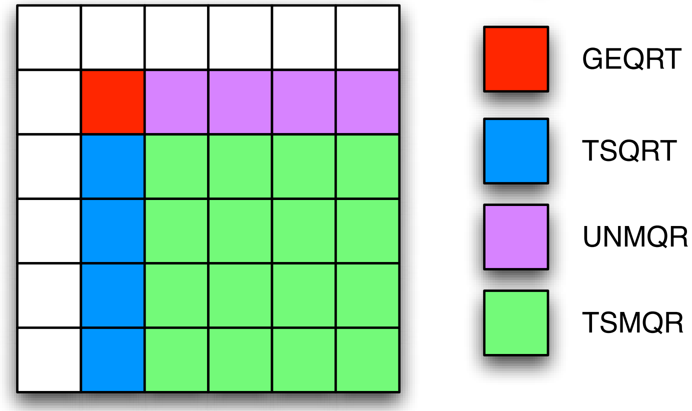
```
FOR k = 0 .. SIZE - 1

    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] )
```



- A dataflow description based on data tracking
- A simple affine description of the algorithm can be understood and translated by a compiler into a more complex, control-flow free, form
- Abide to all constraints imposed by current compiler technology

Task Graph (JDF)





- A dataflow description based on data tracking
- A simple affine description of the algorithm can be understood and translated by a compiler into a more complex, control-flow free, form
- Abide to all constraints imposed by current compiler technology

The Parameterized Task Graph (JDF)

```
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

- The resulting intermediary language is however more flexible
- Accept non-dense iterators
- Allow inlined C/C++ code to augment the language

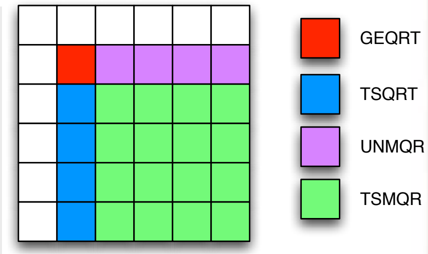
- JDF Drawbacks:
- Need to know the number of tasks
- The dependencies had to be globally (and statically) defined prior to the execution
  - No dynamic DAGs
  - No data dependent DAGs

DPLASMA = ScaLAPACK interface & PaRSEC capabilities

1. Original pseudo- or PLASMA code is converted by a preprocessor into PaRSEC internal representation (shown below)

2. Dataflow representation is assembled with the runtime to create a set of executable parameterized tasks (PT), which can execute the kernels, and unfold successors in the graph

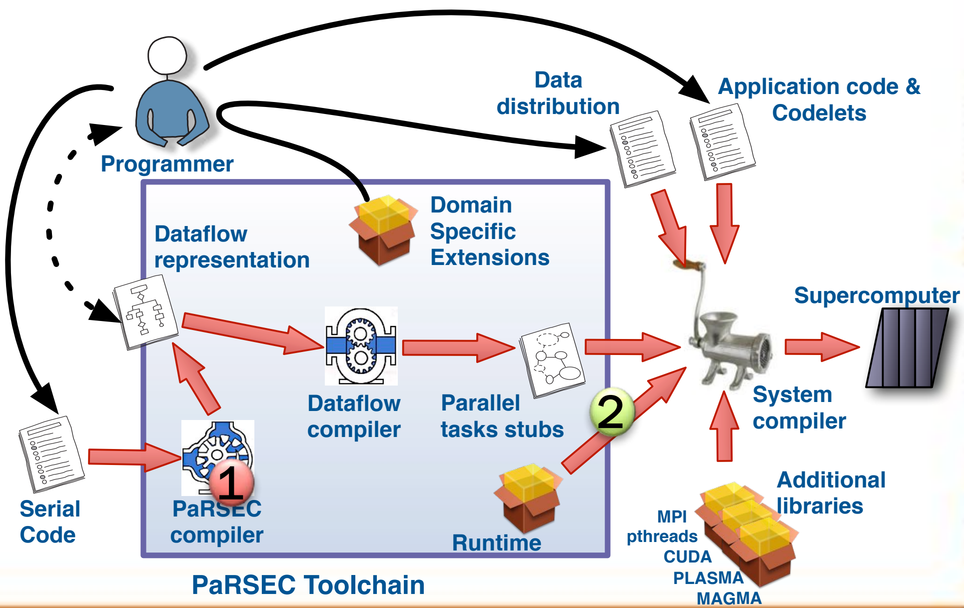
```
FOR k = 0 .. SIZE - 1
    A[k][k], T[k][k]  <-  GEQRT( A[k][k] )
    FOR m = k+1 .. SIZE - 1
        A[k][k]|Up, A[m][k], T[m][k]  <-
            TSQRT( A[k][k]|Up, A[m][k], T[m][k] )
    FOR n = k+1 .. SIZE - 1
        A[k][n] <- UNMQR( A[k][k]|Low, T[k][k], A[k][n] )
        FOR m = k+1 .. SIZE - 1
            A[k][n], A[m][n] <-
                TSMQR( A[m][k], T[m][k], A[k][n], A[m][n] )
```



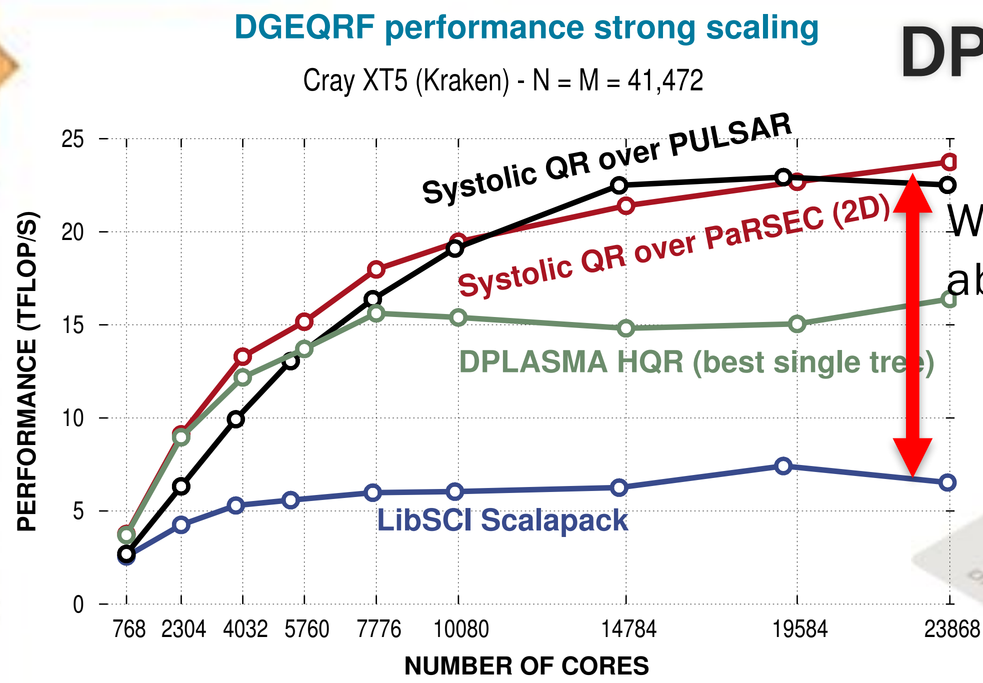
Tiled QR algorithm: how kernels are applied on the matrix during an iteration k

```
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)
```

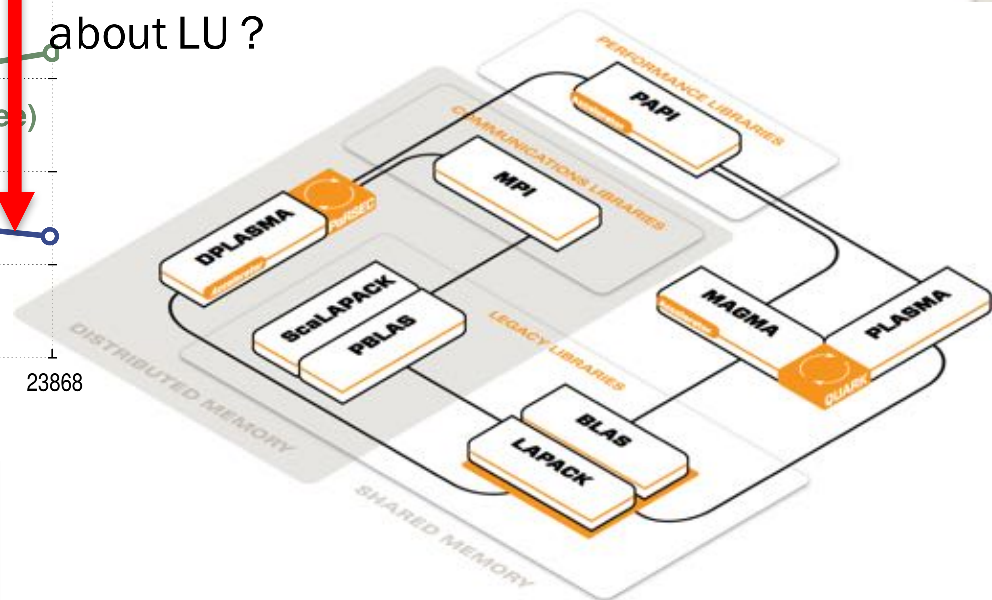
Intermediate dataflow representation

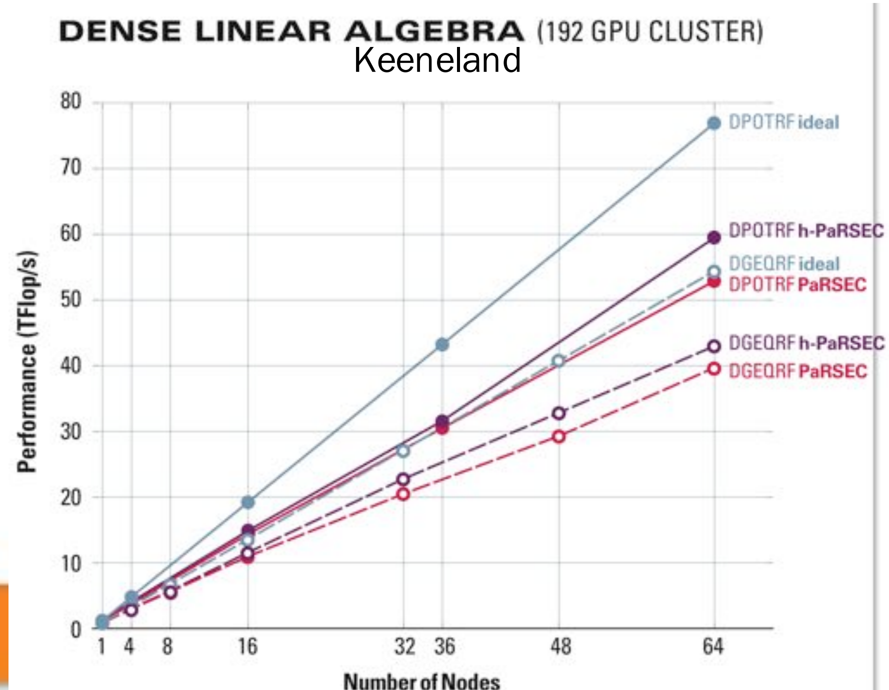


DPLASMA = ScaLAPACK + PaRSEC



What about LU ?

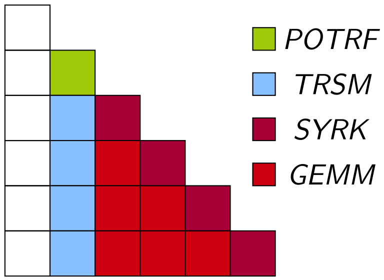




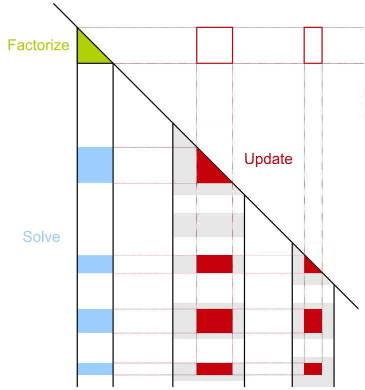
FUNCTIONALITY	COVERAGE
Linear Systems of Equations	Cholesky, LU (inc. pivoting, PP), LDL (prototype)
Least Squares	QR & LQ
Symmetric Eigenvalue Problem	Reduction to Band (prototype)
Level 3 Tile BLAS	GEMM, TRSM, TRMM, HEMM/SYMM, HERK/SYRK, HER2K/SYR2K
Auxiliary Subroutines	Matrix generation (PLRNT, PLGHE/PLGSY, PLTMG), Norm computation (LANGE, LANHE/LANSY, LANTR), Extra functions (LASET, LACPY, LASCAL, GEAD, TRADD, PRINT), Generic Map functions

Sparse support

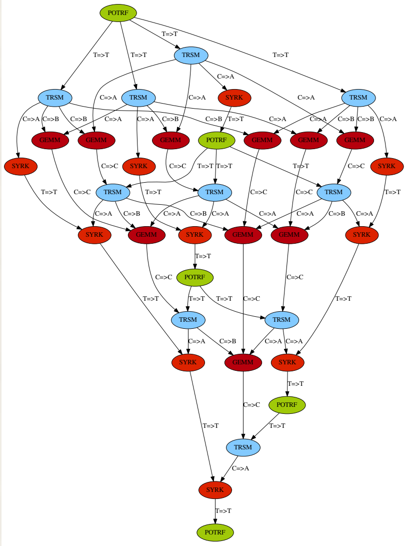
Total, Inria Bordeaux, Inria Pau, LaBRI, ICL



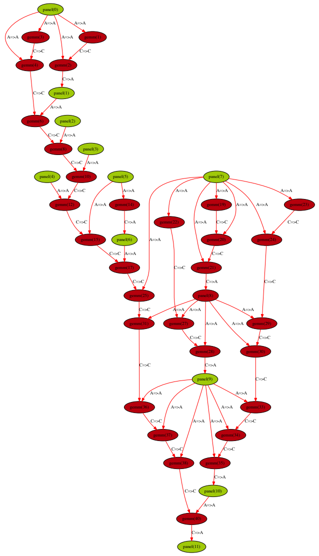
(a) Dense tile task decomposition



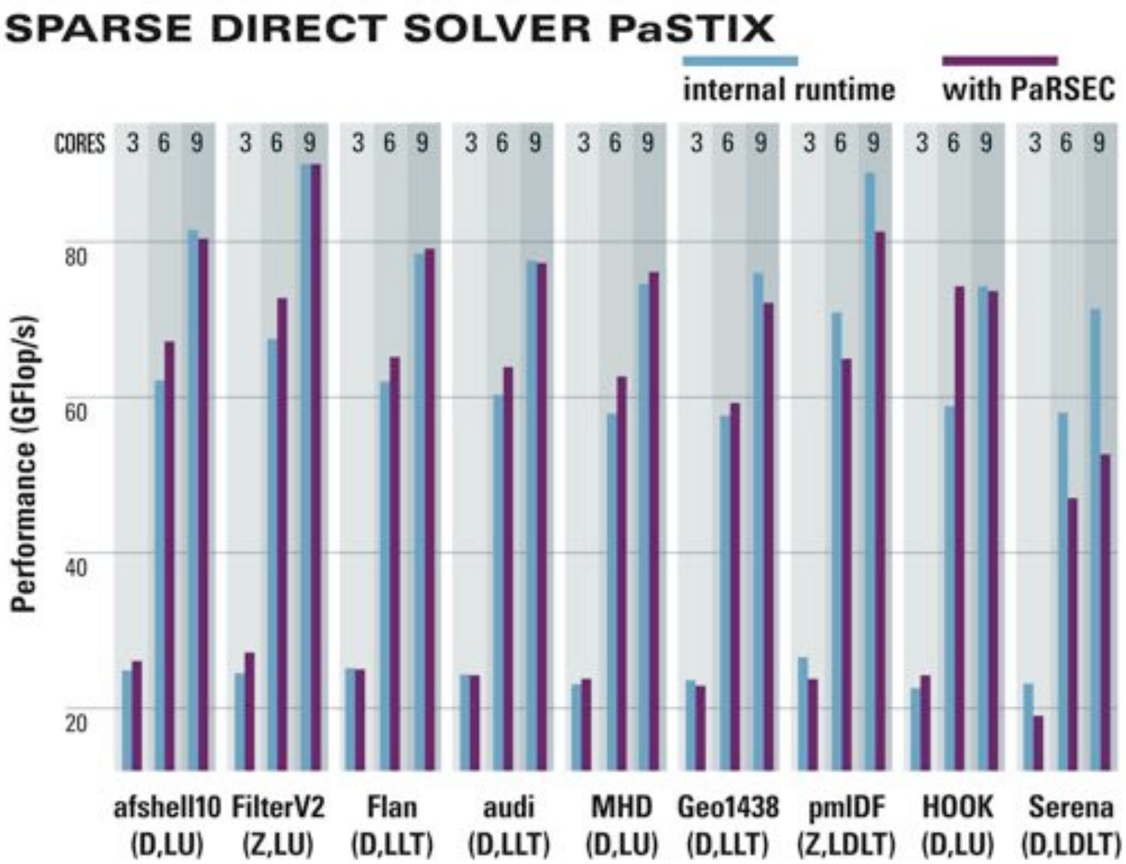
(b) Decomposition of the task applied while processing one panel



(c) Dense DAG



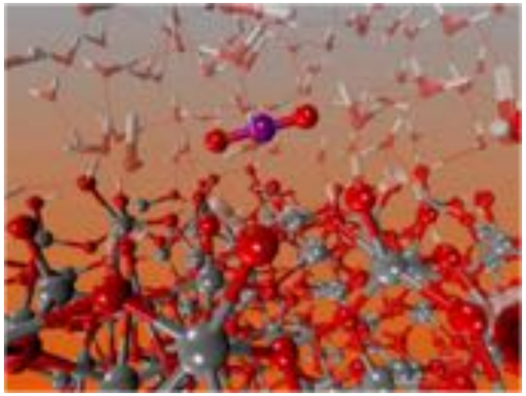
(d) Sparse DAG representation of a sparse LDL^T factorization



Other interactions with PaRSEC

NWCHEM 6.5

With Teresa Windus, Heike Jagode and Anthony Danalis



A Open Source High-Performance Computational Chemistry

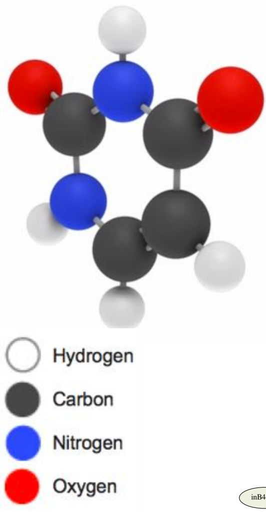
Conversion of NWChem CC code into dataflow form not trivial

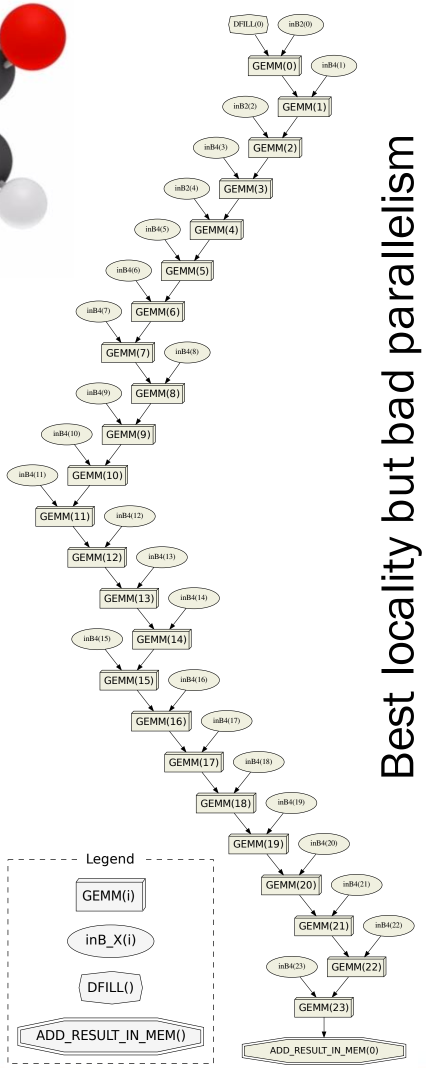
(CCSD code generated by TCE)

- Control flow is not affine nor statically decidable:
    - Loop execution space has holes,
    - dataflow goes through external routines,
    - conditional branches depend on program data,
    - memory access completely hidden in Global Arrays layer, etc.

→ None of the traditional Compiler Analysis tools can be used

Uracil-dimer





Best locality but bad parallelism

Integration of PaRSEC in CCSD

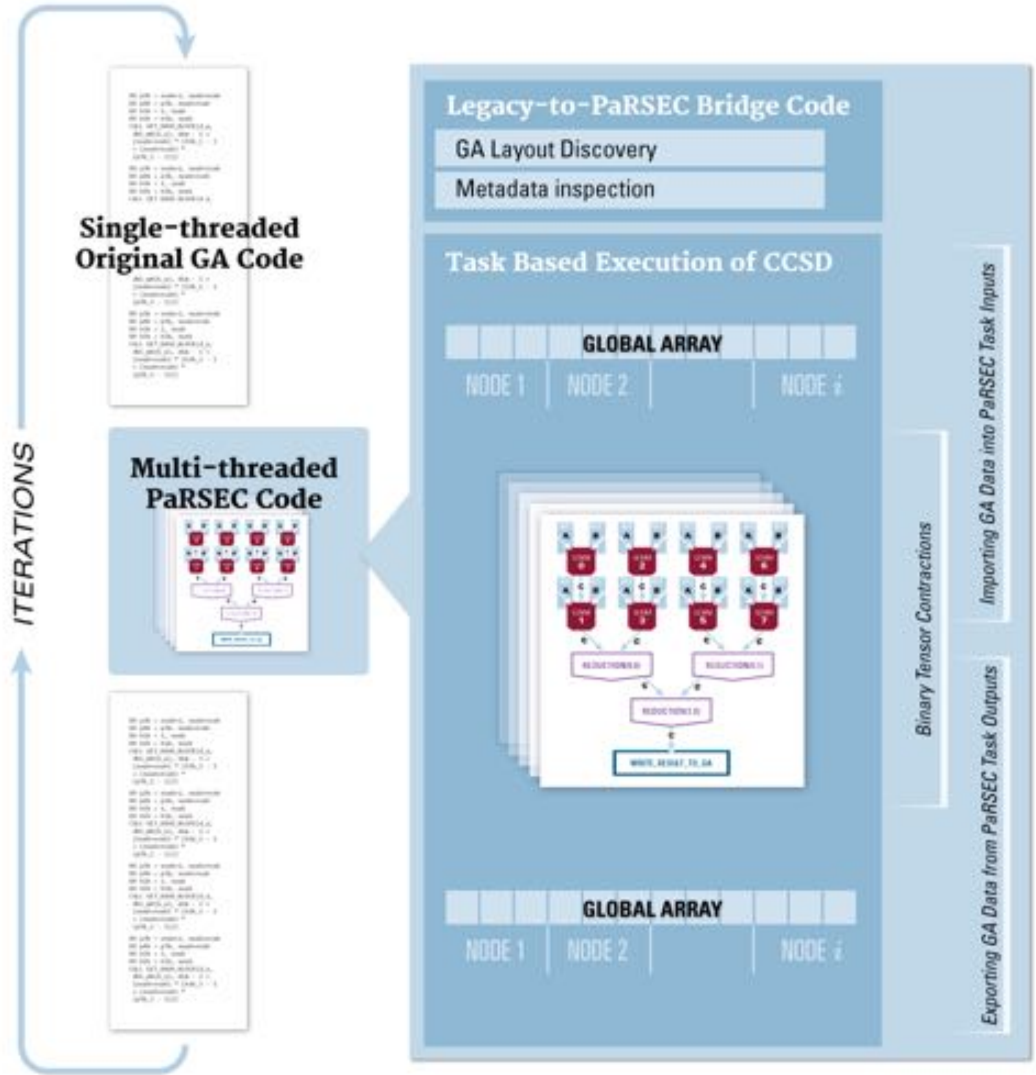
Elimination of synchronization points by describing data dependencies between matrix blocks

Finer grained (pure) tasks to allow for exploitation of more parallelism

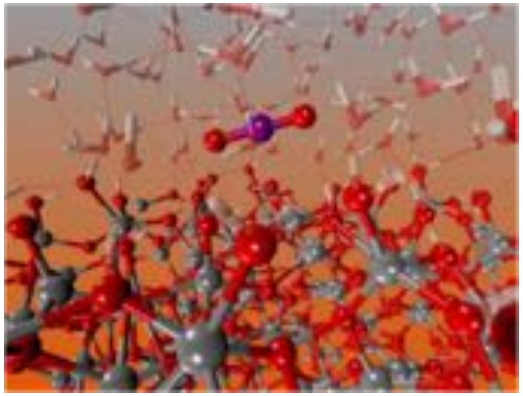
PARSEC-enabled version in 2 steps:

1. Traverse execution space and evaluate `IF` branches, without executing the actual computation (Since the data that affects the control flow is immutable at run-time, this step only needs to be performed once)

2. Create PTG – which includes lookups into our meta-data vectors populated by step 1.



NWCHEM 6.5



A Open Source High-Performance Computational Chemistry

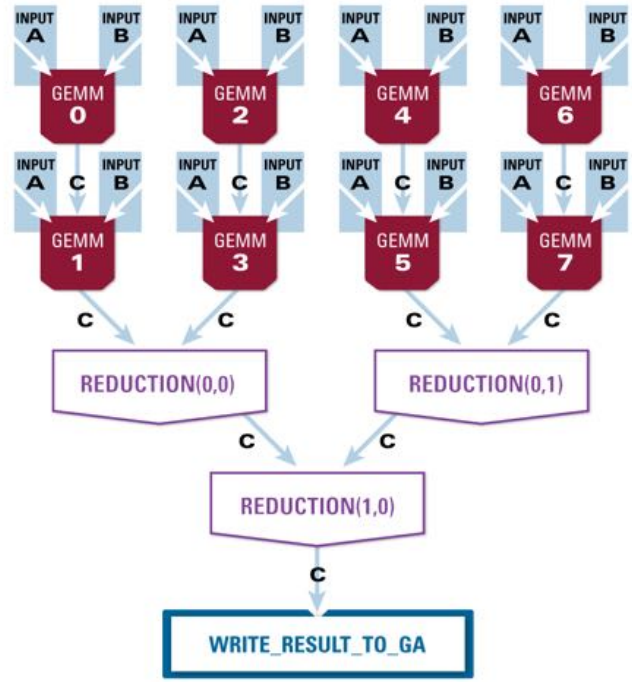
Conversion of NWChem CC code into dataflow form not trivial

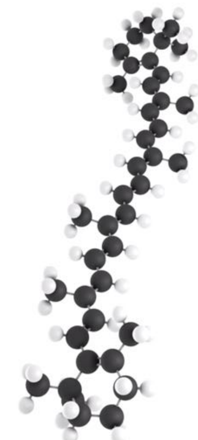
- Control flow is not affine nor statically decidable:
  - Loop execution space has holes,
  - dataflow goes through external routines,
  - conditional branches depend on program data,
  - memory access completely hidden in Global Arrays layer, etc.

→ None of the traditional Compiler Analysis tools can be used

Most significant outcomes of porting CC over PARSEC:

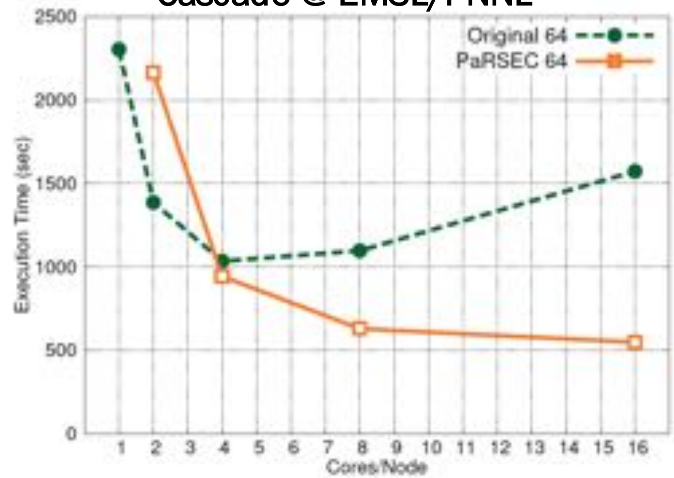
1. Ability of expressing tasks and their data dependencies at a finer granularity
2. Decoupling of computation and communication enable more advanced communication patterns than serial chains





$C_{40}H_{56}$

Cascade @ EMSL/PNNL

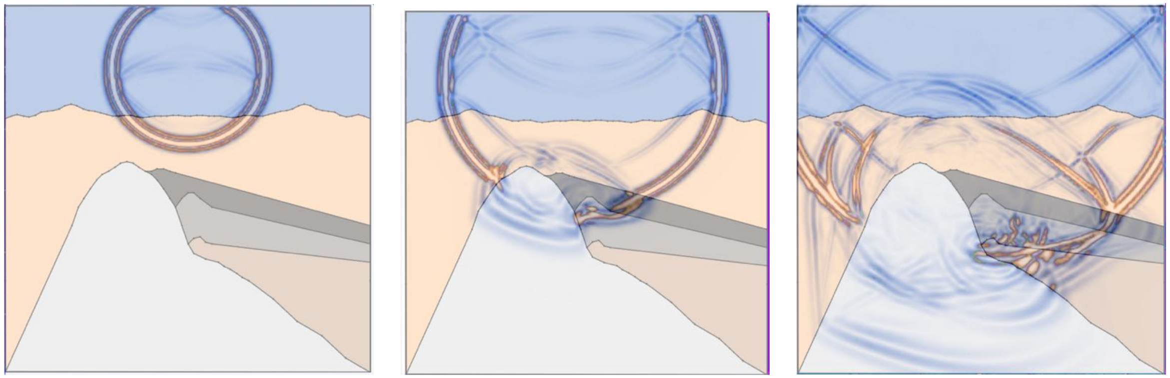


(b) Performance on 64 nodes

Unbounded parallelism

- The only requirement is that upon a task completion the potential descendants are known
- Uncountable DAGs
  - ” %option nb_local_tasks_fn = …”
  - Need user defined global termination
- Add support for dynamic DAGs
  - Already in the language
  - Properties of the algorithm / tasks
    - ”hash_fn = …”
    - ”find_deps_fn = …”

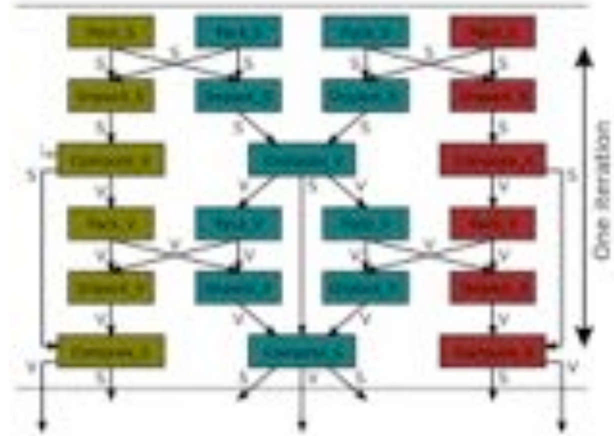
DIP: Elastodynamic Wave Propagation

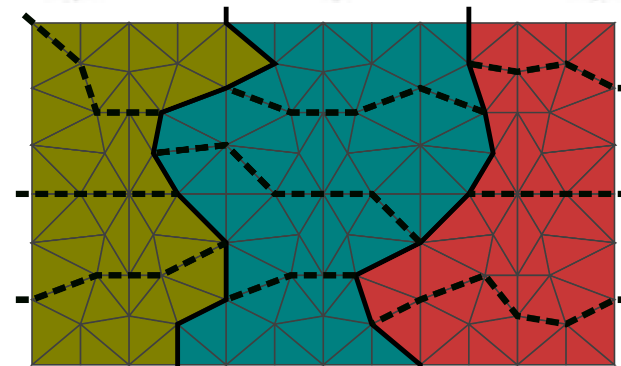


$$\begin{cases} v_h^{n+1} = v_h^n + M_v^{-1}[\Delta t R_{\underline{\sigma}} \underline{\sigma}_h^{n+1/2}] & \textit{UpdateVelocity} \\ \underline{\sigma}_h^{n+3/2} = \underline{\sigma}_h^{n+1/2} + M_{\underline{\sigma}}^{-1}[\Delta t R_v v_h^{n+1}] & \textit{UpdateStress} \end{cases}$$

For $n = 1 : n\_timesteps\_T$
  Communication($\sigma_h^{n+1/2}$)
  $v_h^{n+1} \leftarrow computeVelocity(v_h^n, \sigma_h^{n+1/2}, \Delta_t)$
  Communication(v_h^{n+1})
  $\sigma_h^{n+3/2} \leftarrow computeStress(\sigma_h^{n+1/2}, v_h^{n+1}, \Delta_t)$
End For t

Finer grain partitioning compared with MPI
Increased communications but also increased potential for parallelism
Need for load-balancing

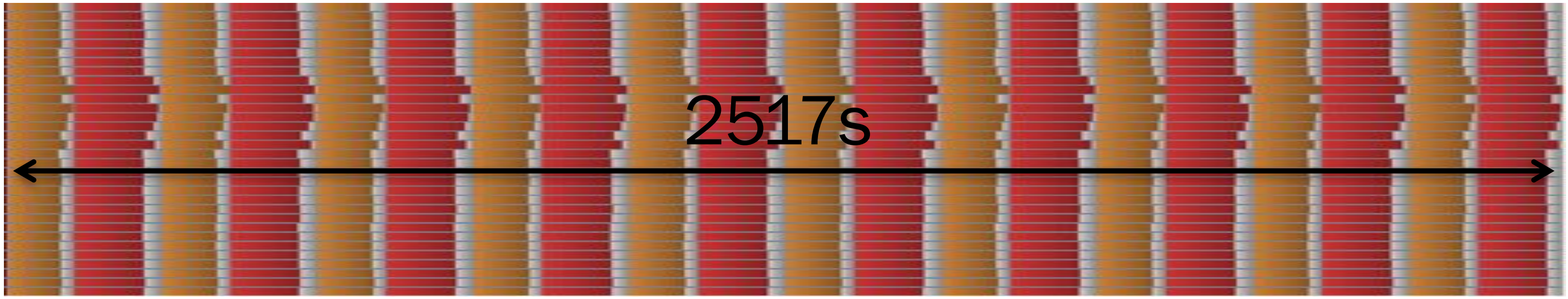


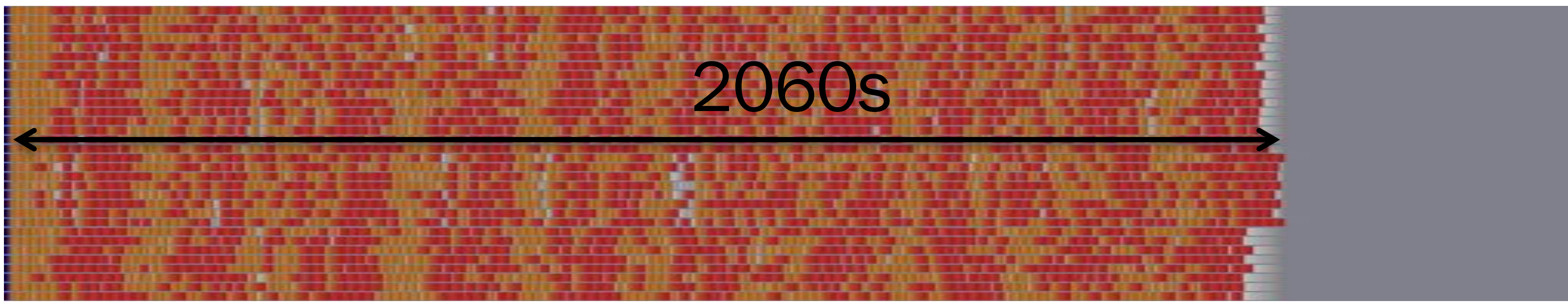


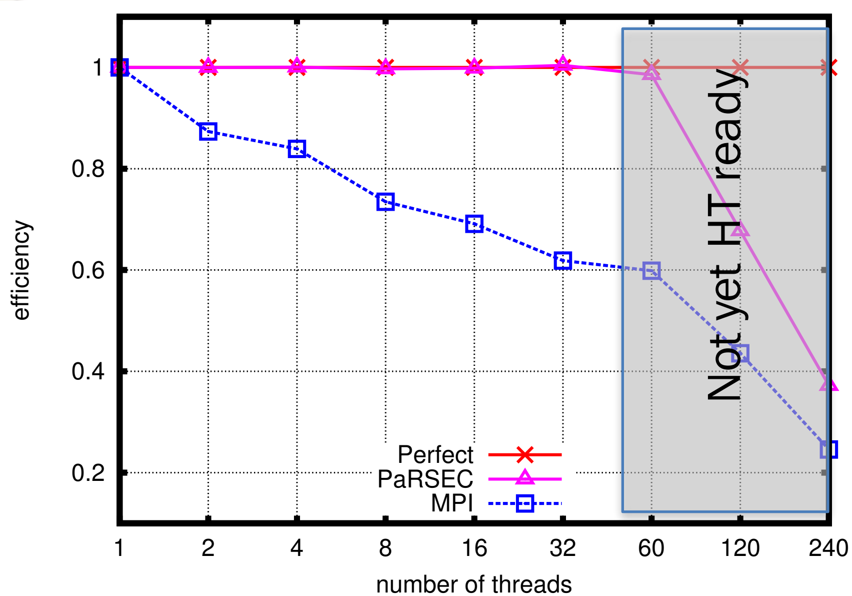
Dynamically redistribute the data
- use PAPI counters to estimate the imbalance
- reshuffle the frontiers to converge to a load balanced scenario

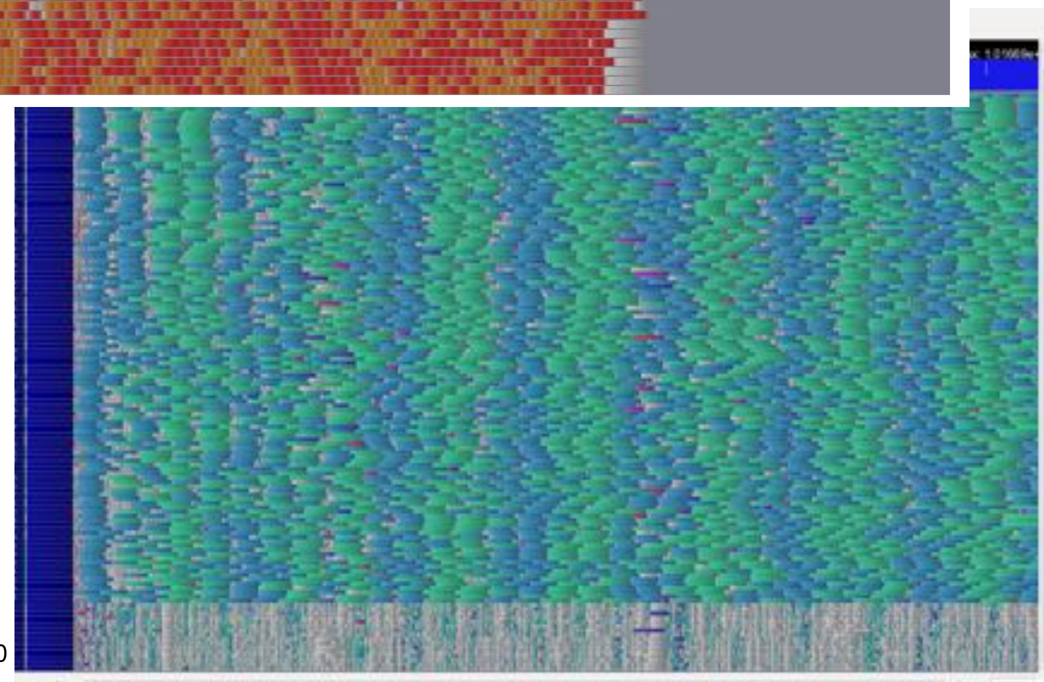
Total, Inria Bordeaux, Inria Pau, ICL

DIP: Elastodynamic Wave Propagation





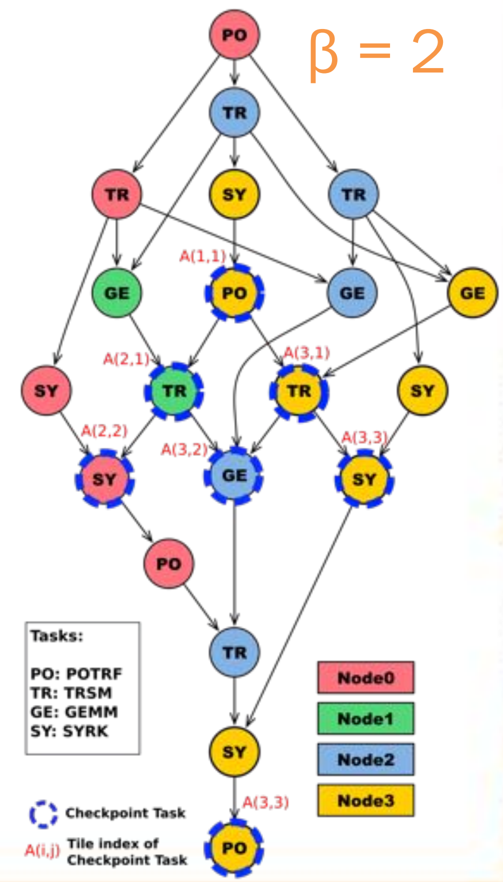




Total, Inria Bordeaux, Inria Pau, ICL

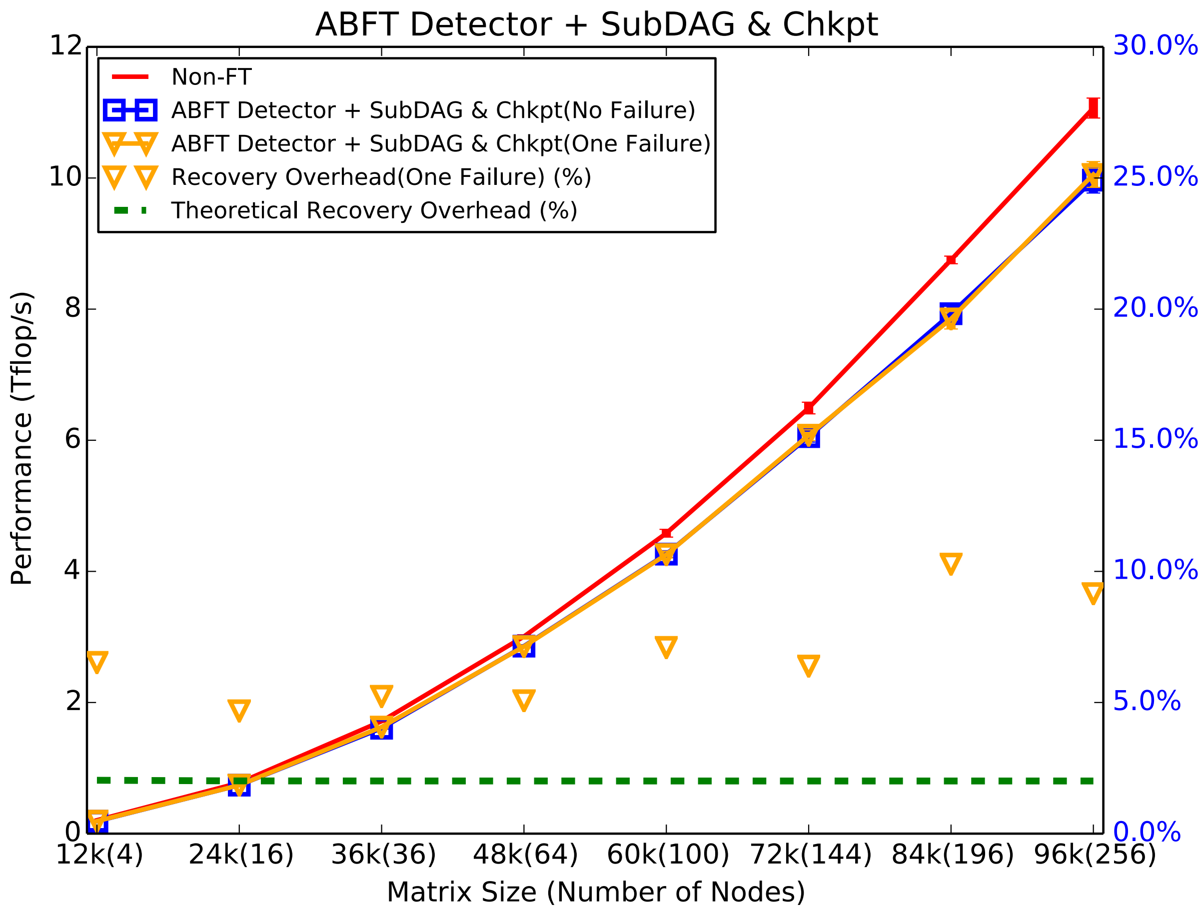
Resilience: *Data Logging Strategy*

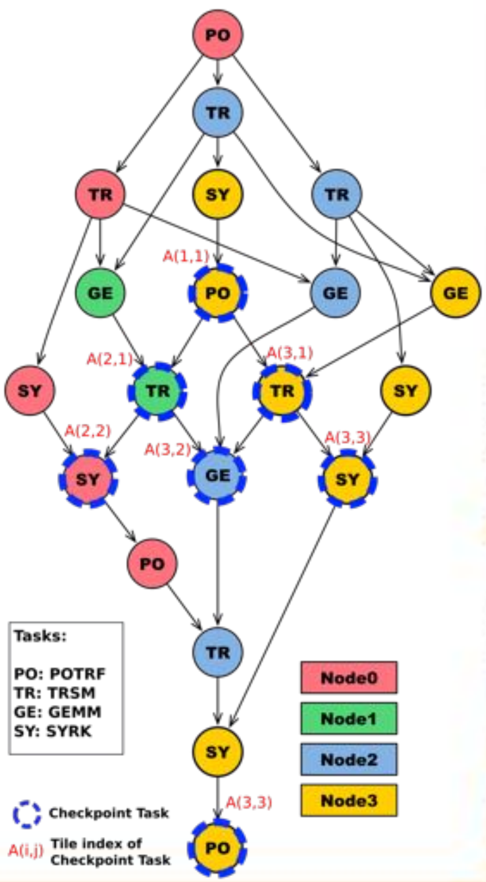
- Minimize the amount of tasks reexecutions by logging data
- Checkpoint interval β, a process will save a copy of each data every β updates.
- Input of failed task:
  - The same tile checkpointed at most β updates ago
  - Final output of another task (validated)
- Max number of re-executions is β for factorizations



Checkpoint	Beginning	Middle	End	No Failure
β	$(NB/N)^3$	$\beta 6(NB/N)^3$	$\beta 6(NB/N)^3$	0
never	$(NB/N)^3$	12.5%	100%	0

Resilience: *Data Logging Strategy*



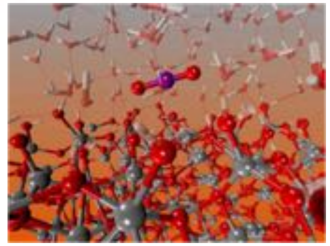


Checkpoint	Beginning	Middle	End	No Failure
β	$(NB/N)^3$	$\beta 6(NB/N)^3$	$\beta 6(NB/N)^3$	0
never	$(NB/N)^3$	12.5%	100%	0

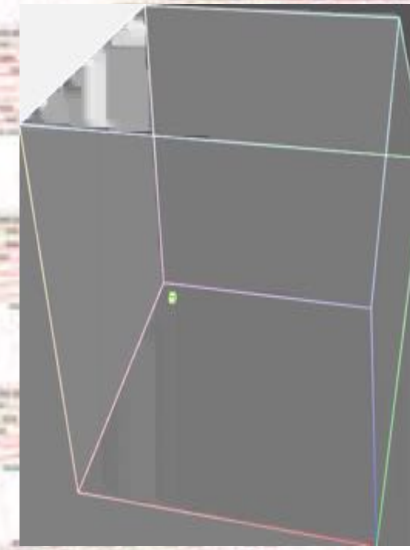
Conclusions

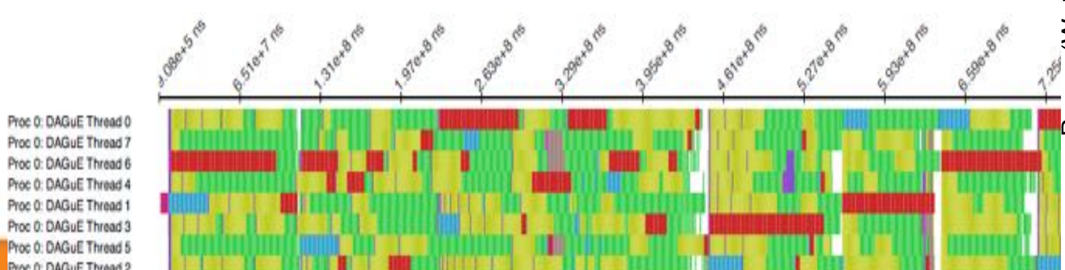
- Don’t make hardware a serious impediment to scientific simulation
- Programming must be made easy(ier)
  - Portability: inherently take advantage of all hardware capabilities
  - Efficiency: deliver the best performance on several families of algorithms
- Build a scientific enabler allowing different communities to focus on different problems
  - Application developers on their algorithms
  - Language specialists on Domain Specific Languages
  - System developers on system issues
  - Compilers on optimizing the task code

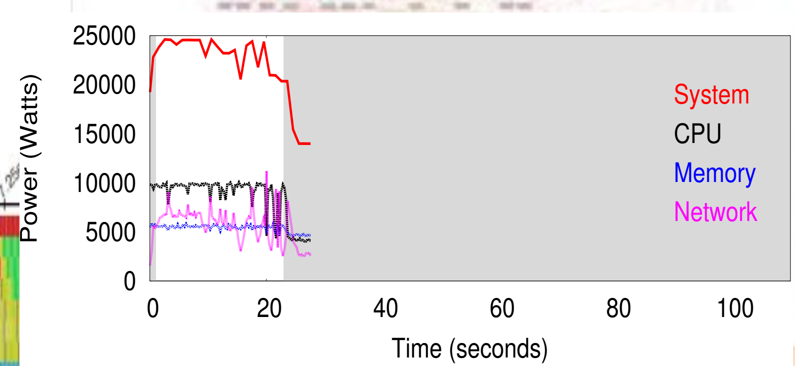
The PaRSEC ecosystem



- Support for many different types of applications
  - Dense Linear Algebra: DPLASMA, MORSE/Chameleon
  - Sparse Linear Algebra: PaSTIX
  - Geophysics: Total - Elastodynamic Wave Propagation
  - Chemistry: NWChem Coupled Cluster, MADNESS, TiledArray
  - *: ScaLAPACK, MORSE/Chameleon
- A set of tools to understand the performance







(b) DPLASMA.