Alternative Programming Models
Distributed tasking with PaRSEC

George Bosilca and many others

BSC, Nov, 2018

IN COLLABORATION WITH

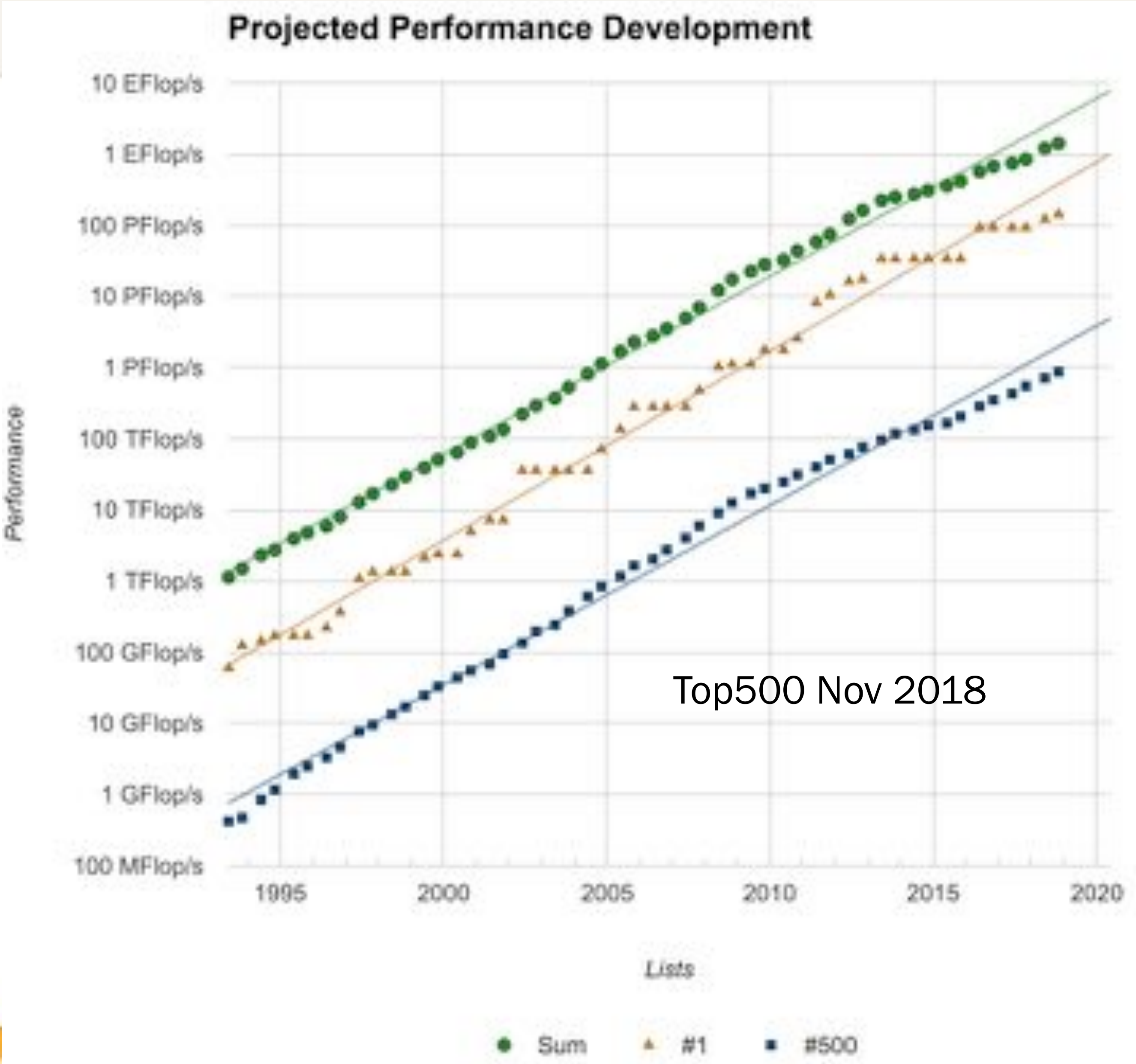
WITH SUPPORT FROM

SPONSORED BY

https://bitbucket.org/icldistcomp/parsec

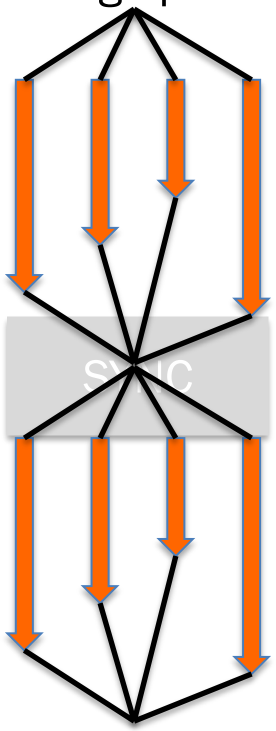
Exponential growth in HPC

- Appetite for compute will continue to grow exponentially
- Fueled by the need to solve many fundamental problems and deal with a growing amount of data
  - Energy, weather forecast, health, understanding the universe but also connected devices, deep learning
- The path forward seems to be a mix of many-core general purpose supported by special purpose units (not necessarily computation only)
- New challenges arise: power, space, cost, reliability, memory, …
  - But also software
  - Hardware solutions that require major changes in software ecosystem are less likely to gain widespread acceptance [quickly]

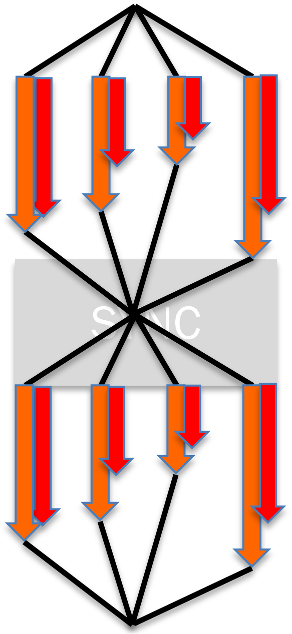


A [very brief] history of computing paradigms

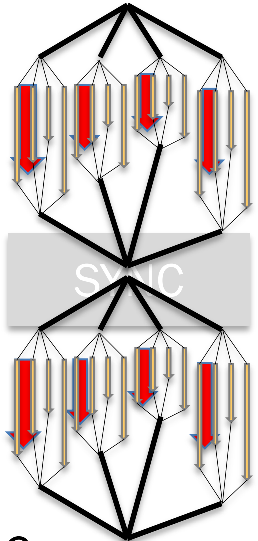
BSP & early message passing



MPI + X



MPI + X + Y + Z + ...



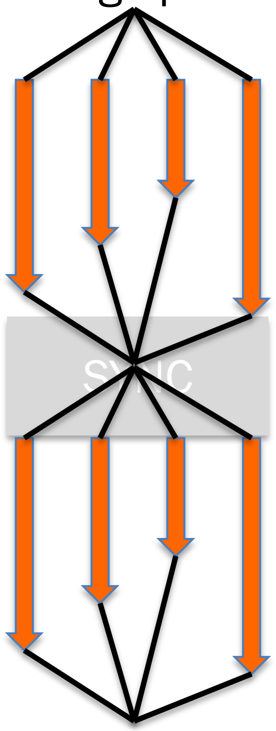
Concurrency*
Heterogeneity
Resiliency

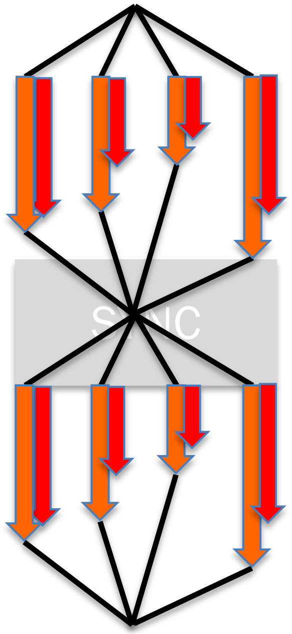
A [very brief] history of computing paradigms

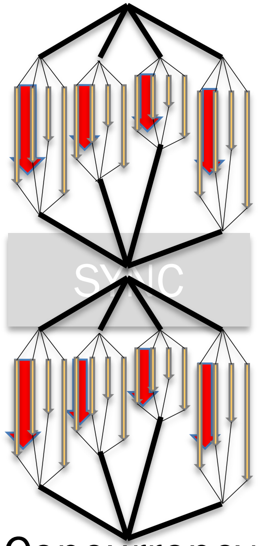
BSP & early message passing

MPI + X

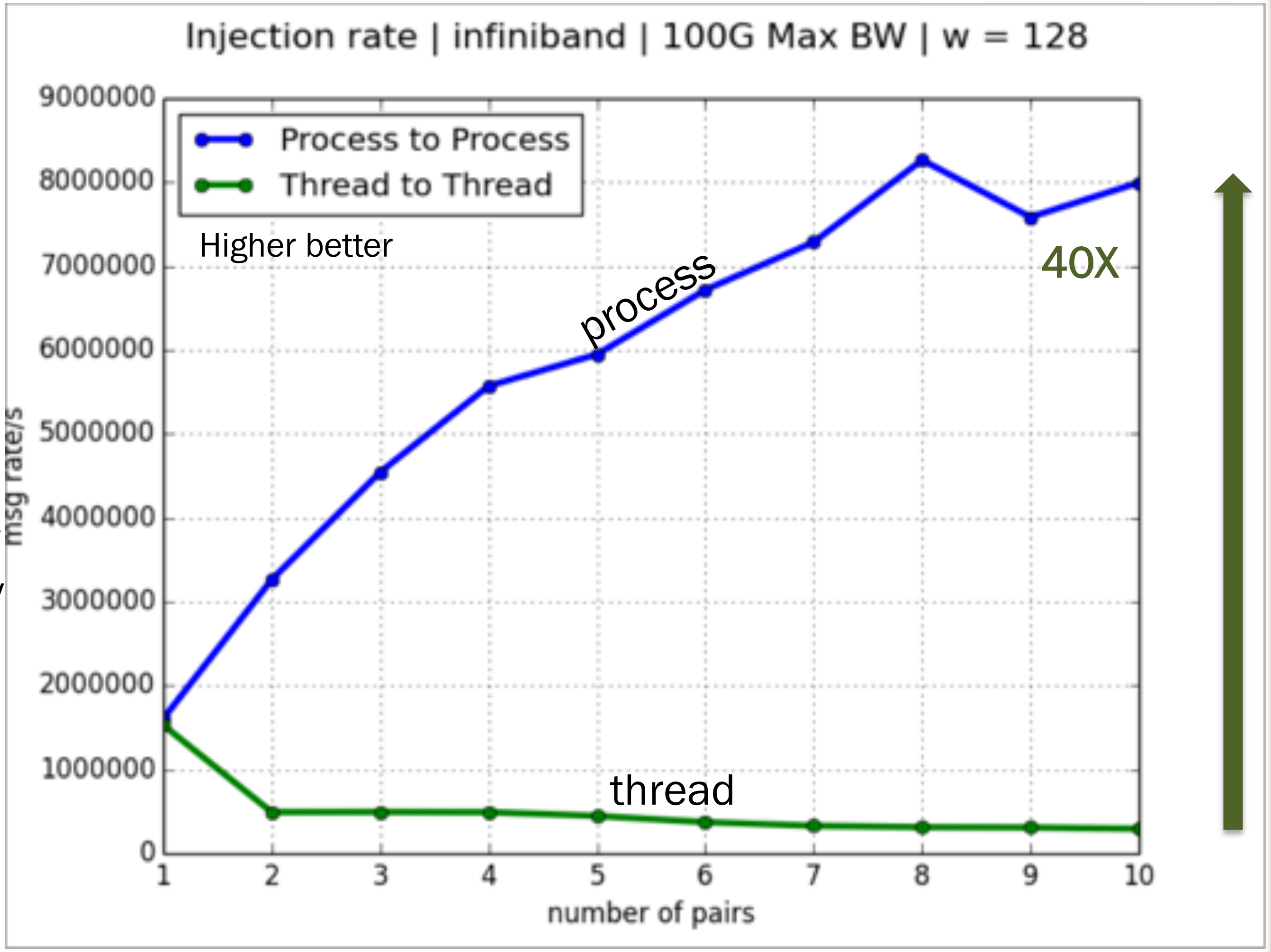
MPI + X + Y + Z + ...





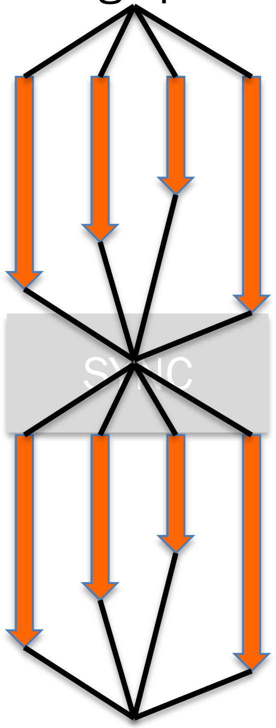


Concurrency*
Heterogeneity
Resiliency

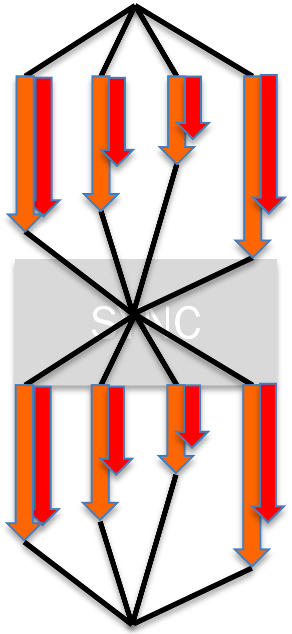


A [very brief] history of computing paradigms

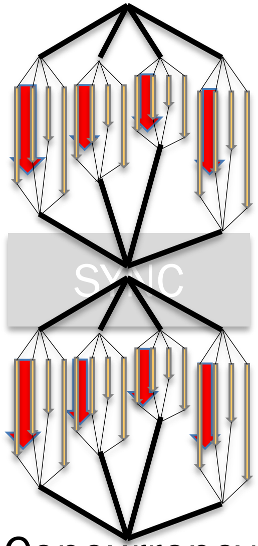
BSP & early message passing



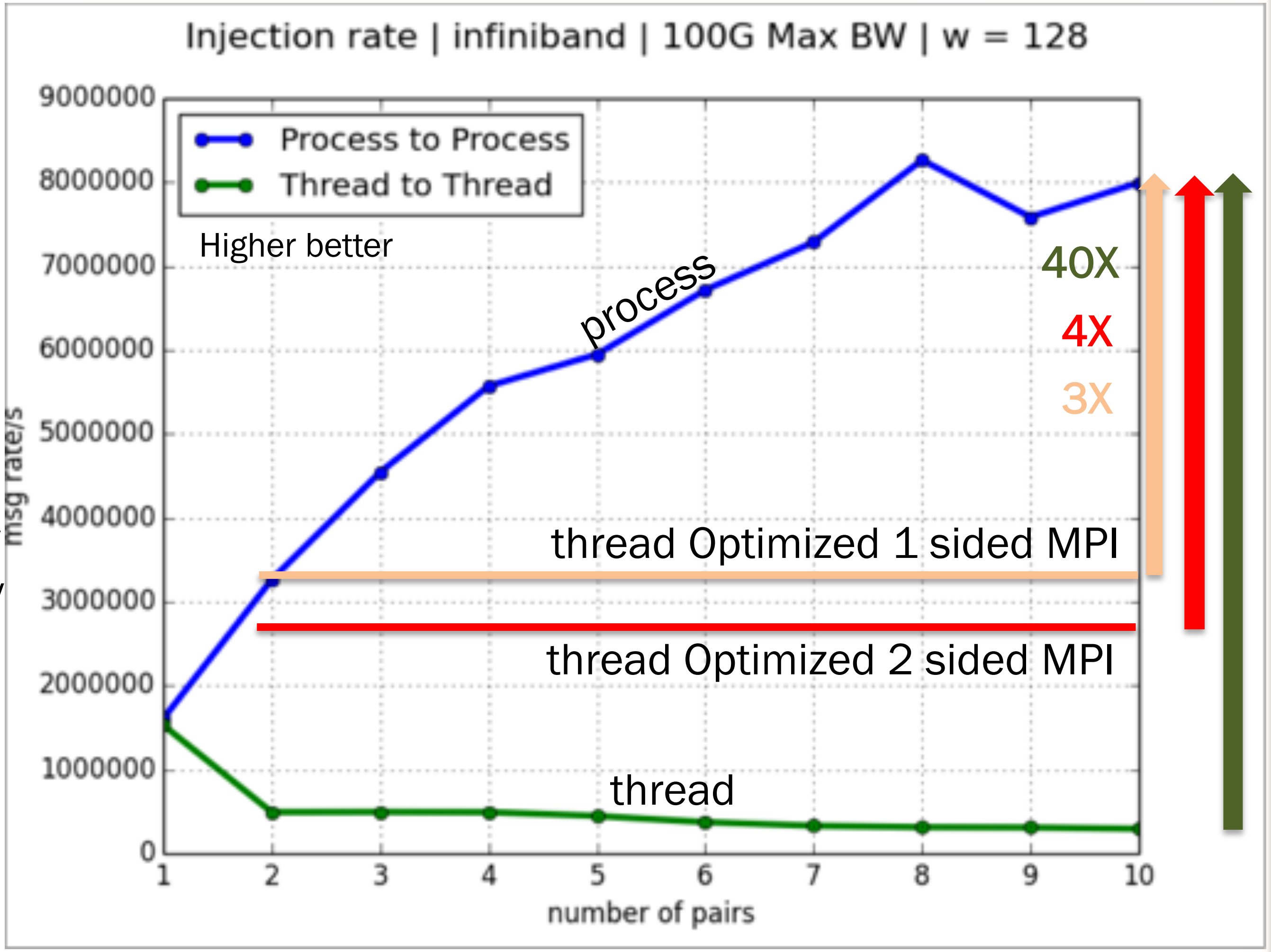
MPI + X



MPI + X + Y + Z + ...



Concurrency*
Heterogeneity
Resiliency

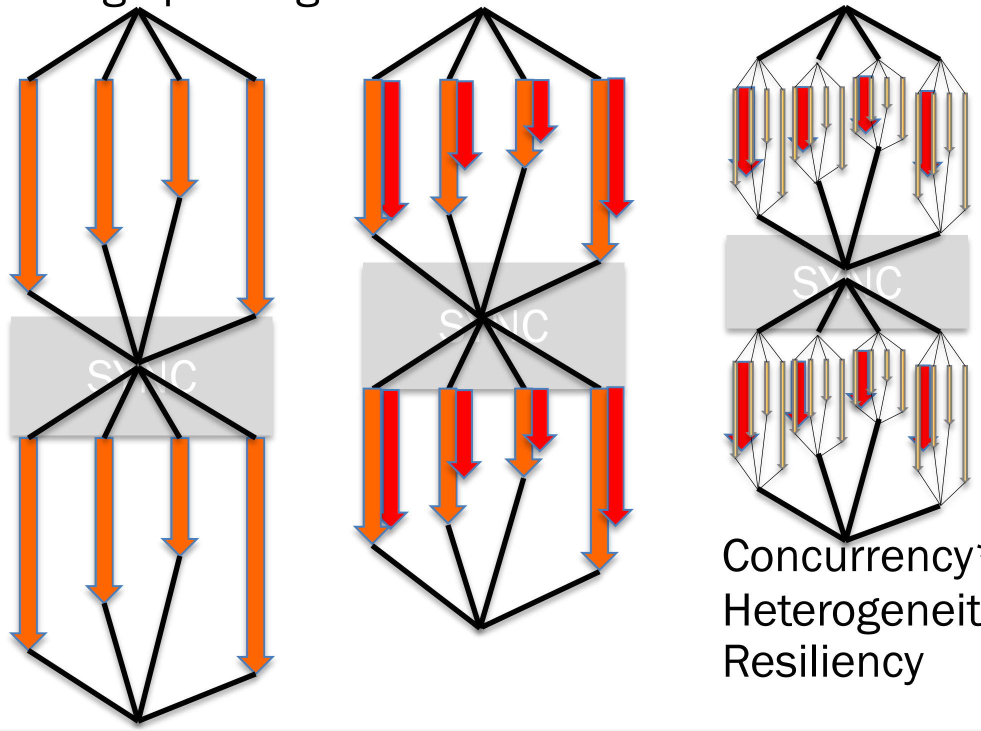


A [very brief] history of computing paradigms

BSP & early message passing

MPI + X

MPI + X + Y + Z + ...



Concurrency*
Heterogeneity
Resiliency

Over-subscription:

- User level threads (Qthreads, MassiveThreads, Nanos++, Argobots)

Task-ification:

- Shared memory: OpenMP, Tascel, Quark, TBB*, PPL, Kokkos**, SuperGluer...
- Distributed Memory: StarPU, StarSS*, DARMA**, Legion, CnC, HPX, Dagger, X10, DuctTeip, Hihat**, ...

* explicit communications
** nascent effort

- Difficult to express the potential inter-algorithmic parallelism
  - Why are we still struggling with 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 about developers productivity ?

Task-based programming support

MPI (1992)
- More of a communication library than a runtime
- Explicit communication

Charm++ (1993)
- Supports distributed memory
- Low level – not based on dataflow (communication is explicit)

DAGuE/PaRSEC (2010)
- Supports distributed memory
- Data-flow based
- Implicit communication

OMPSs/StarSs (2010)
- Supports distributed, master/slave model

Legion (2012)
- Nascent support for distributed memory
- Data-flow based

StarPU (2012)
- Support shared memory
- Communication possible through tasks(explicit)

OpenMP (2013)
- Introduces dependency tracking for tasks in Standard 4

PaRSEC insert_task (DTD) (2013)

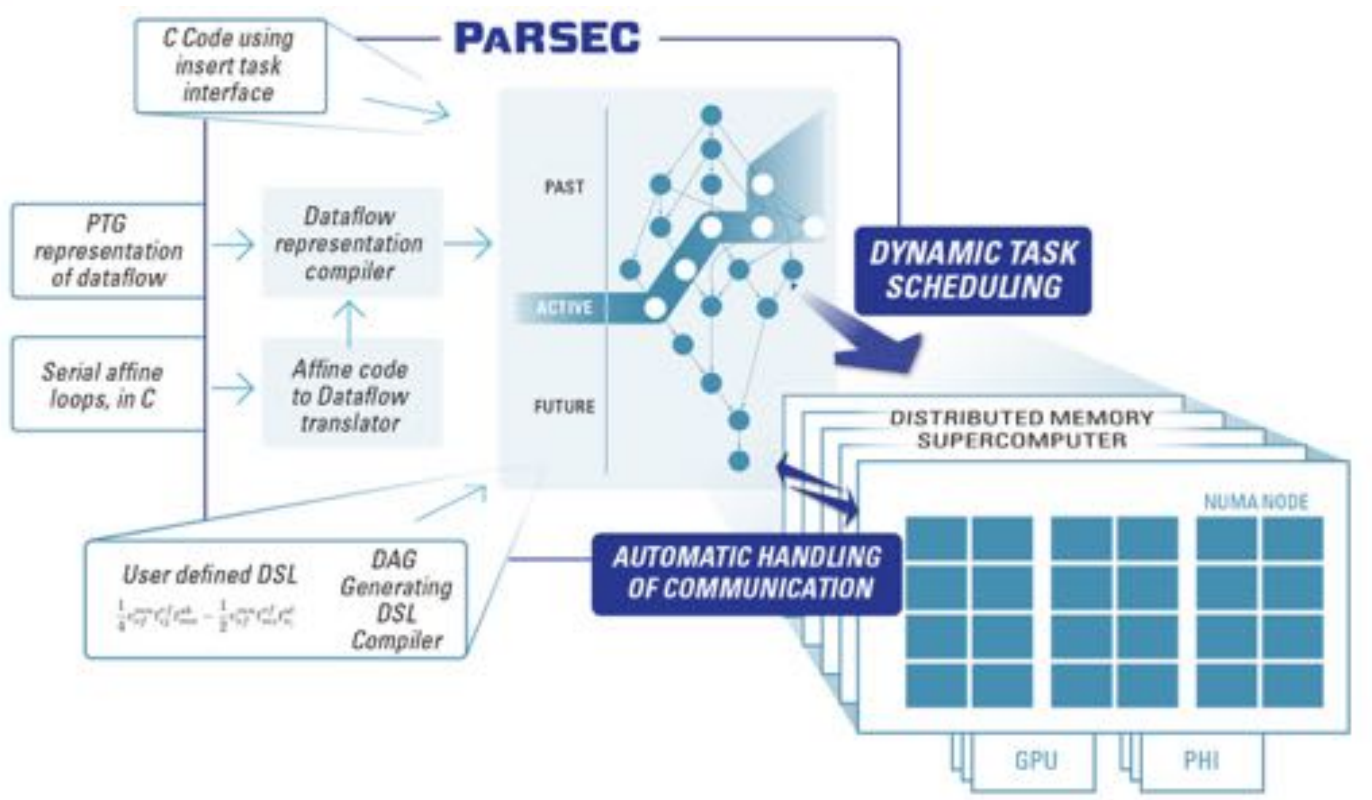
Legion (2017)
- Supports distributed memory
- Focus on compiler to allow programmer express applications more easily

StarPU (2017)
- Supports distributed memory
- Performance most close to DTD

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

Concepts

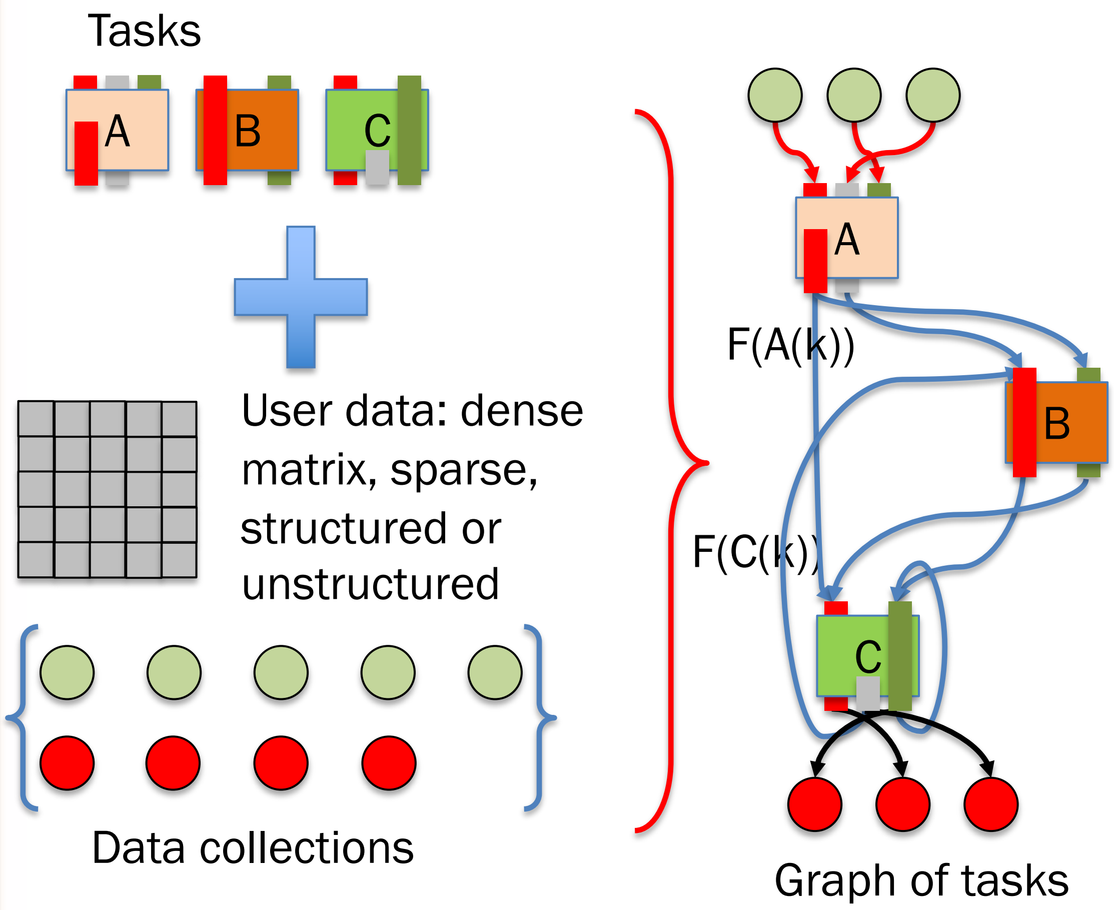
- Clear separation of concerns: compiler optimize each task class, developer describe dependencies between tasks, the runtime orchestrate the dynamic execution
- Interface with the application developers through specialized domain specific languages (PTG/JDF/TTG, Python, insert_task, fork/join, …)
-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



Runtime

- 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

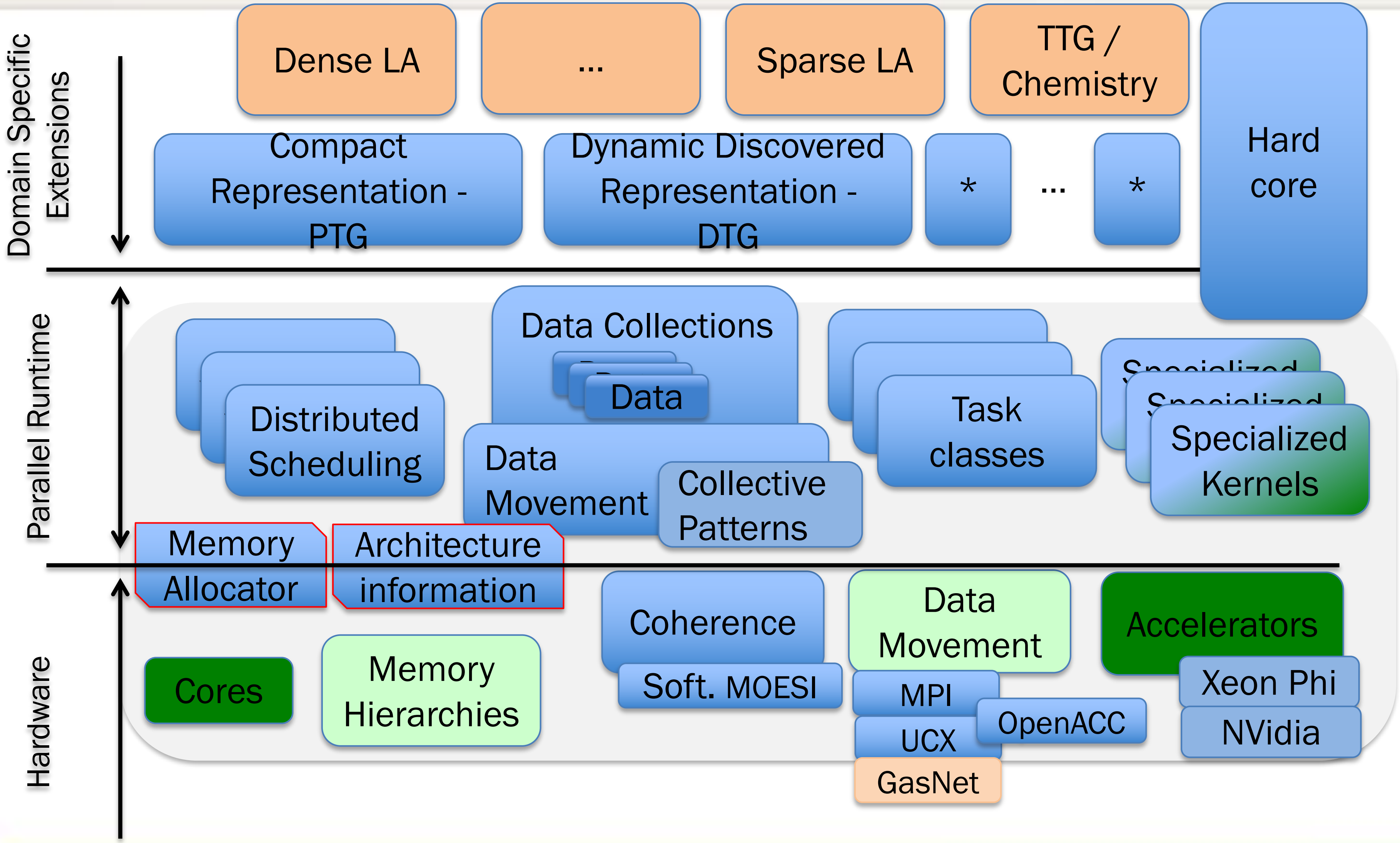
PaRSEC



= a data centric programming environment based on asynchronous tasks executing on a heterogeneous distributed environment

- An execution unit taking a set of input data and generating, upon completion, a different set of output data
- Data have a coherent distributed scope managed by the runtime (similar to promises)
- Low-level API allowing the design of Domain Specific Languages (JDF, DTD, TTG)
- Supports distributed heterogeneous environments
  - Communications are implicit (the runtime moves data)
  - Resources (threads, accelerators) are dynamic encapsulated in distributed domains (similar to executors)
  - Built-in resilience, performance instrumentation and analysis (R, python)

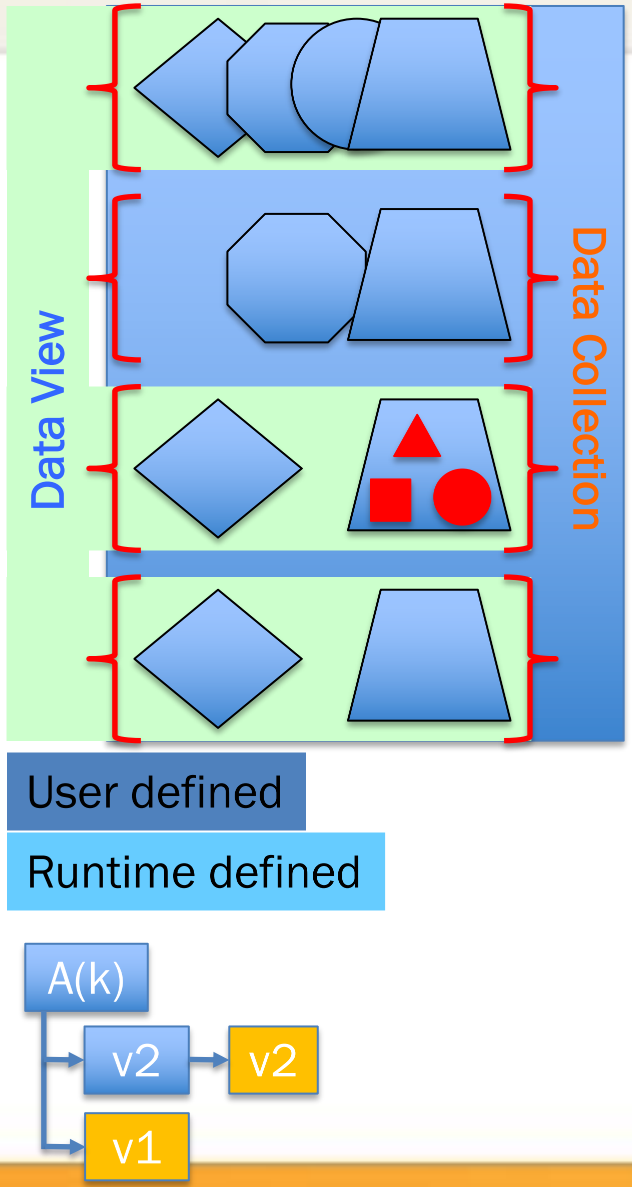
PaRSEC Architecture



Software design based on Modular Component Architecture (MCA) of Open MPI.

- Well defined components API
- Runtime selection of components
- Providing a new capability by implementing a new component has no impact on the rest of the software stack.
    - Can be provided as dynamic libraries by vendors

The PaRSEC data



- A data is a manipulation token, the basic logical element (view) used in the description of the dataflow
  - Locations: 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 user-defined
- 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

DSL: The PaRSEC application

Define a distributed collection of data (here 1 dimension array of integers)

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

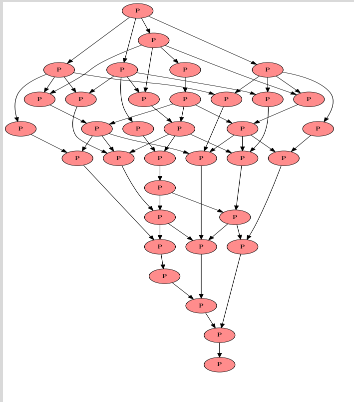
Start PaRSEC (resource allocation)

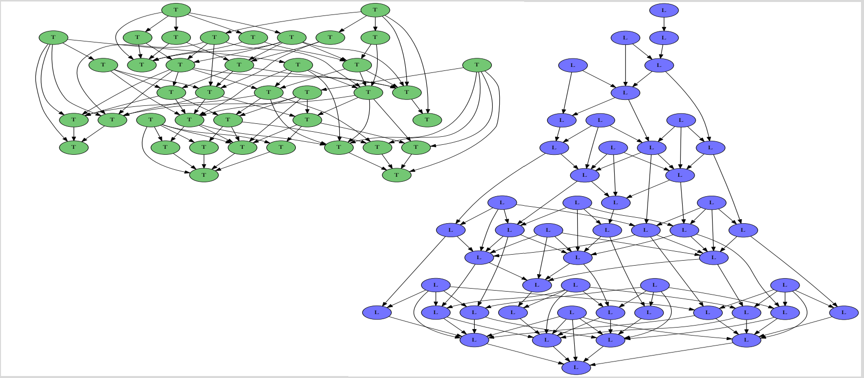
```
parsec_context_t* parsec;
parsec = parsec_context_init(NULL, NULL); /* start the PaRSEC engine */
```

Create a tasks placeholder and associate it with the PaRSEC context

```
parsec_taskpool_t* ts = parsec_taskpool_new ();
parsec_context_add_taskpool (parsec, ts);

parsec_context_start(parsec);
```

Add tasks. A configurable window will limit the number of pending tasks





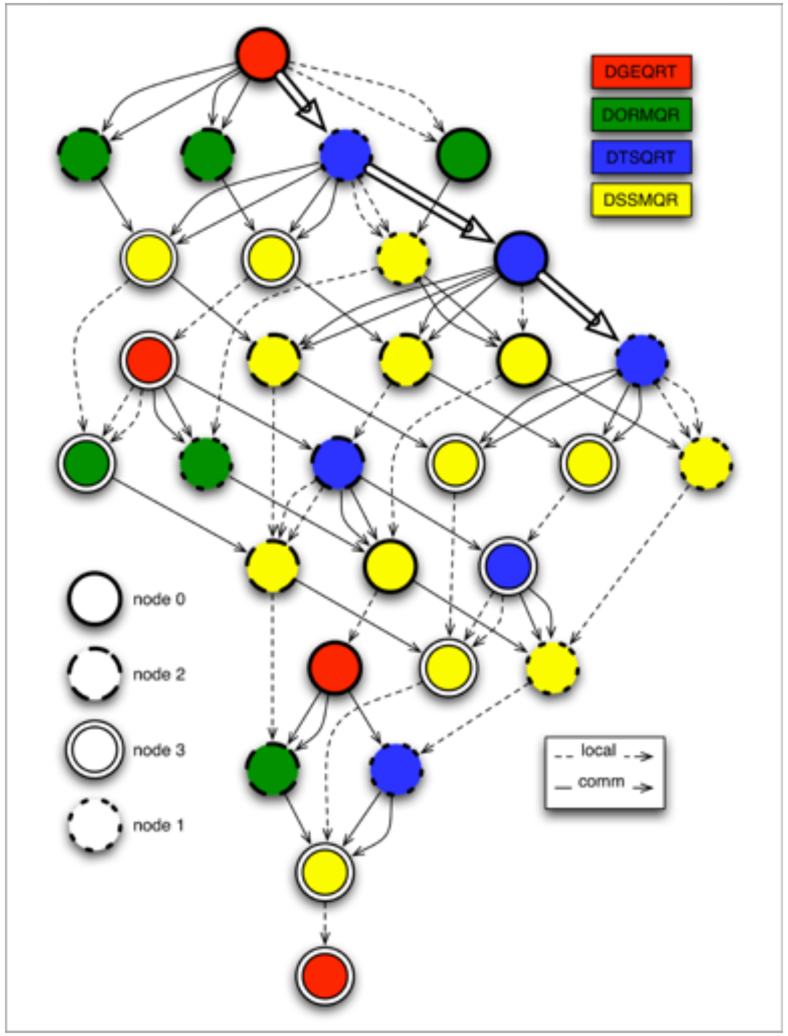
Wait 'till completion

```
parsec_context_wait(parsec);
```

Data initialization and PaRSEC context setup. Common to all DSL

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, Uintah, DARMA, Kokkos, RAJA, OMPSS
  - 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 internal dataflow depiction



DSL: The insert_task interface

Define a distributed collection of data (here 1 dimension array of integers)

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

Start PaRSEC (resource allocation)

```
parsec_context_t* parsec;
parsec = parsec_context_init(NULL, NULL);  /* start the PaRSEC engine */
```

Create a tasks placeholder and associate it with the PaRSEC context

```
parsec_taskpool_t* ts = parsec_taskpool_new ();
parsec_context_add_taskpool (parsec, ts);

parsec_context_start(parsec);
```

Add tasks. A configurable window will limit the number of pending tasks

```
for( n = 0; n < N; n++ ) {
    parsec_insert_task( ts,
        call_to_kernel_type_write,    "Create Data",
        PASSED_BY_REF,    DATA_AT(dDATA, n),   OUT | REGION_FULL,
        0 /* DONE */);
  for( k = 0; k < K; k++ ) {
    parsec_insert_task( ts,
        call_to_kernel_type_read,    "Read_Data",
        PASSED_BY_REF,    DATA_AT(dDATA, n),   INPUT | REGION_FULL,
        0  /* DONE */ );
    }
}
```

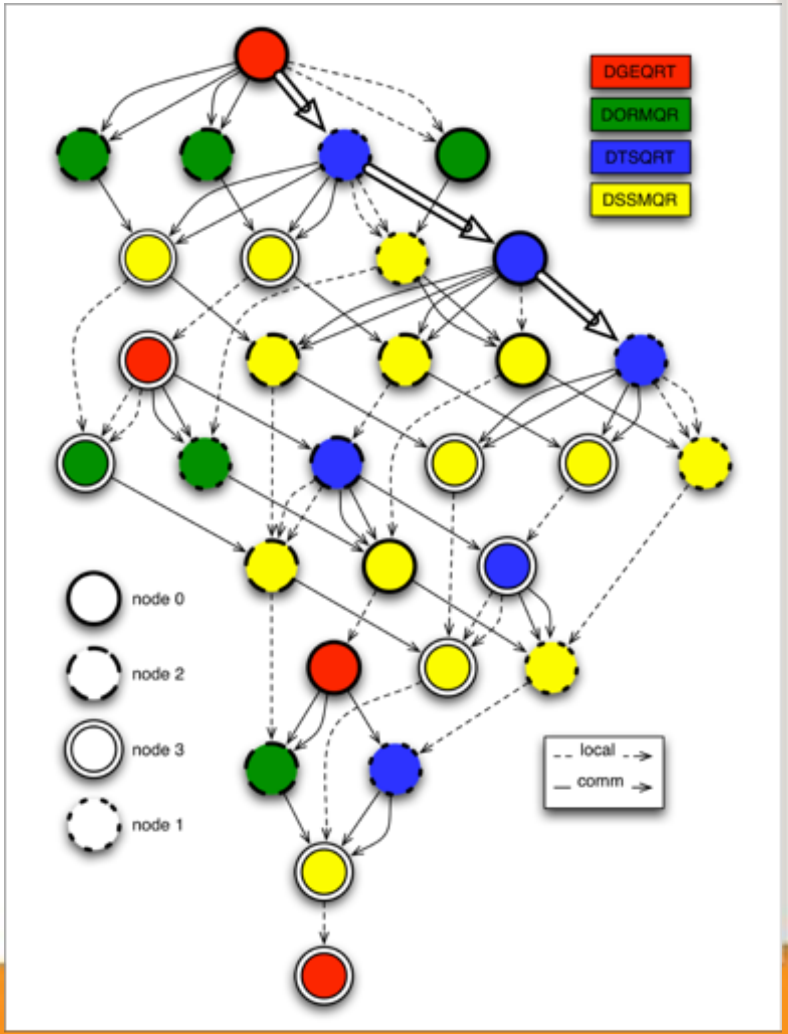
Wait 'till completion

```
parsec_context_wait(parsec);
```

Data initialization and PaRSEC context setup. Common to all DSL

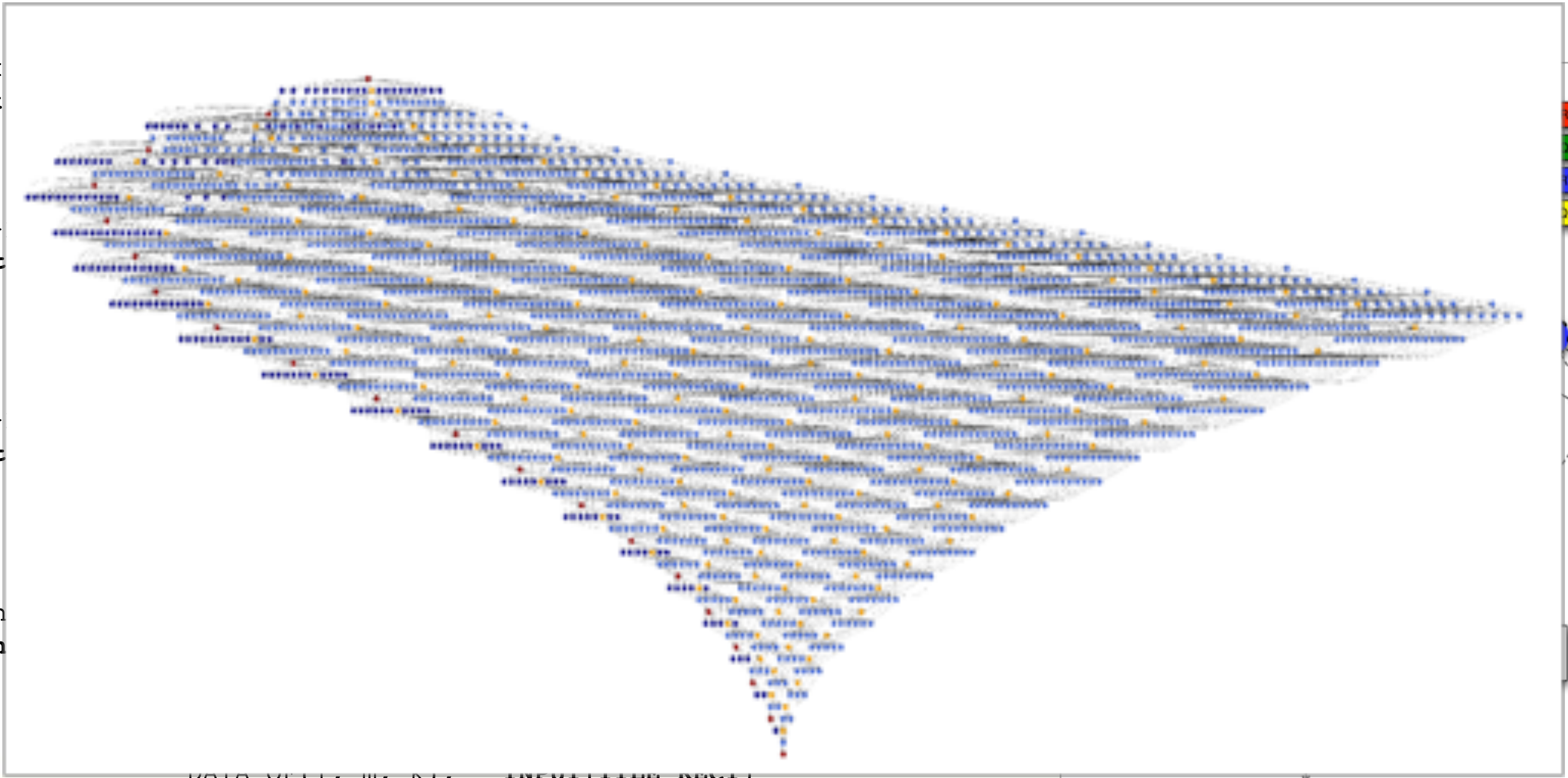
The insert_task in action

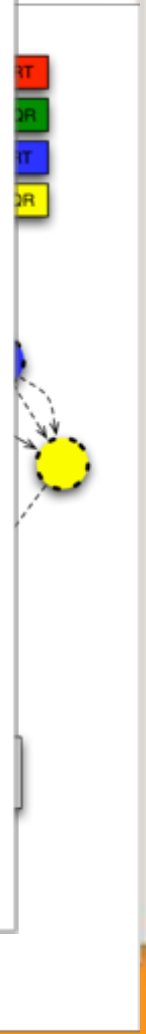
```
for( k = 0; k < SIZE; k++ ) {
    insert_task( "GEQRT",
                 DATA_OF(A, k, k),          INOUT|AFFINITY,
                 DATA_OF(T, k, k),          OUTPUT|TILE_RECT)

    for( n = k+1; n < SIZE; n++ )
        insert_task( "UNMQR",
                     DATA_OF(A, k, k),      INPUT|TILE_L,
                     DATA_OF(T, k, k),      INPUT|TILE_RECT,
                     DATA_OF(A, k, n),      INOUT|AFFINITY)

    for( m = k+1; m < SIZE; m++ ) {
        insert_task( "TSQRT",
                     DATA_OF(A, k, k),      INOUT|TILE_U,
                     DATA_OF(A, m, k),      INOUT|AFFINITY,
                     DATA_OF(T, m, k),      OUTPUT|TILE_RECT)

        for( n = k+1; n < SIZE; n++ )
            insert_task( "TSMQR",
                         DATA_OF(A, k, n),  INOUT,
                         DATA_OF(A, m, n),  INOUT|AFFINITY,
                         DATA_OF(A, m, k),  INPUT,
                         DATA_OF(T, m, k),  INPUT|TILE_RECT)
    }
}
```



The insert_task in action

```
for( k = 0; k <
    insert_task

    for( n = k+
        insert

    for( m = k+
        insert

        for( n
            in

               DATA_OF(T, m, k),   INPUT|TILE_RECT)
    }
}
```





Overhead of insert_task

DTD overhead

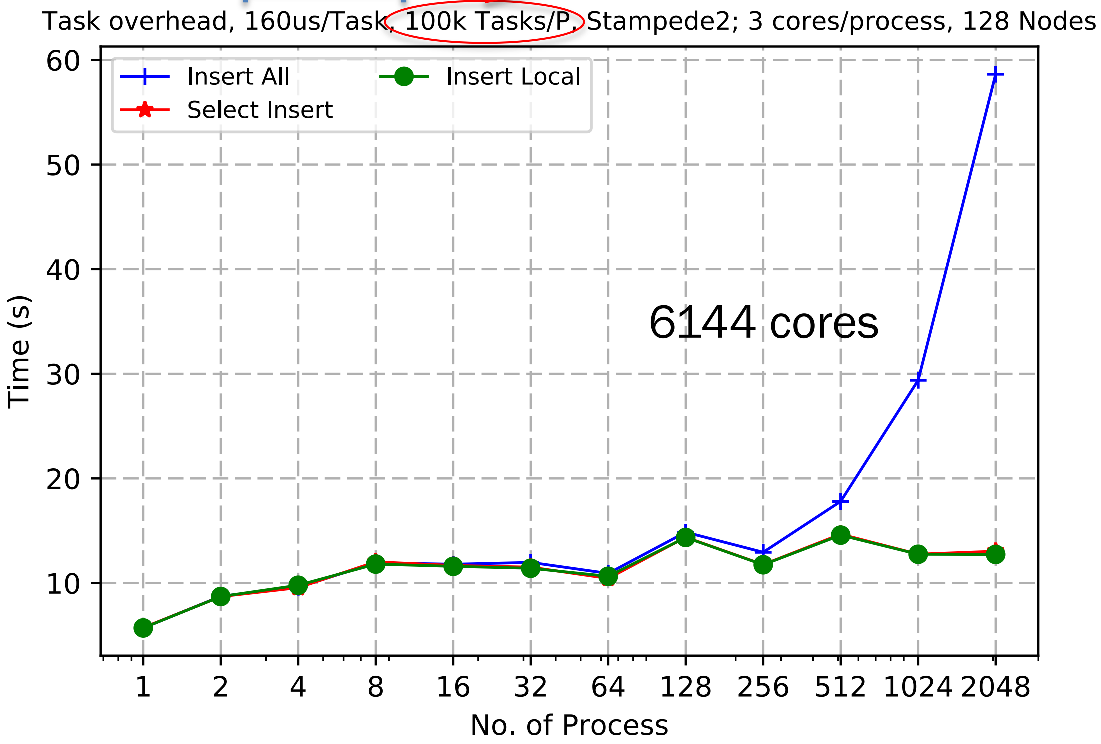
$$T_{DTD} = \frac{N \times C_T}{P \times n} + N \times C_D + \frac{N \times C_R}{P}$$

$T_{DTD/PTG}$: Overall time
N: Total number of tasks
C_T: Cost/duration of each task
P: Total number of nodes/process
n: Total number of cores
C_D: Cost of discovering a task
C_R: Cost of building DAG/relationship

Benefits: critical path is defined by the sequential ordering
Drawbacks: impossible to build collective patterns, selecting the window size is difficult, all data movement must be known globally (and their order is critically important)

Fixed task duration

Weak scaling: Fixed number of tasks per process



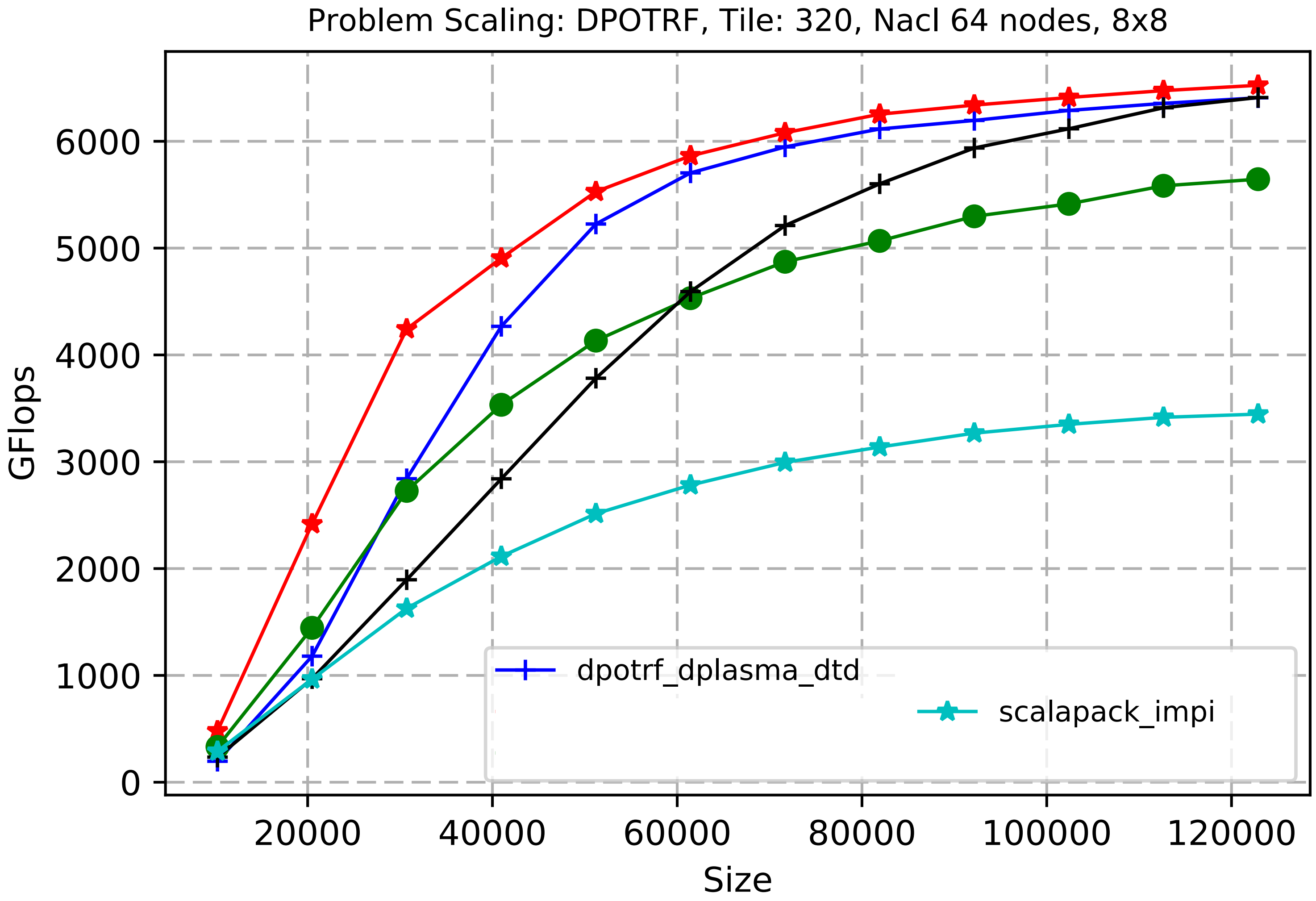
- There are three types of scenario
  - Insert All: Each rank inserts all tasks, and executes only locals
  - Select Insert: Each rank inserts only local tasks, but iterates over all tasks.
  - Insert Local: Each rank only inserts local tasks.

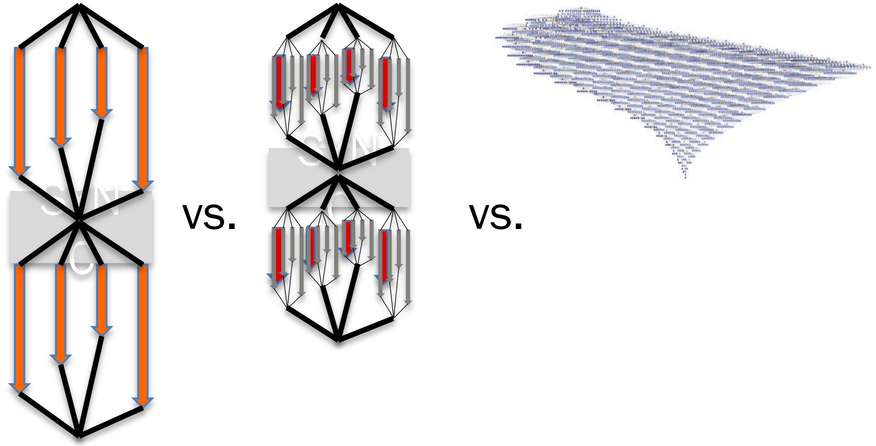
What's missing from insert_task?

- Need to balance between task graph knowledge and memory overhead
  - The task graph creation must happen in a single thread
- To trim or not to trim ? Who is tracking the data in order to orchestrate global data coherence ?

- Difficult to type the input and output data, especially if one expects the dependencies to only apply on partial data
- Difficult to reliably expose collective patterns without complete knowledge of the task graph as different processes might have discovered different sections of the task graph

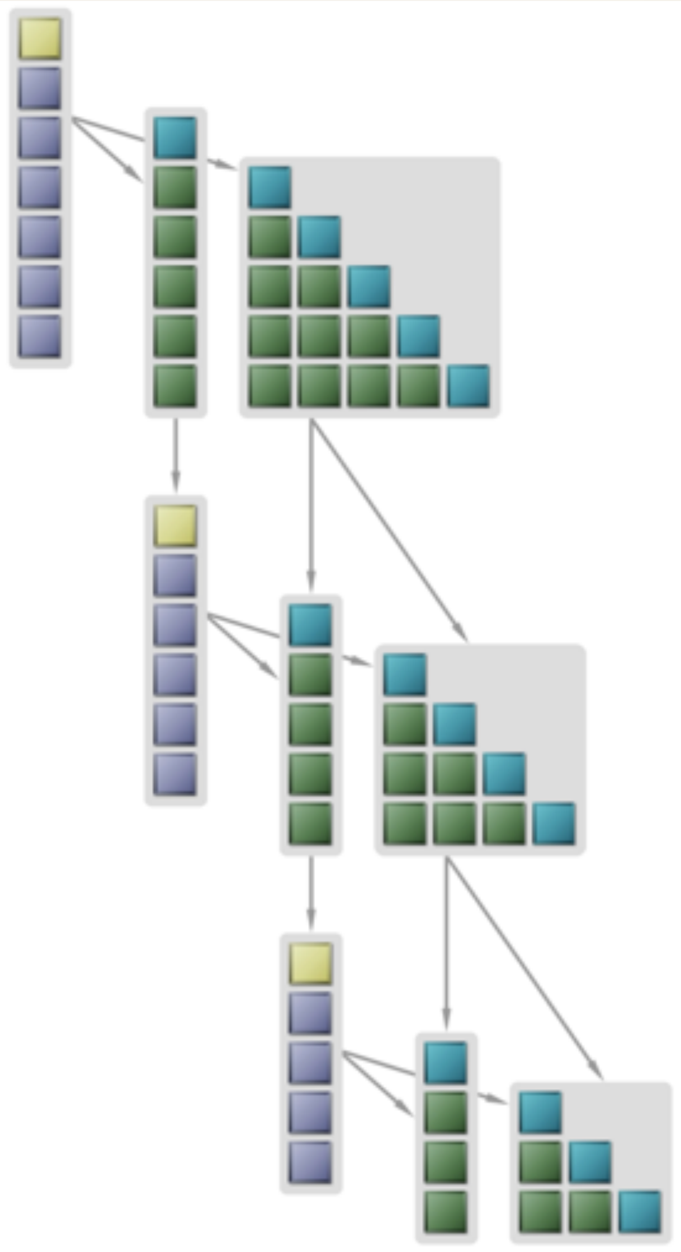
PaRSEC DSL comparaison



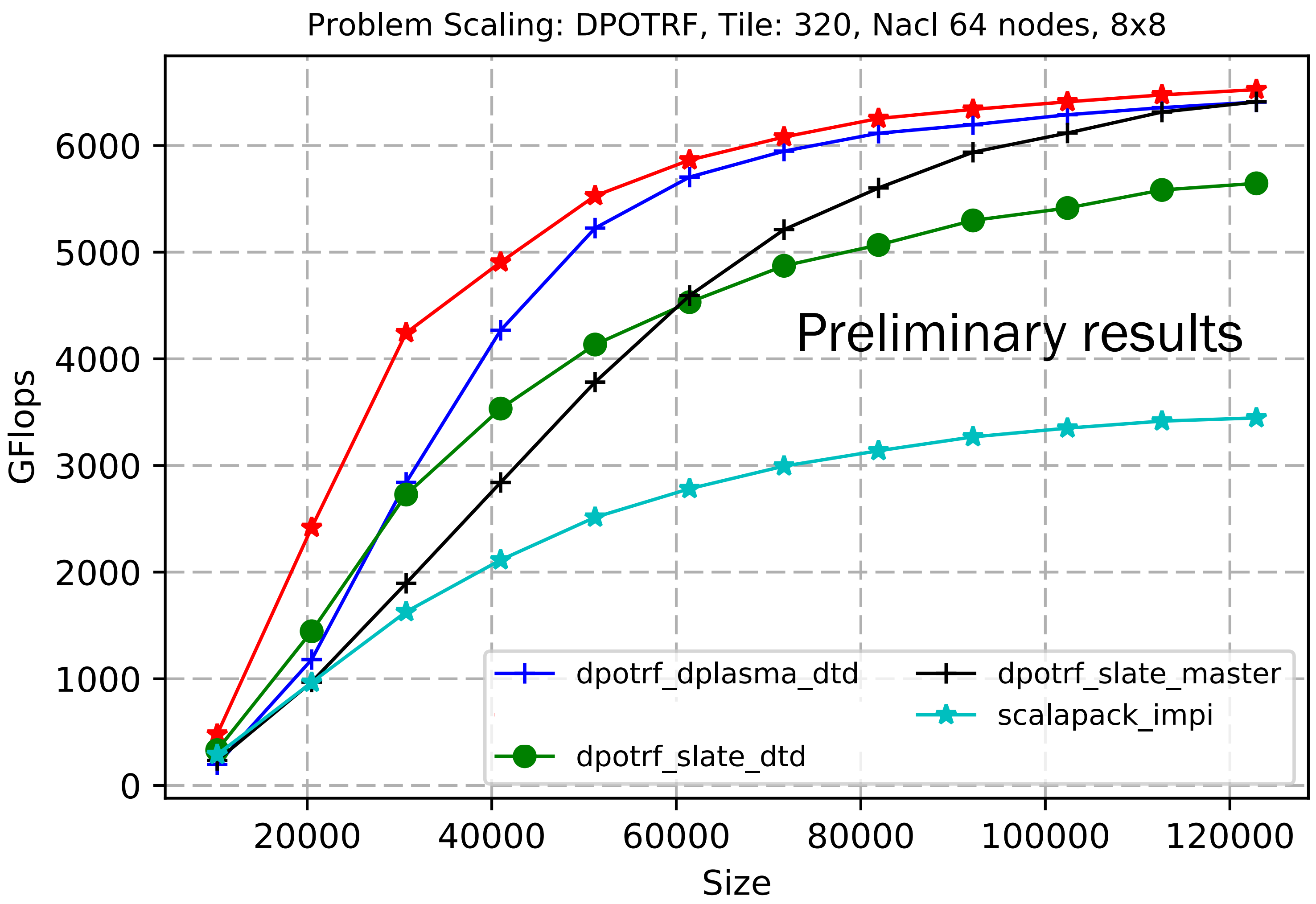


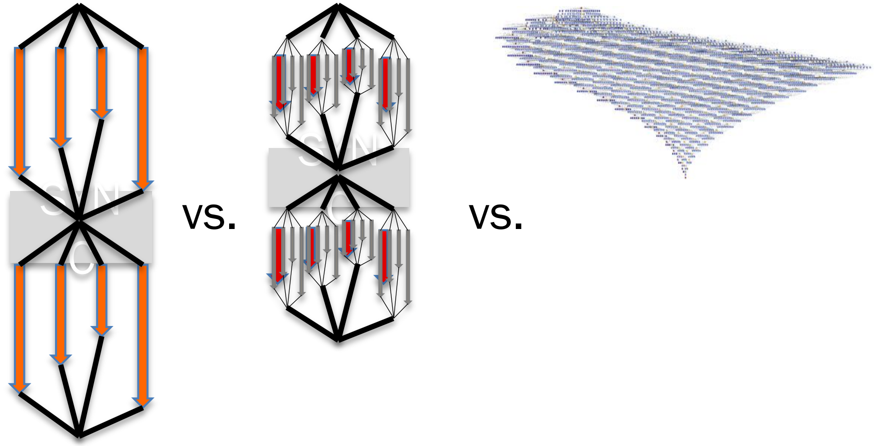
SLATE (over PaRSEC) approach

- Back to the future: return to the ScaLAPACK approach, the problem hierarchically divided (panel + update) with flexible lookahead
  - Remove most data dependencies (except data movement and versioning)
    - The design is flexible enough to allow good performance on runtimes with high scheduling costs (such as OpenMP or StarPU)
  - Variable granularity with several benefits: task duration, task location, well exposed “batched” operation
    - Potential benefit for accelerators
  - Present a different view to data movement
    SLATE.send(data, [data range]+)
    - Explicit life-expectancy for remote data
    - Expose collective communications
- Everything is done in templated C++11, but the resulting programming model (DSL) is generic



PaRSEC DSL comparaison





The Parameterized Task Graph (PTG/JDF)

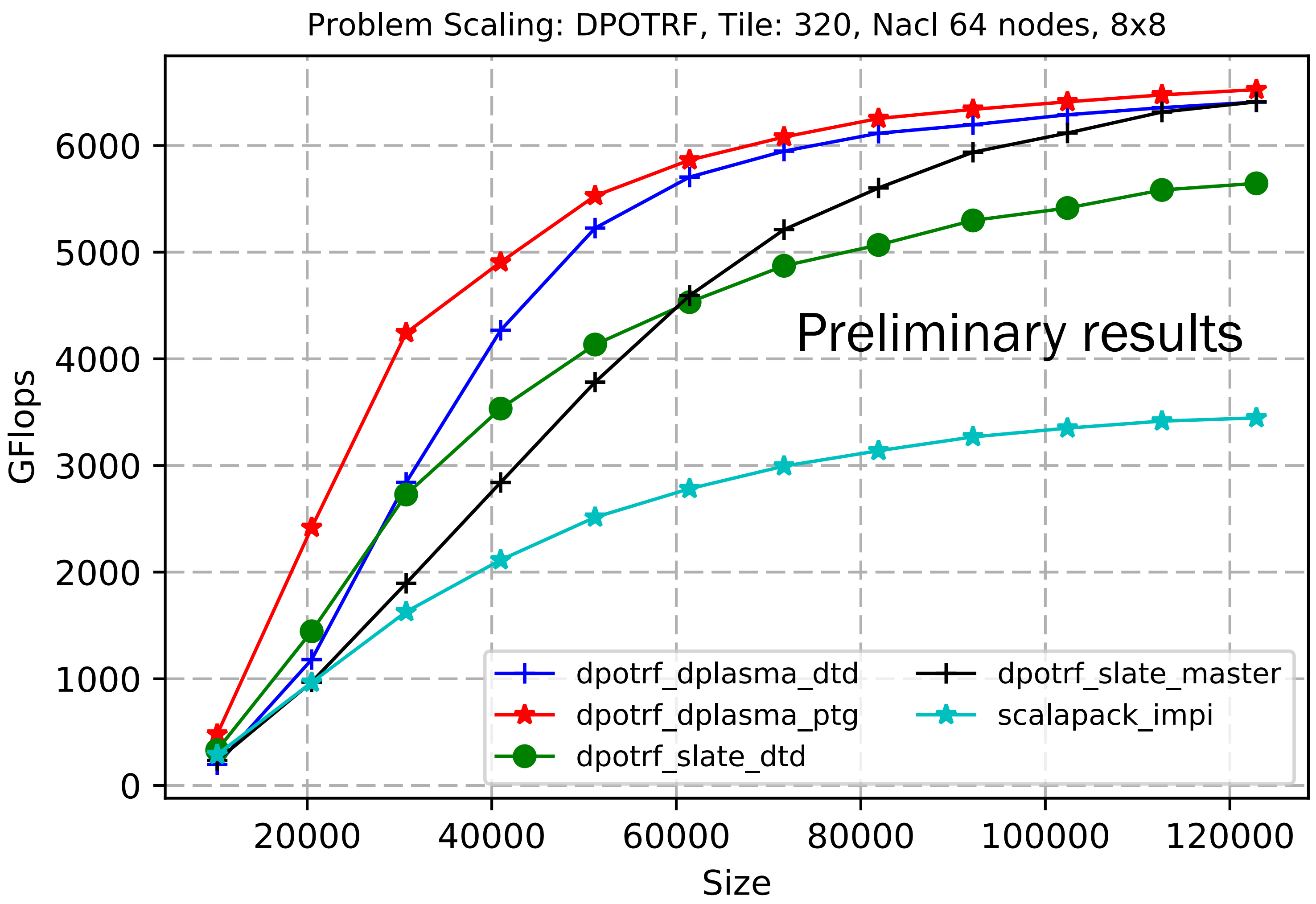
```
GEQRT(k)

 k = 0..( MT < NT ) ? MT-1 : NT-1 )

 : 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)

BODY [type = CPU]  /* default */
   zgeqrt( A, T );
END

BODY [type = CUDA]
   cuda_zgeqrt( A, T );
END
```

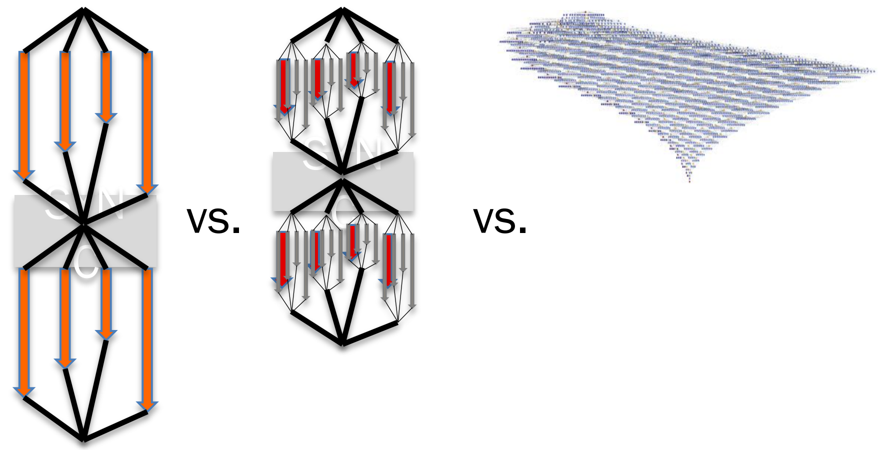
Control flow is eliminated, therefore maximum parallelism is possible

- A dataflow parameterized and concise language
- Accept non-dense iterators
- Allow inlined C/C++ code to augment the language [any expression]

- Termination mechanism part of the runtime (i.e. needs to know the number of tasks per node)
- The dependencies had to be globally (and statically) defined prior to the execution
  - Dynamic DAGs non-natural
  - No data dependent DAGs

PaRSEC DSL comparaison

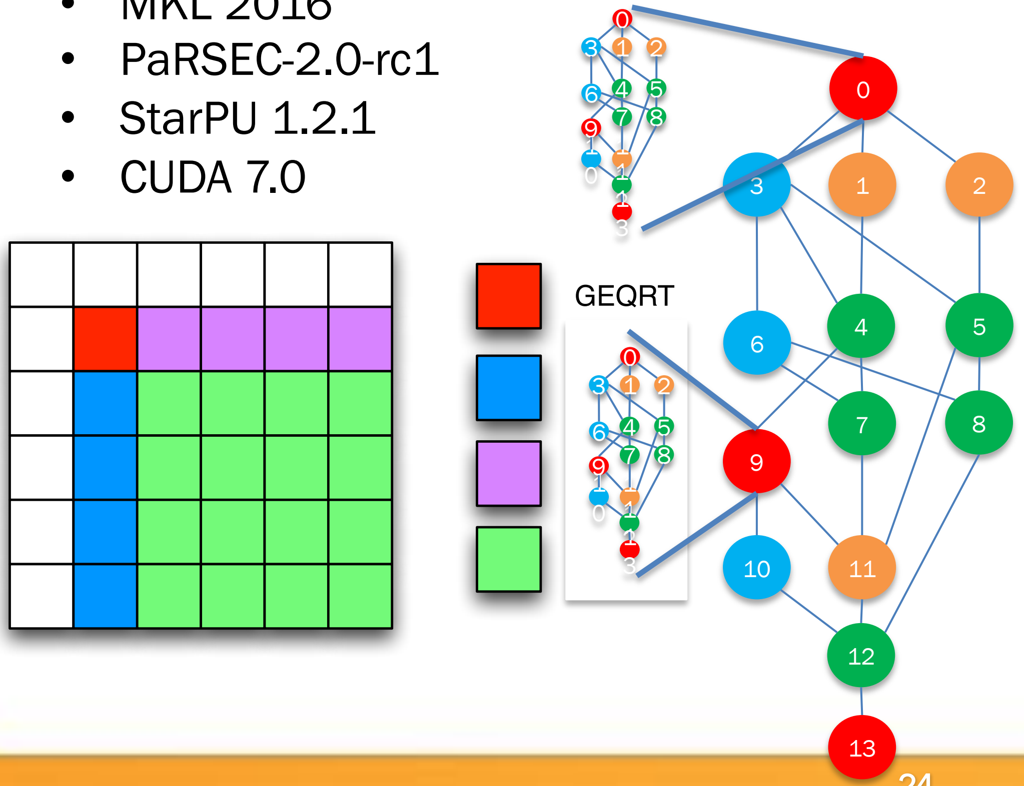


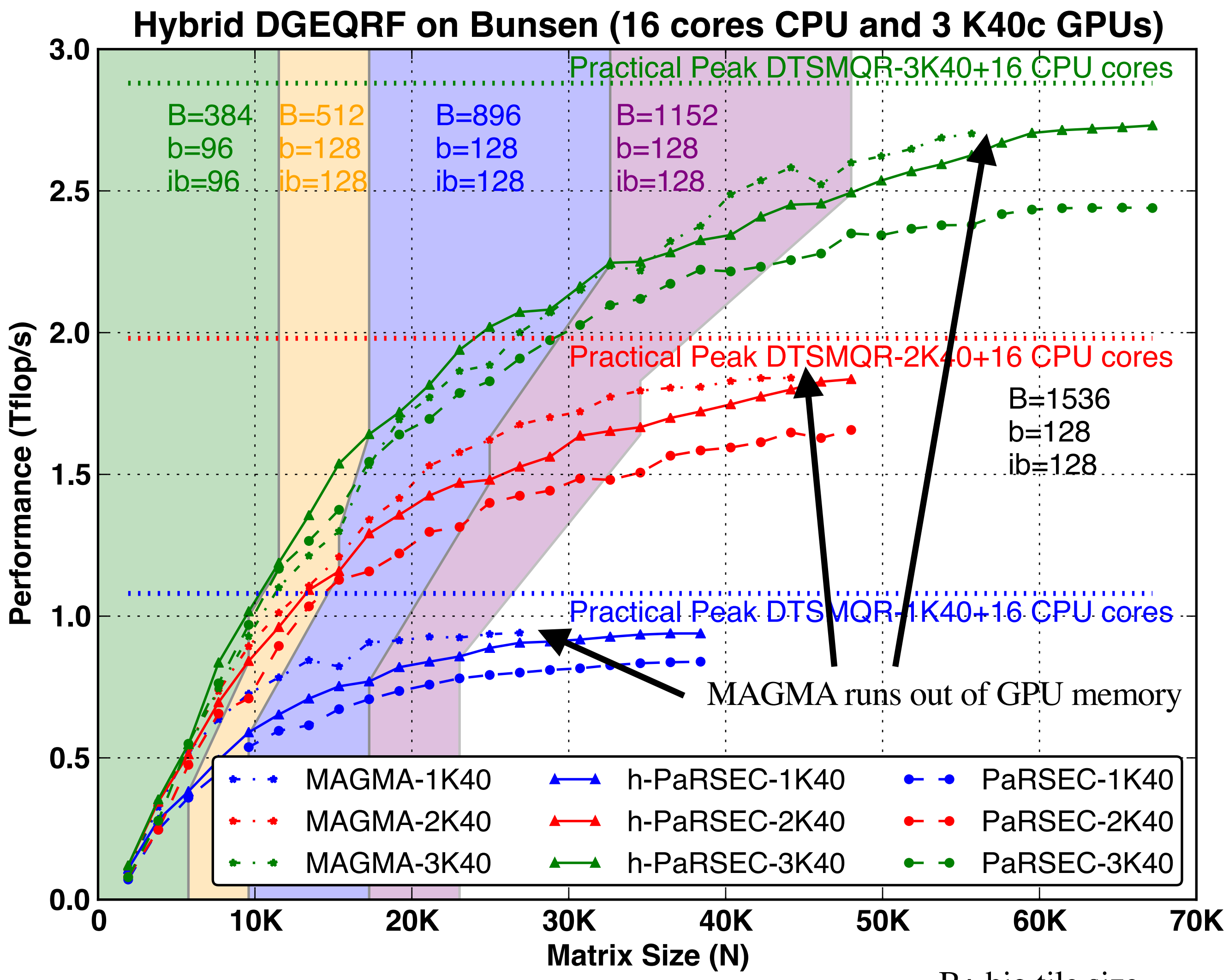


Dense Linear Algebra: QR heterogeneous

Experiments on Arc machines,

- E5-2650 v3 @ 2.30GHz
- 20 cores
- gcc 6.3
- MKL 2016
- PaRSEC-2.0-rc1
- StarPU 1.2.1
- CUDA 7.0

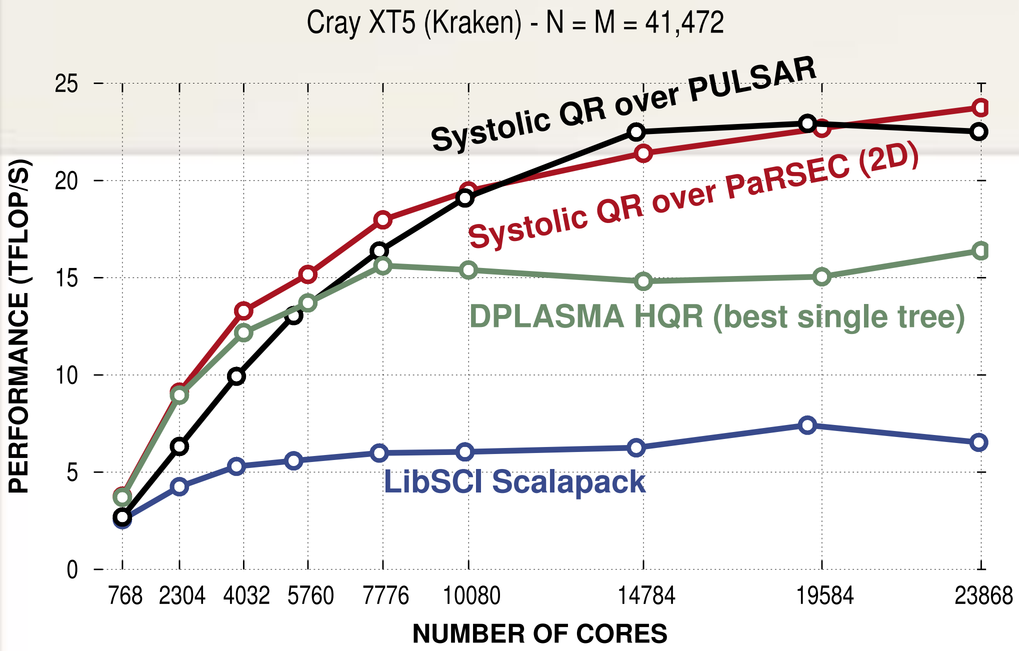


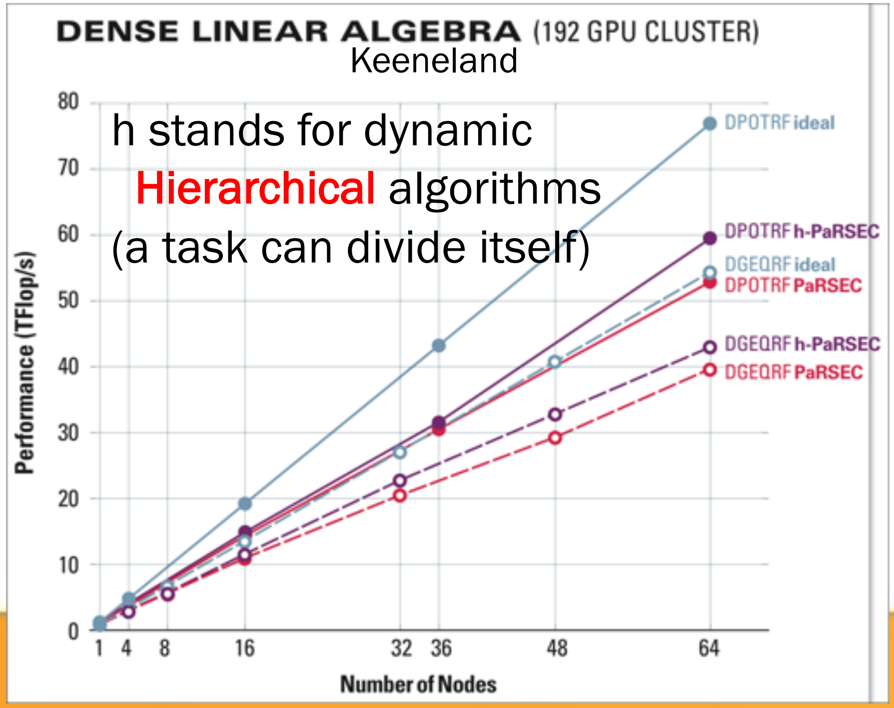


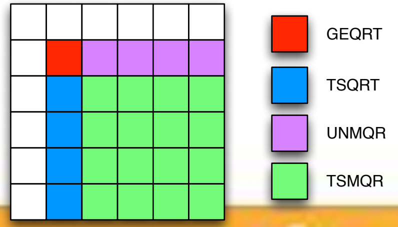
B: big tile size
b: small tile size
ib: inner block size

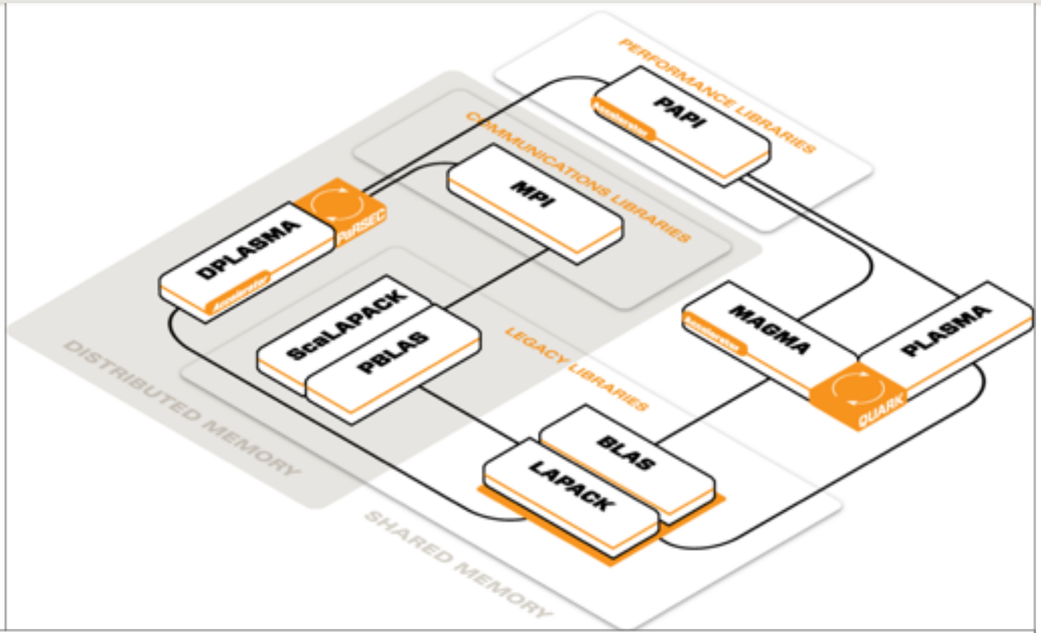
Dense Linear Algebra
DPLASMA = ScaLAPACK + runtime (PaRSEC)

DGEQRF performance strong scaling







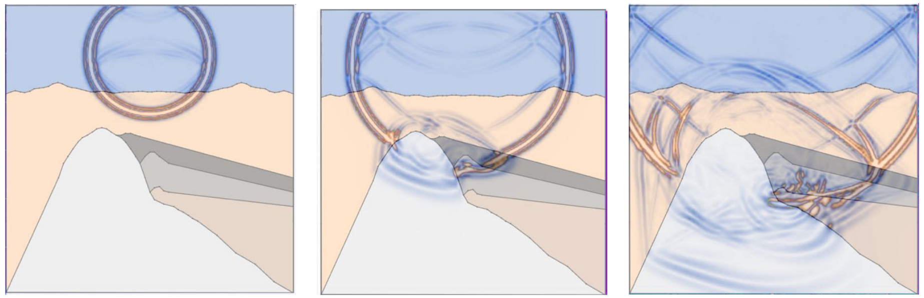


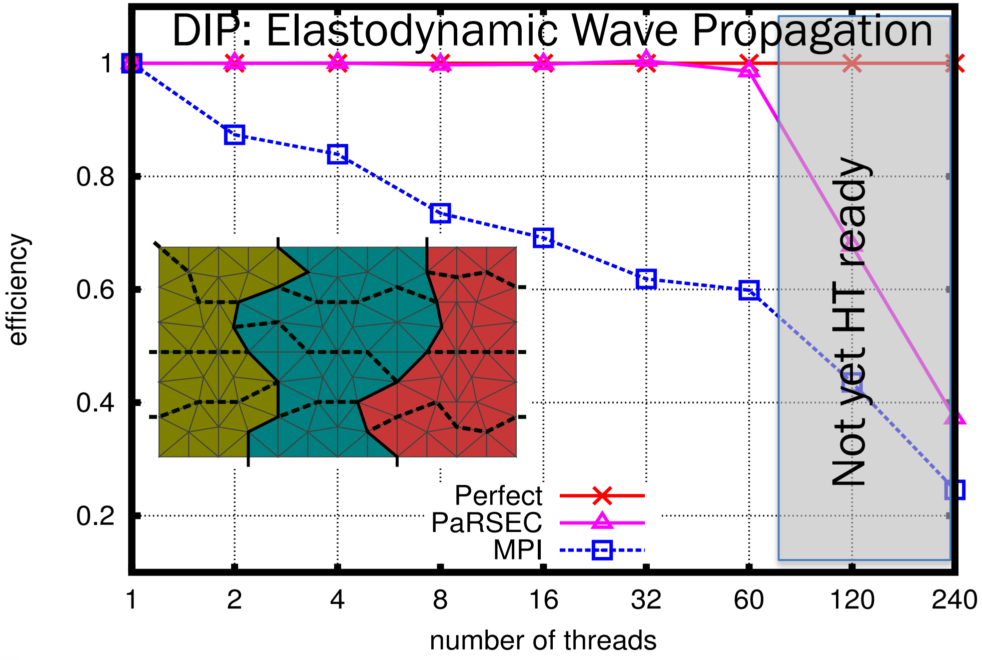
FUNCTIONALITY	COVERAGE
Linear Systems of Equations	Cholesky, LU (inc. pivoting, PP), LDL (prototype)
Least Squares	QR & LQ
Symmetric Eigenvalue Problem	Reduction to Band (prototype)
Level 3 Tile BLAS	GEMM, TRSM, TRMM, HEMM/SYMM, HERK/SYRK, HER2K/SYR2K
Auxiliary Subroutines	Matrix generation (PLRNT, PLGHE/PLGSY,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

Relaxing constraints: Unhindered parallelism

- The only requirement is that upon a task completion the descendants are locally known
  - Information packed and propagated to participants where the descendent tasks are supposed to execute
- Uncountable DAGs
  - ” %option nb_local_tasks_fn = …”
  - PaRSEC provides support for global termination detection (or user provided)
- Add support for dynamic DAGs
  - Properties of the algorithm / tasks
    - ”hash_fn = …”
    - ”find_deps_fn = …”
- Allow dataflow specialization (RMA, datatype, displacement)

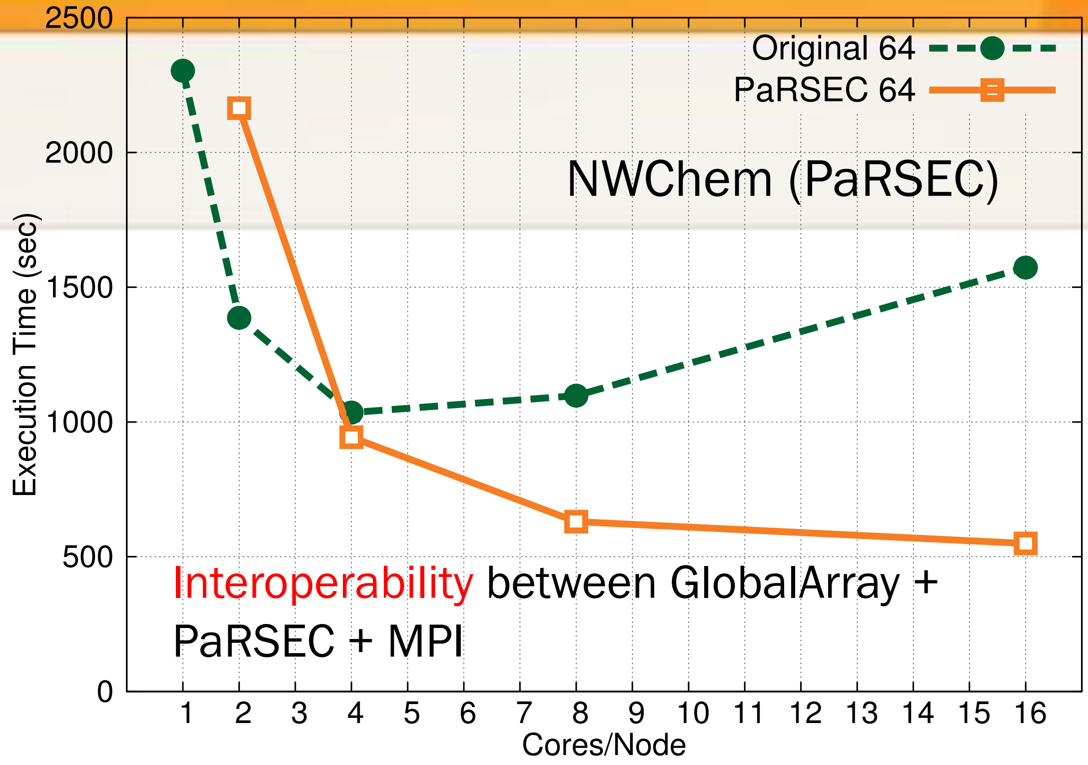
More dynamic applications

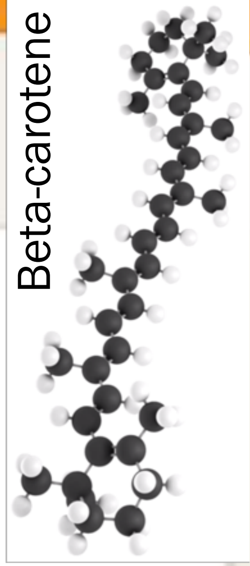




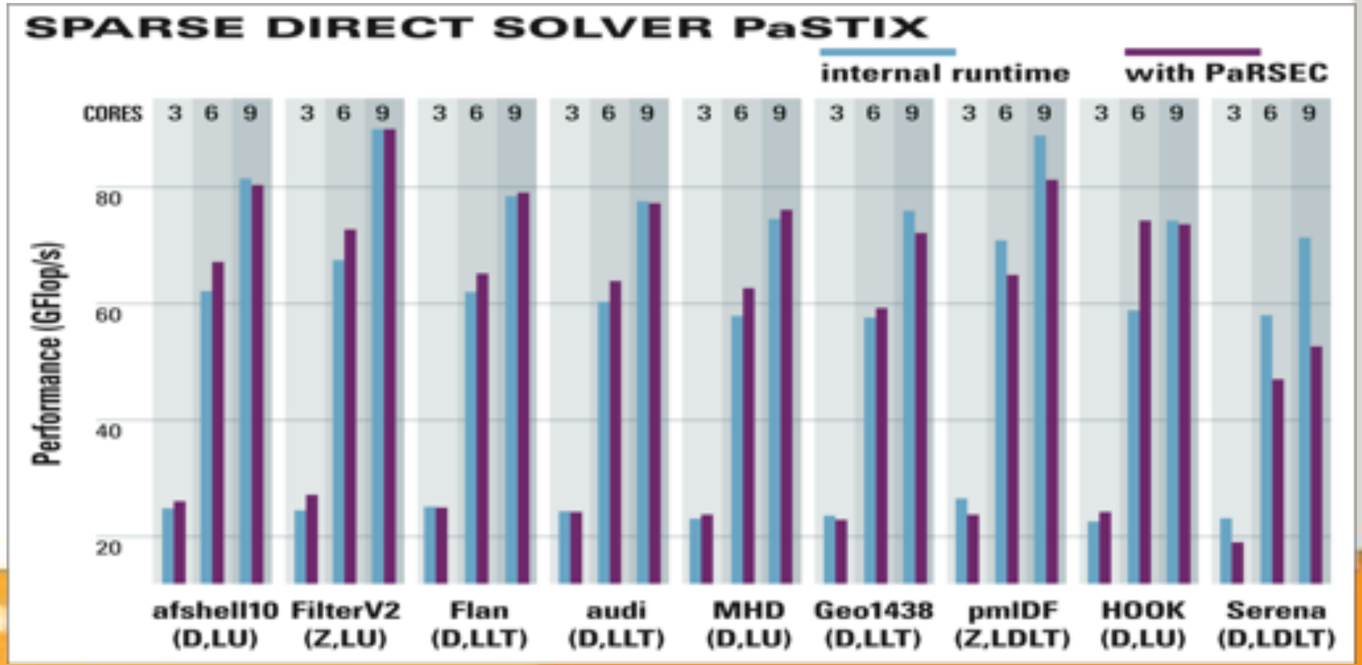
Dynamically redistribute the data

- use PAPI counters to estimate the imbalance
- reshuffle the frontiers to balance the workload

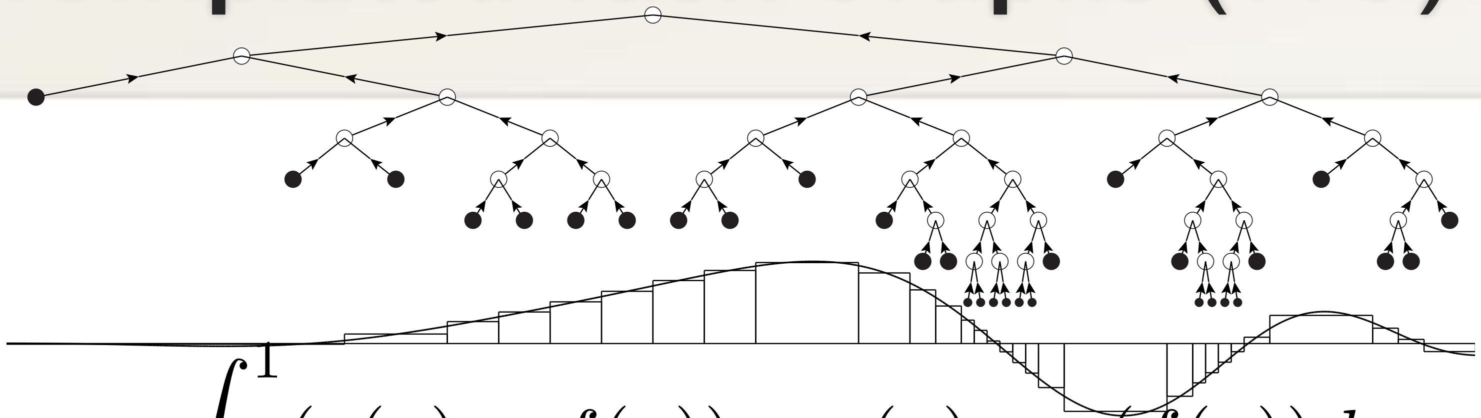




$C_{40}H_{56}$



Templated Task Graphs (TTG)

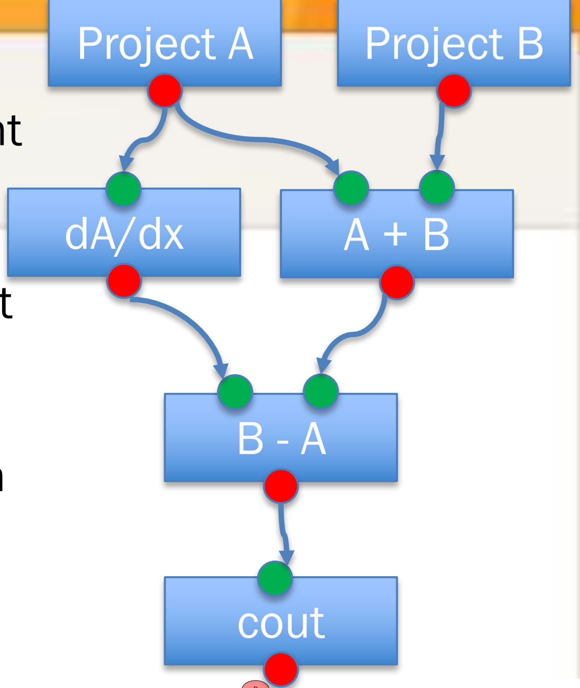


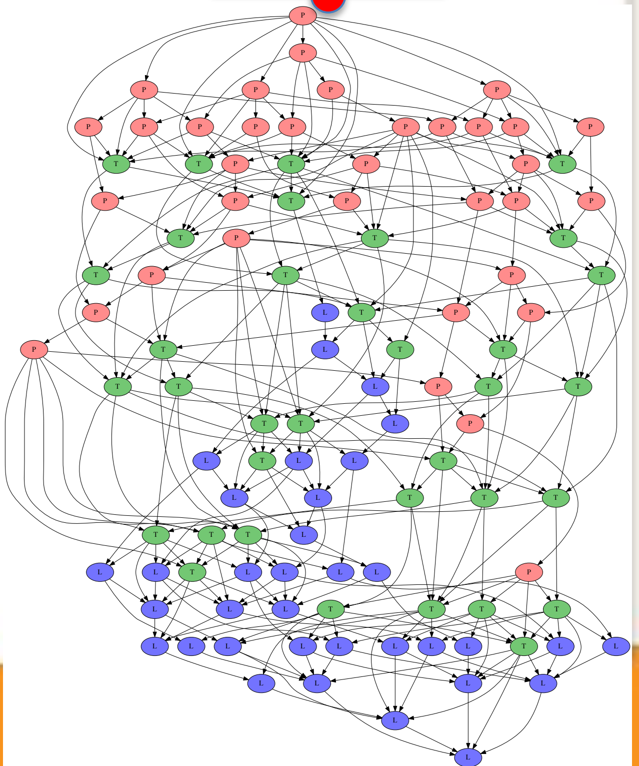
$$\int_0^1 (g(x) + f(x)) \times g(x) - g(f(x))dx$$

```
double A(const double x) { return std::exp(-x * x); }
double B(const double x) { return std::exp(-x * x) * std::cos(x); }

    auto p1 = make_project(&A, ctl, a, "project A");
    auto p2 = make_project(&B, ctl, b, "project B");

    auto d = make_diff(a, deriva, "dA/dx numeric");

    auto b1 = make_binary_op(add, a, b, a_plus_b, "a+b");

    auto b1 = make_binary_op(add, deriva, a_plus_b, b_minus_a, "b-a");
```

No synchronizations between the different algorithms, the data flows from one to another as soon as it becomes available.

Need for termination detection in the runtime



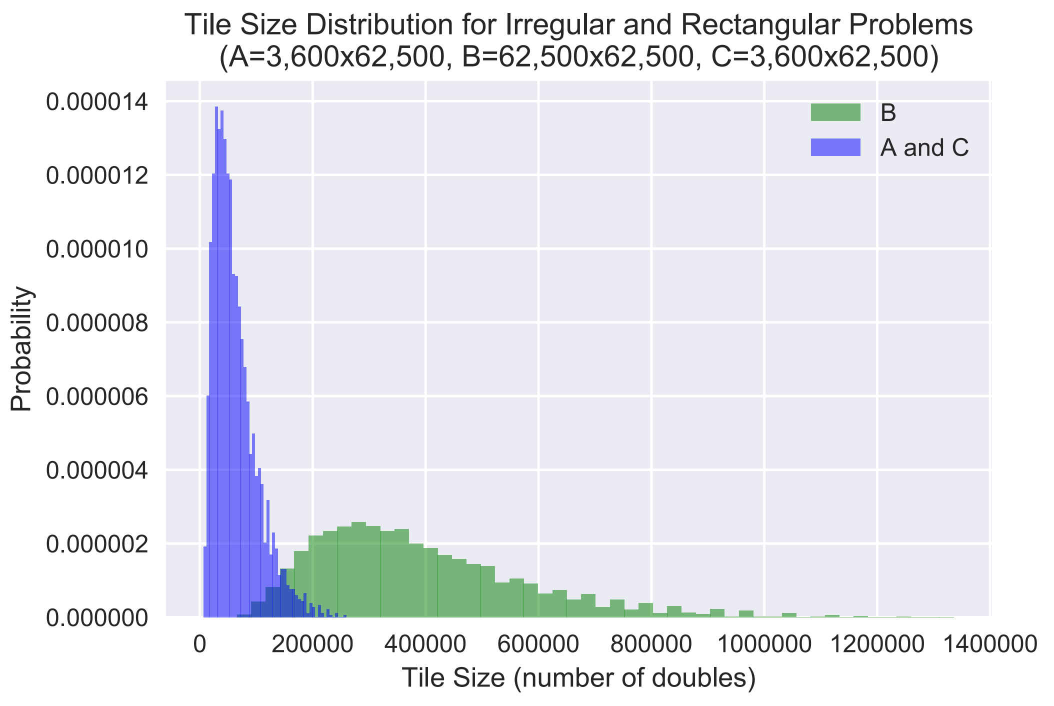


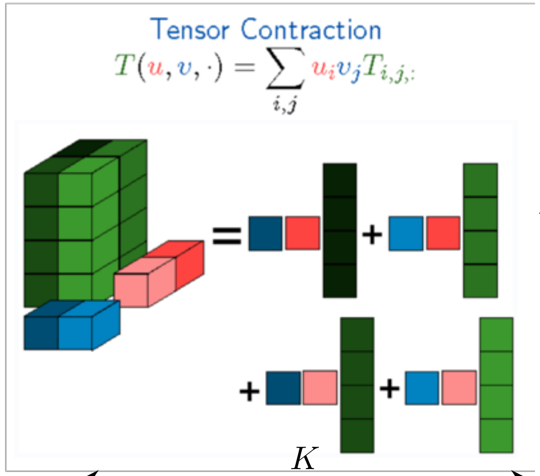
TESSE: Irregular tensor contraction

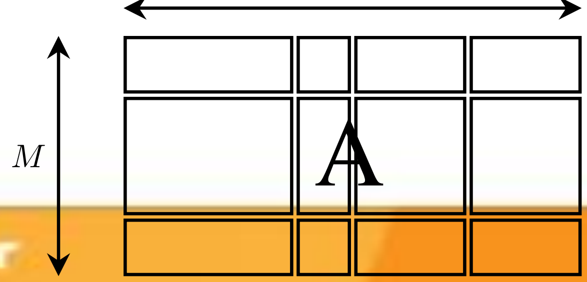
Accurate simulation of the electronic structure of molecules and solids using Coupled Cluster Singles and Doubles method (CCSD). Novel formulations replace the usual dense tensors with block-sparse and/or block-rank-sparse tensors increasing applicability from dozen to thousands of atoms.

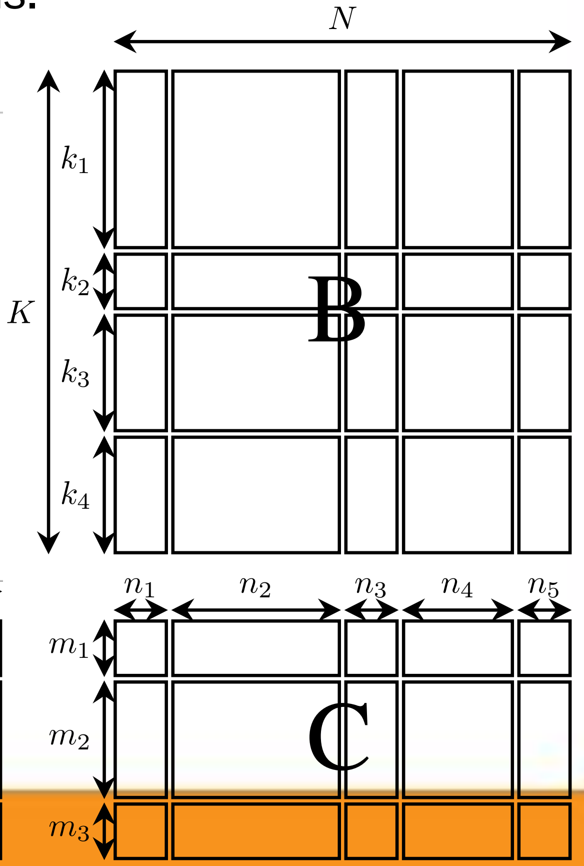
$$R^{ij}_{ab} = \sum_{cd} T^{ij}_{cd} G^{cd}_{ab} + \dots,$$

Application dominated (90% of execution time) by 4-index block-distributed tensor contractions. These tensor operations can be mapped to matrix-matrix multiplications with irregular and imbalanced tiling.

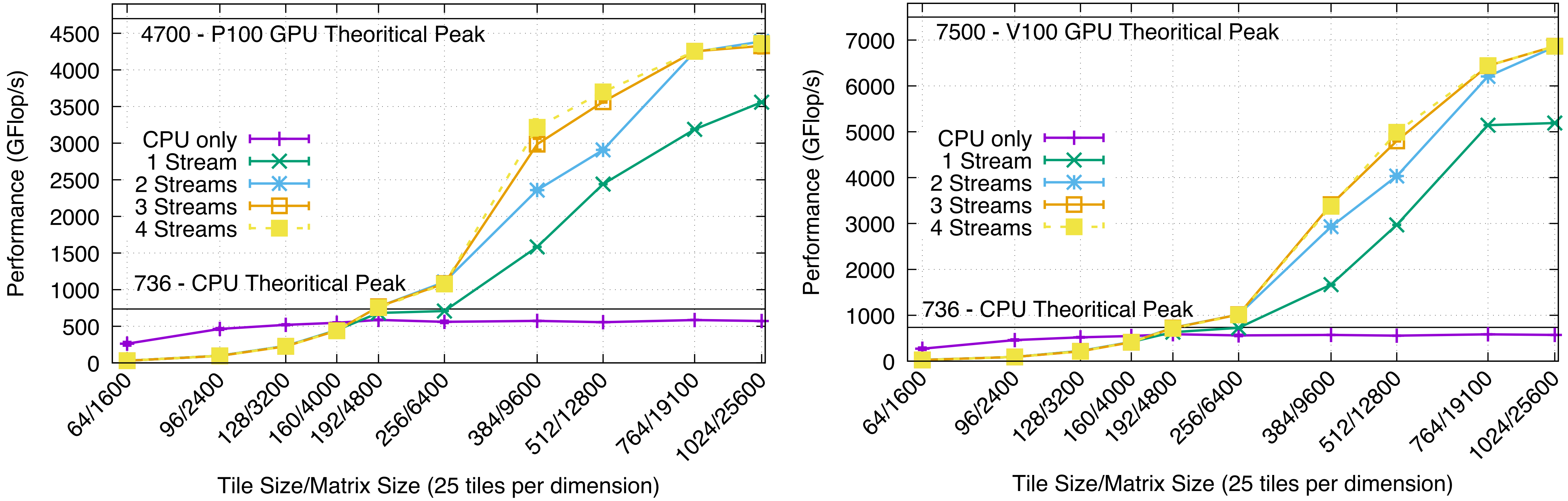








Irregular tensor contraction on Nvidia GPU



2 streams are dedicated to memory transfer (up and down), we study the impact of the computing streams

- 1 stream serializes calls, it will asymptotically reach peak;
- 2 streams already give enough performance by overlapping kernel calls with another kernel;
- 3 streams is enough in practice;
- 4+ might not [yet] be necessary.

Irregular tensor contraction

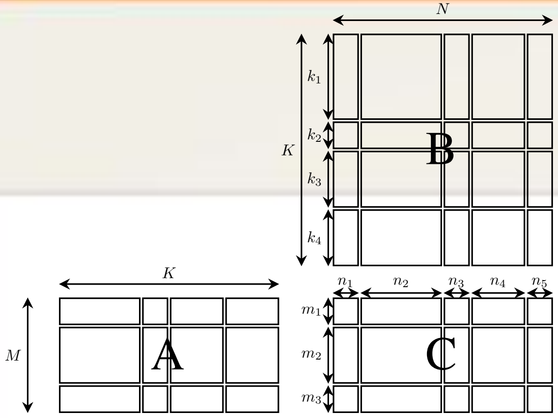
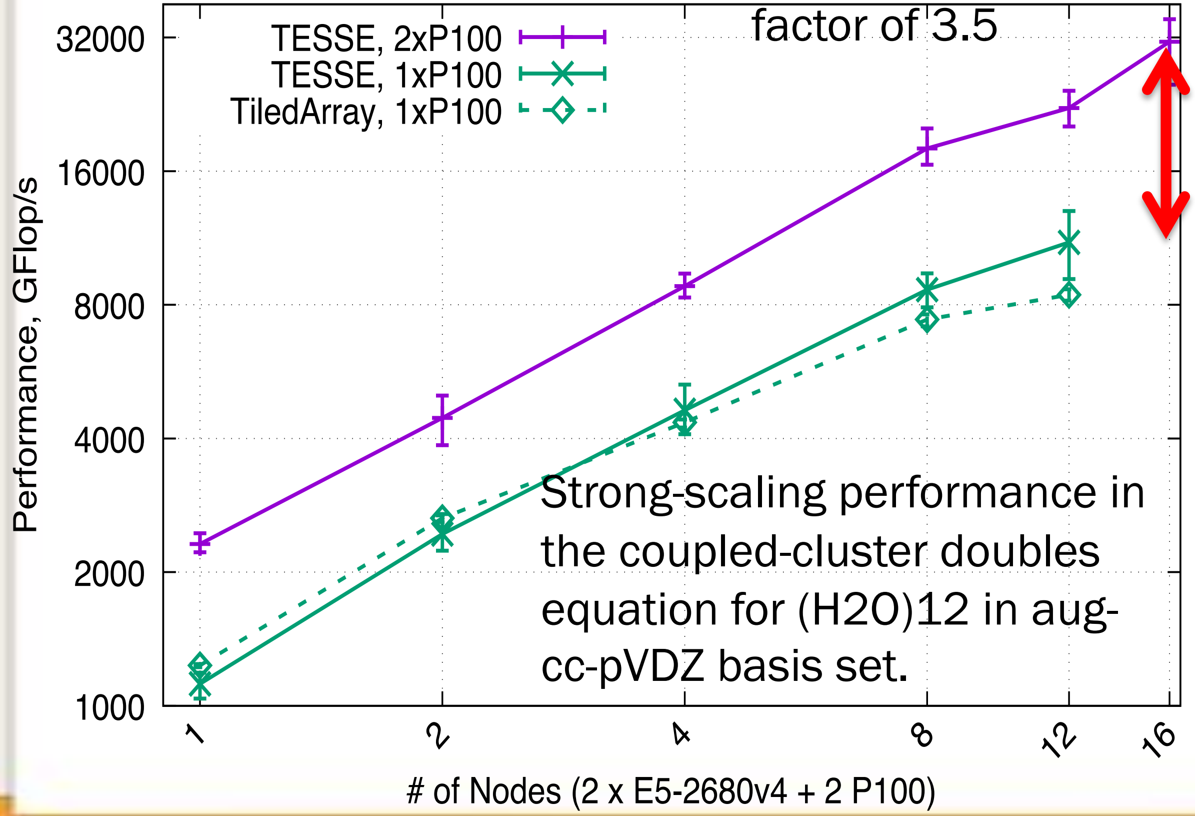
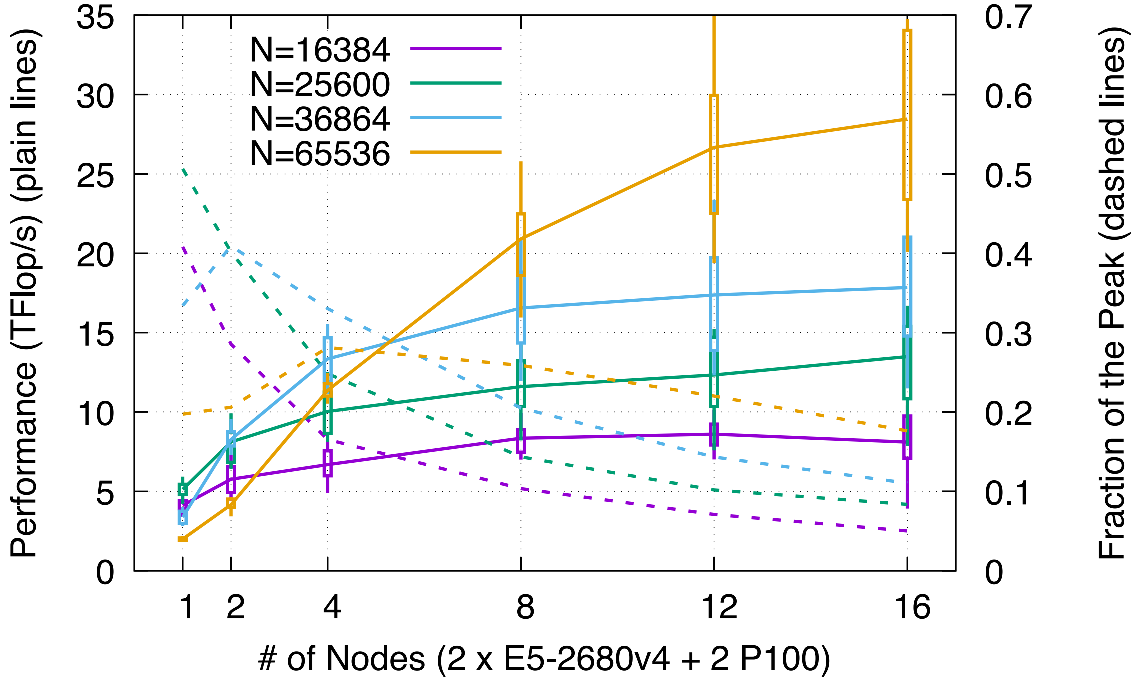


Fig. 1. Example of a Irregular Tiled GEMM operation

Success story: Time to solution reduced by a factor of 3.5



Strong-scaling performance in the coupled-cluster doubles equation for (H2O)12 in aug-cc-pVDZ basis set.

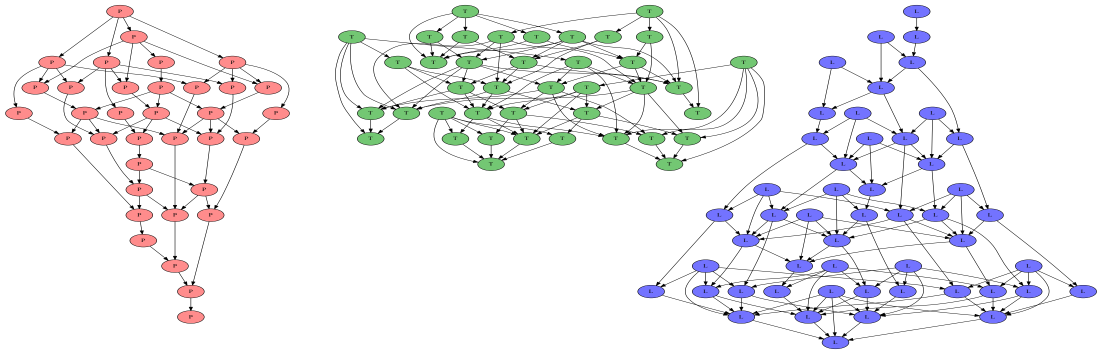


The less appealing story is that despite the significant reduction in time to solution, we only reach 20% of the hardware peak for irregular problems when we are able to reach 60% regular GEMM.
At N = 64k each node holds 34GB of data, 3 times more than the GPU memory. Memory traffic between nodes (IB EDR 100Gbs) is the main culprit.

Natural data-dependent DAG Composition

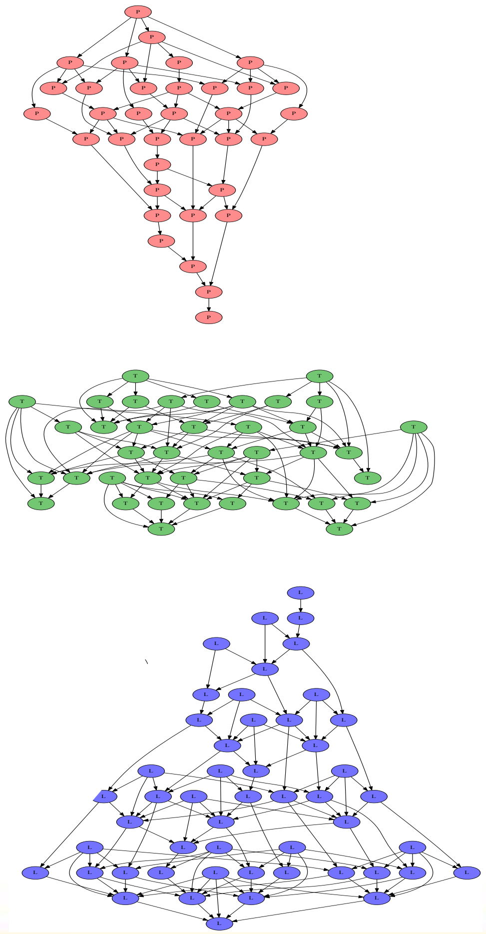
Example POTRI = POTRF + TRTRI + LAUUM

POTRF TRTRI LAUUM

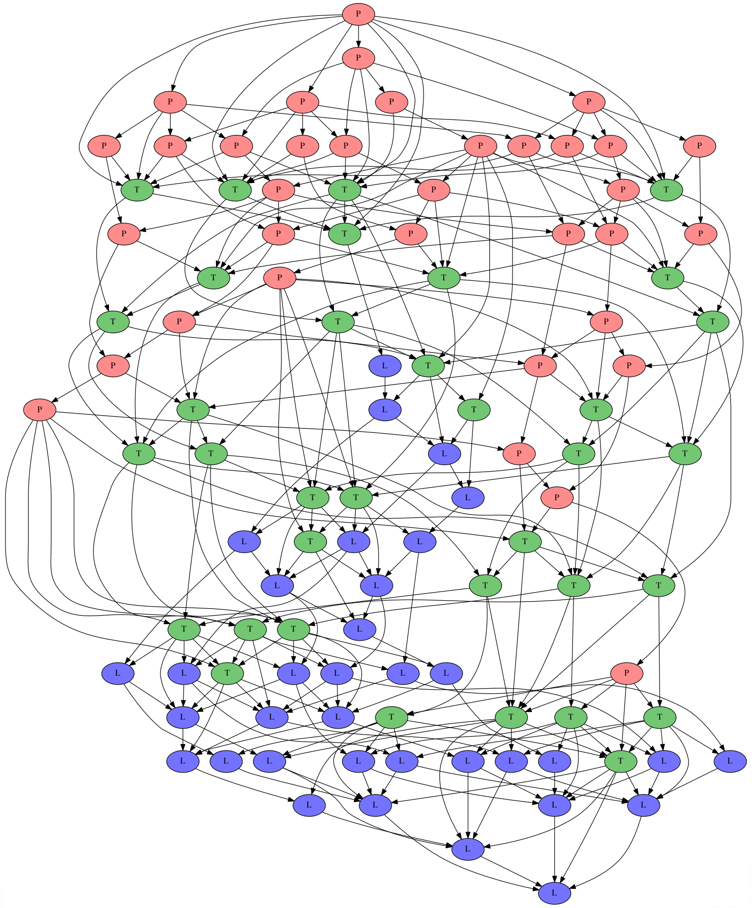


- 3 approaches:
  - **Fork/join:** complete POTRF before starting TRTRI
  - **Compiler-based:** give the three sequential algorithms to the Q2J compiler, and get a single PTG for POINV
  - **Runtime-based:** tell the runtime that after POTRF is done on a tile, TRTRI can start, and let the runtime compose

Traditional



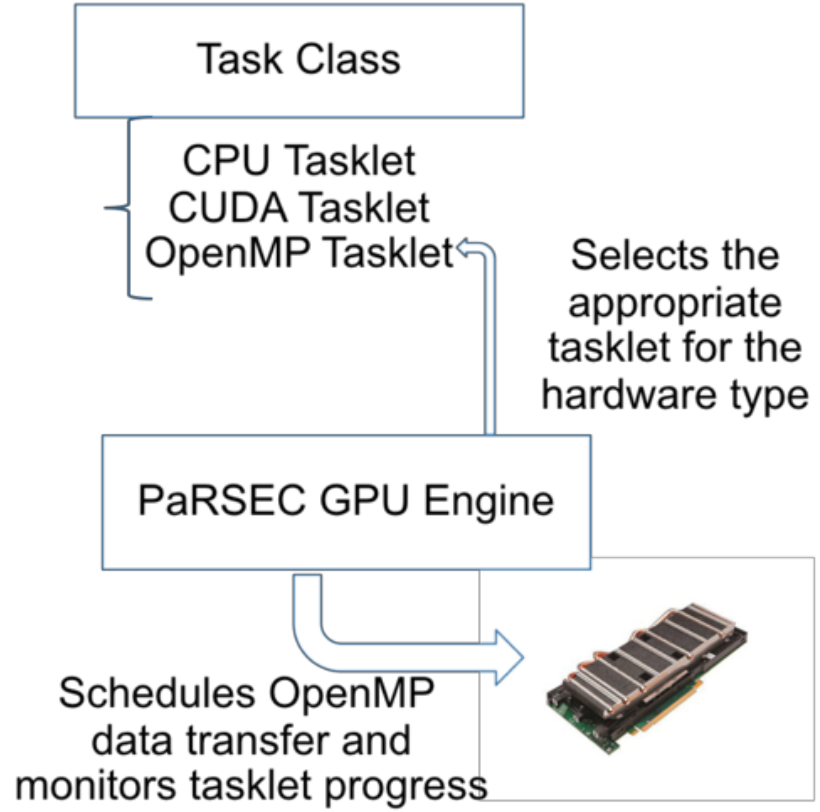
PaRSEC



Interoperability with other programming paradigms

- With OpenMP accelerator target
  - Goal: improve PaRSEC portability by supporting OpenMP accelerators
  - GPU Engine modified to use OpenMP target data movement directives/functions (i.e. support for non-CUDA devices)
  - Data movement and management remains implicit from the end-user (simplified programming)
  - User provides an OpenMP target task that will be scheduled by PaRSEC
- With Kokkos tasks
  - User provides a Kokkos task that will be scheduled by PaRSEC
    - No tracking data dependencies at this point
  - Proof-of-concept demonstrator with C to C++ translation shim
- With MPI programs
  - PaRSEC Communication insulated from application MPI communication
  - MPI programs can enter/exit PaRSEC sections
  - PaRSEC does not consume resources when idle

PaRSEC can schedule data transfers to an OpenMP accelerator and schedule OpenMP target task

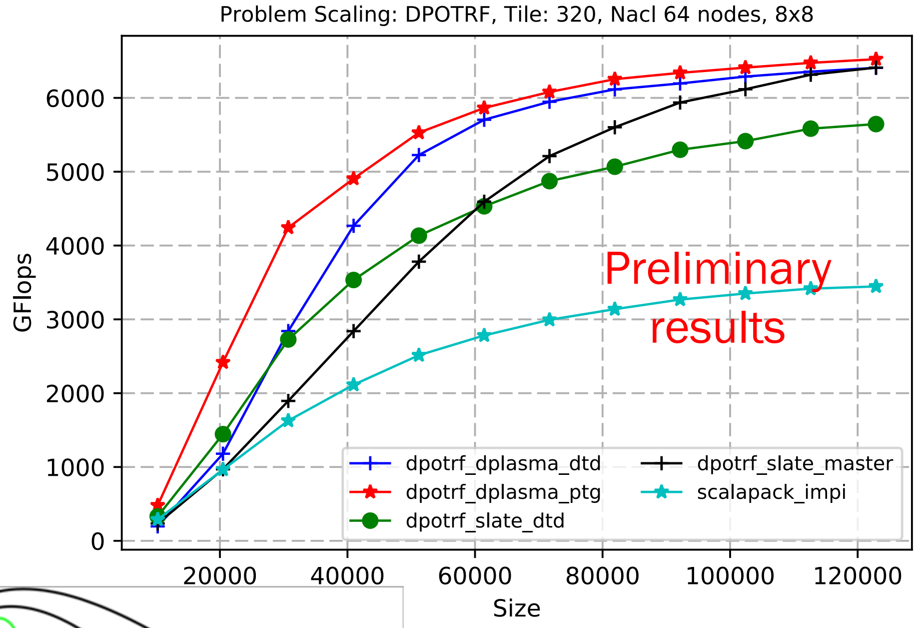


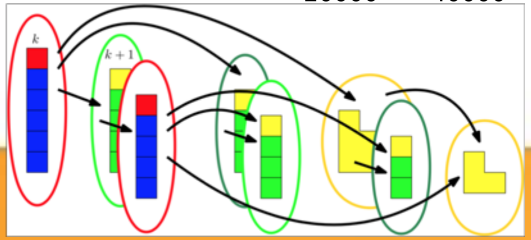
ECP collaboration SLATE and Exa-PAPI

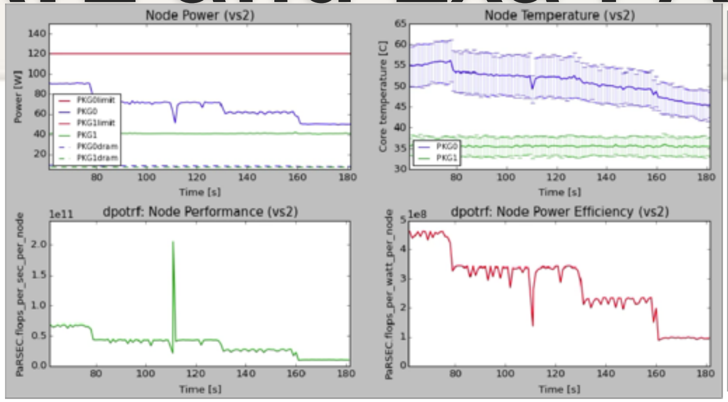
Providing support for SLATE C++ DSL/classes to unfold tasks over PaRSEC. Minimize the number of known tasks, explicit data collective patterns, batches executions, accelerator support.

Enhancing the runtime capabilities:

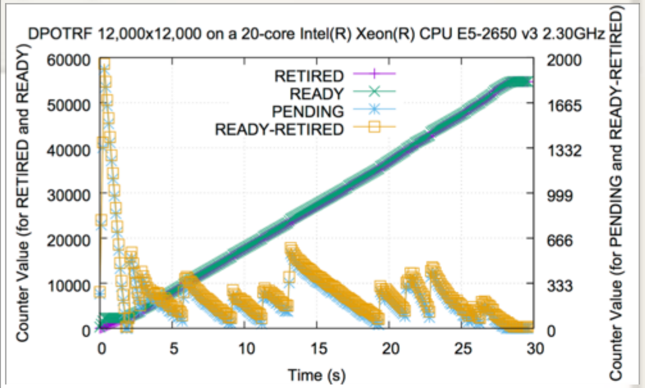
- mechanism for asynchronous completion of taskpools;
- multi-level task insertion to mitigate the overhead of dependencies resolution and enable the early detection of batched operations;
- API for explicit communication, type multicast;







Energy consumption and performance during a dynamic execution where the number of computations resources is reduced

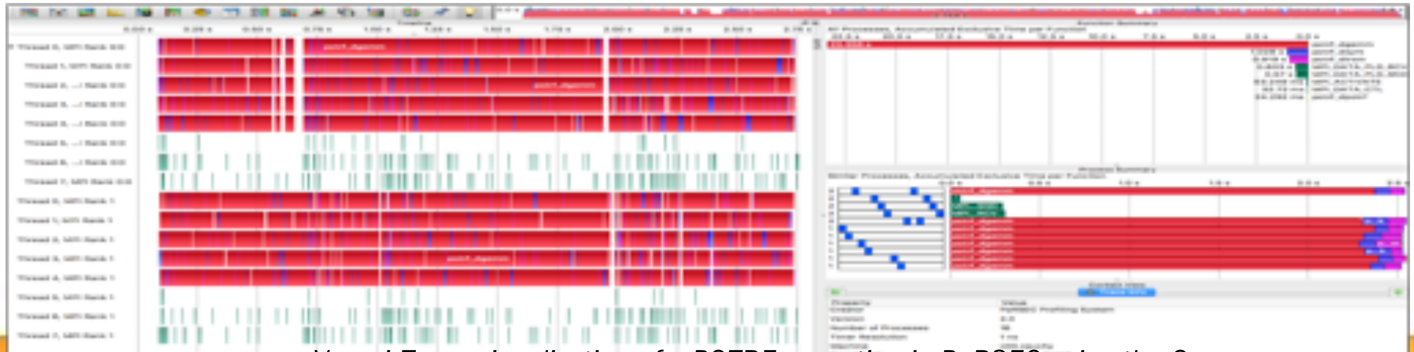


Evolution of some PaRSEC PAPI-SDE events during a POTRF factorization

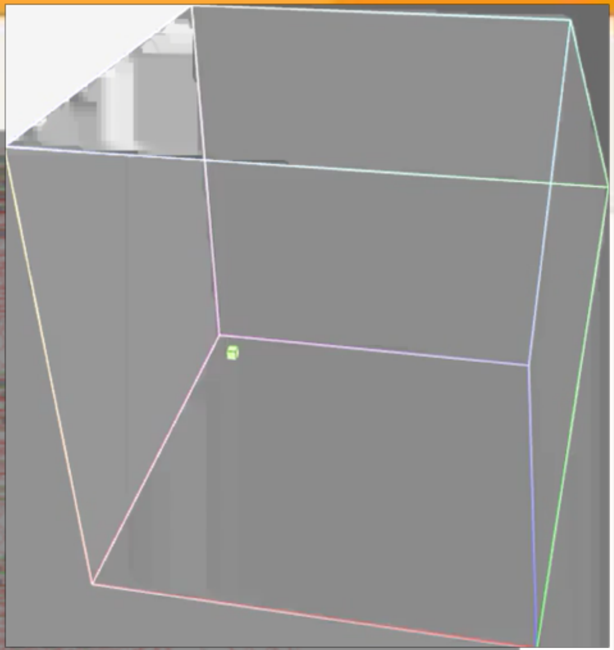
- ECP Collaboration with Exa-PAPI: integration of PAPI-SDE interface into PaRSEC
- PaRSEC presents internal events as PAPI counters for external tools (e.g. tau, ScoreP)
- Counters exposed:
  - Number of pending tasks (in different schedulers), ready tasks, retired tasks
  - Memory usage by internal systems (communication, task, ...)
  - Extend the DSL to provide application/library level counters
- All counters are lock-free / wait-free / atomic-free and

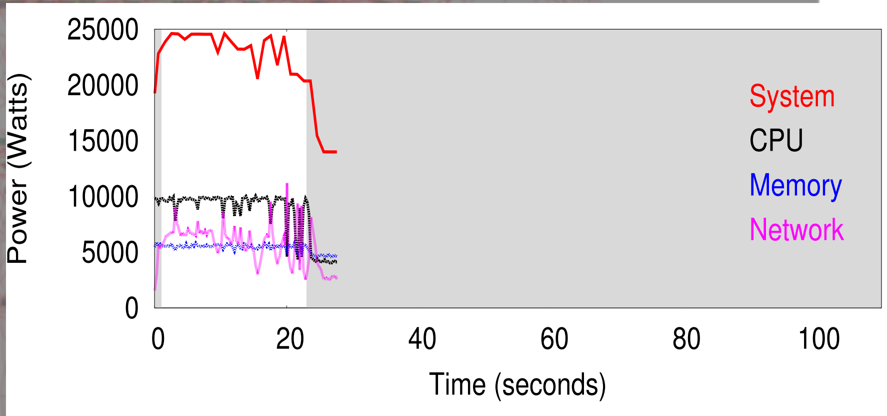
The PaRSEC ecosystem

- 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, SLATE
- A set of tools to understand performance, profile and debug
- A resilient distributed heterogeneous moldable runtime



VampirTrace visualization of a POTRF execution in PaRSEC using the Open Trace Format traces (OTF2)





(b) DPLASMA.

Conclusions

- Programming can be made easy(ier)
  - Portability: inherently take advantage of all hardware capabilities
  - Efficiency: deliver the best performance on several families of algorithms
  - Domain Specific Languages to facilitate development
  - Interoperability: data is the centric piece
- 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
- Interact with hardware designers to improve support for runtime needs
  - HiHAT: A New Way Forward
    for Hierarchical Heterogeneous Asynchronous Tasking