PaRSEC: Distributed task-based runtime for scalable hybrid applications

https://bitbucket.org/icldistcomp/parsec

HiHat Meeting, March. 16, 2017

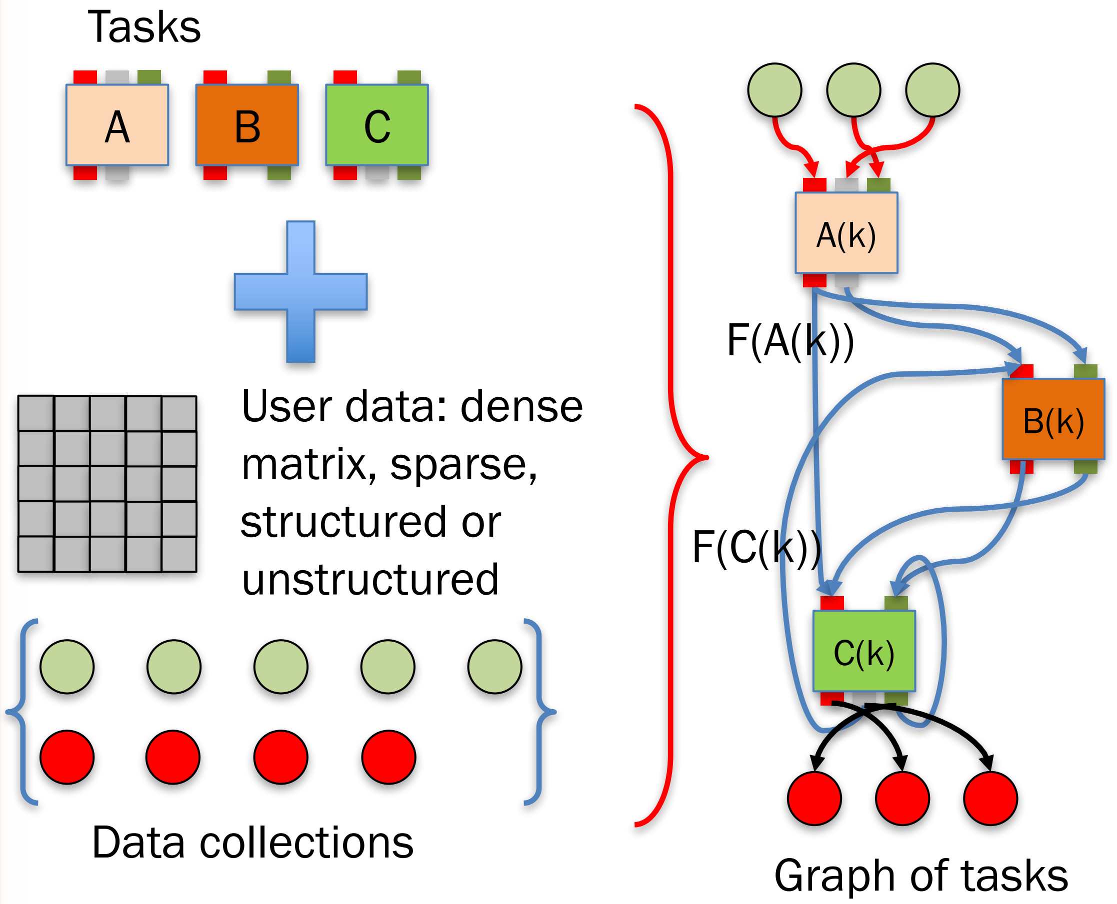
ICL UT INNOVATIVE COMPUTING LABORATORY THE UNIVERSITY of TENNESSEE

IN COLLABORATION WITH Inria

WITH SUPPORT FROM Microsoft

SPONSORED BY U.S. DEPARTMENT OF ENERGY, National Science Foundation

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.
- Tasks and data have a coherent distributed scope (managed by the runtime)
- Low-level API allowing the design of Domain Specific Languages (JDF, DTD, TTG)
- Supports distributed (aka. the runtime moves data) heterogeneous (and trigger tasks execution) environment.
  - Built-in resilience, performance instrumentation and analysis (R, python)

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)

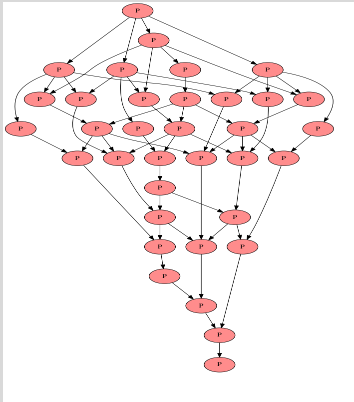
```
parsec_context_t* dague;
parsec = parsec_context_init(NULL, NULL);  /* start the PaRSEC engine */
```

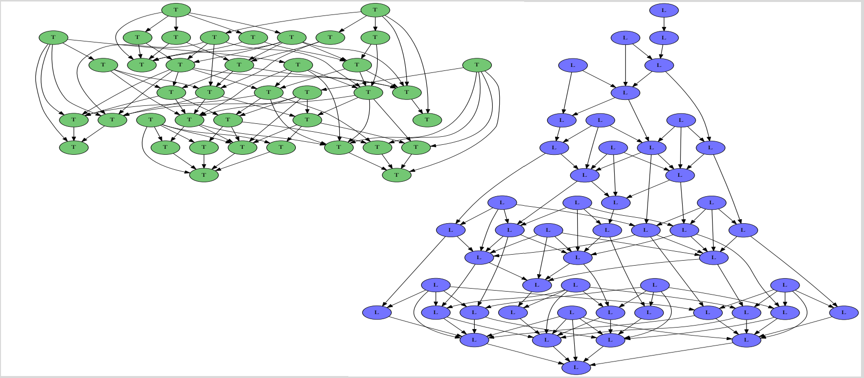
Create a tasks placeholder and associate it with the PaRSEC context

```
parsec_handle_t* parsec_dtd_handle = parsec_handle_new (parsec);
parsec_enqueue(parsec, (parsec_handle_t*) parsec_dtd_handle);
```

```
parsec_context_start(parsec);
```

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





Wait 'till completion

```
parsec_handle_wait( parsec_dtd_handle );
```

Data initialization and PaRSEC context setup. Common to all DSL

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

Create a tasks placeholder and associate it with the PaRSEC context

```
parsec_handle_t* parsec_dtd_handle = parsec_handle_new (parsec);
parsec_enqueue(parsec, (parsec_handle_t*) parsec_dtd_handle);

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(
        parsec_dtd_handle,
        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(
        parsec_dtd_handle,
        call_to_kernel_type_read,    "Read_Data",
        PASSED_BY_REF,    DATA_AT(&dDATA, n),    INPUT | REGION_FULL,
        0  /* DONE */ );
    }
}
```

Wait 'till completion

```
parsec_handle_wait( parsec_dtd_handle );
```

Data initialization and PaRSEC context setup. Common to all DSL

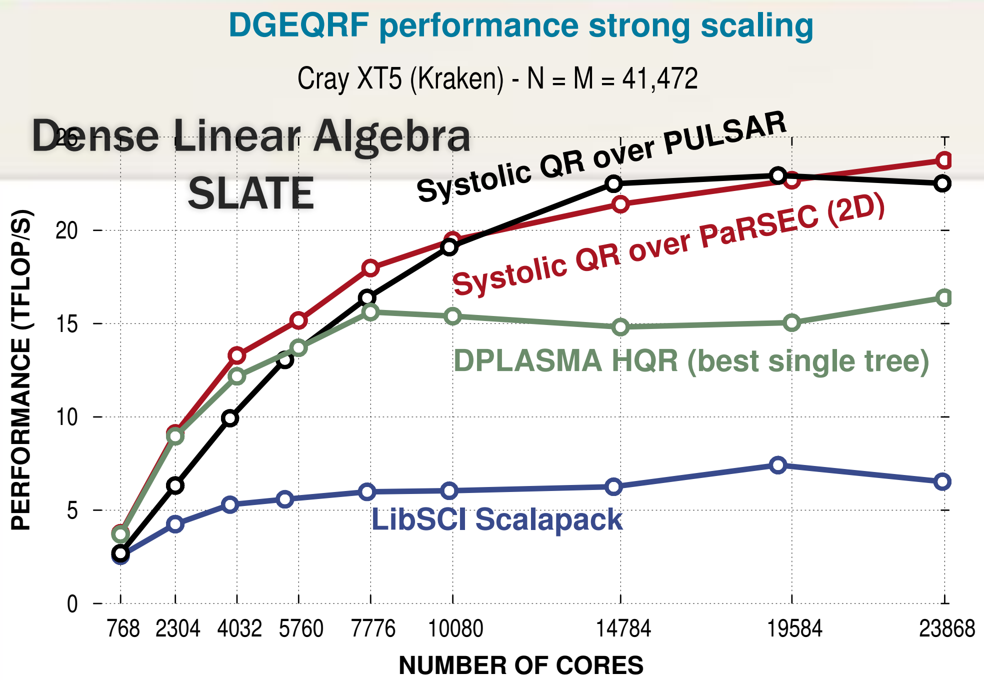
The Parameterized Task Graph (JDF)

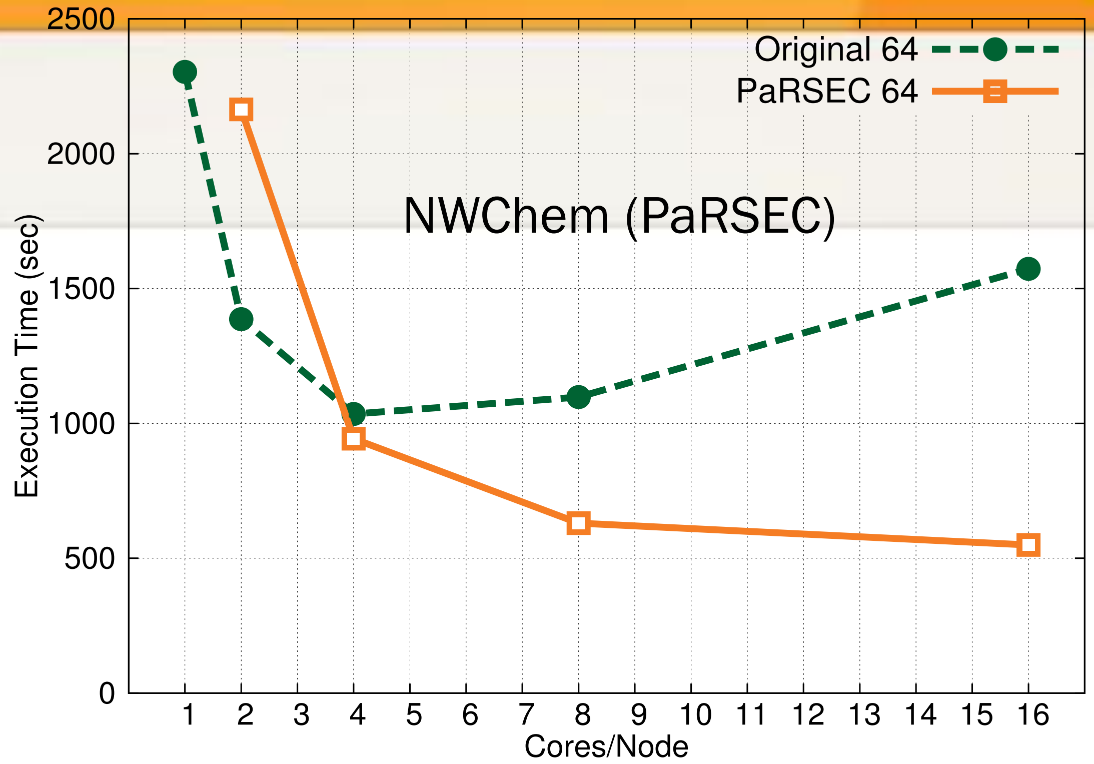
```
GEQRT(k)

 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
```

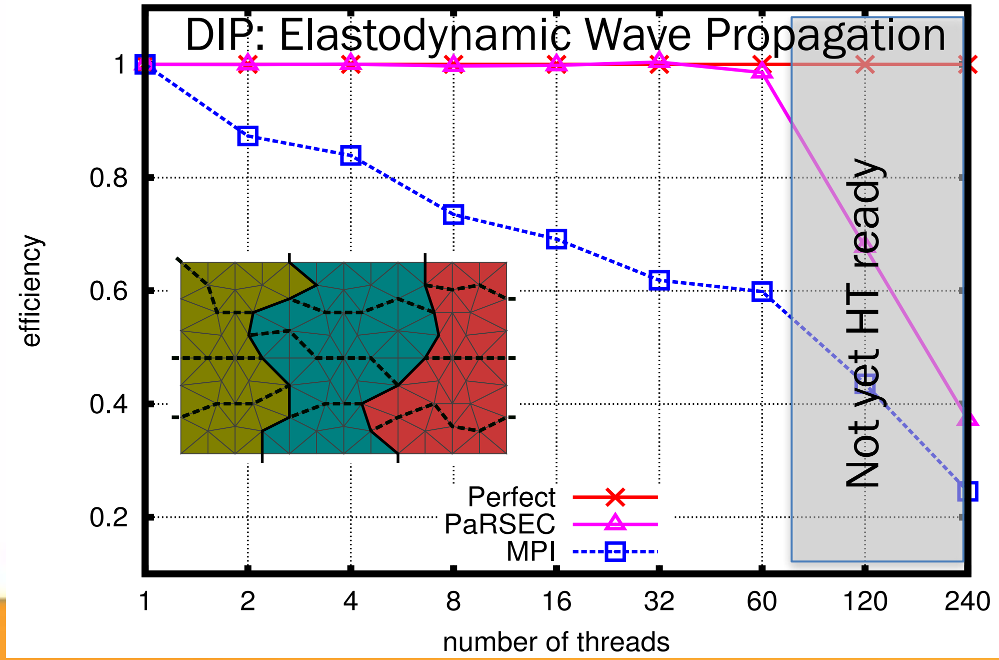
Control flow is eliminated, therefore maximum parallelism is possible

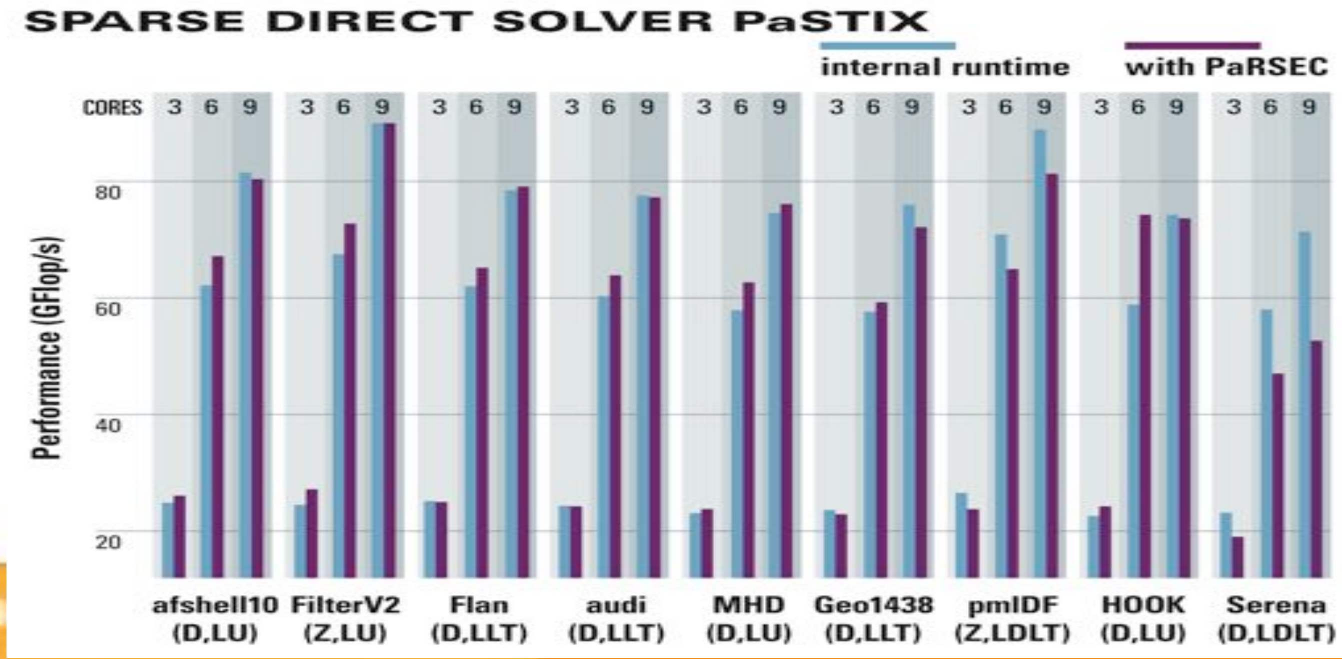
- A dataflow parameterized and concise language
- Accept non-dense iterators
- Allow inlined C/C++ code to augment the language [any expression]

- 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
  - No dynamic DAGs
  - No data dependent DAGs

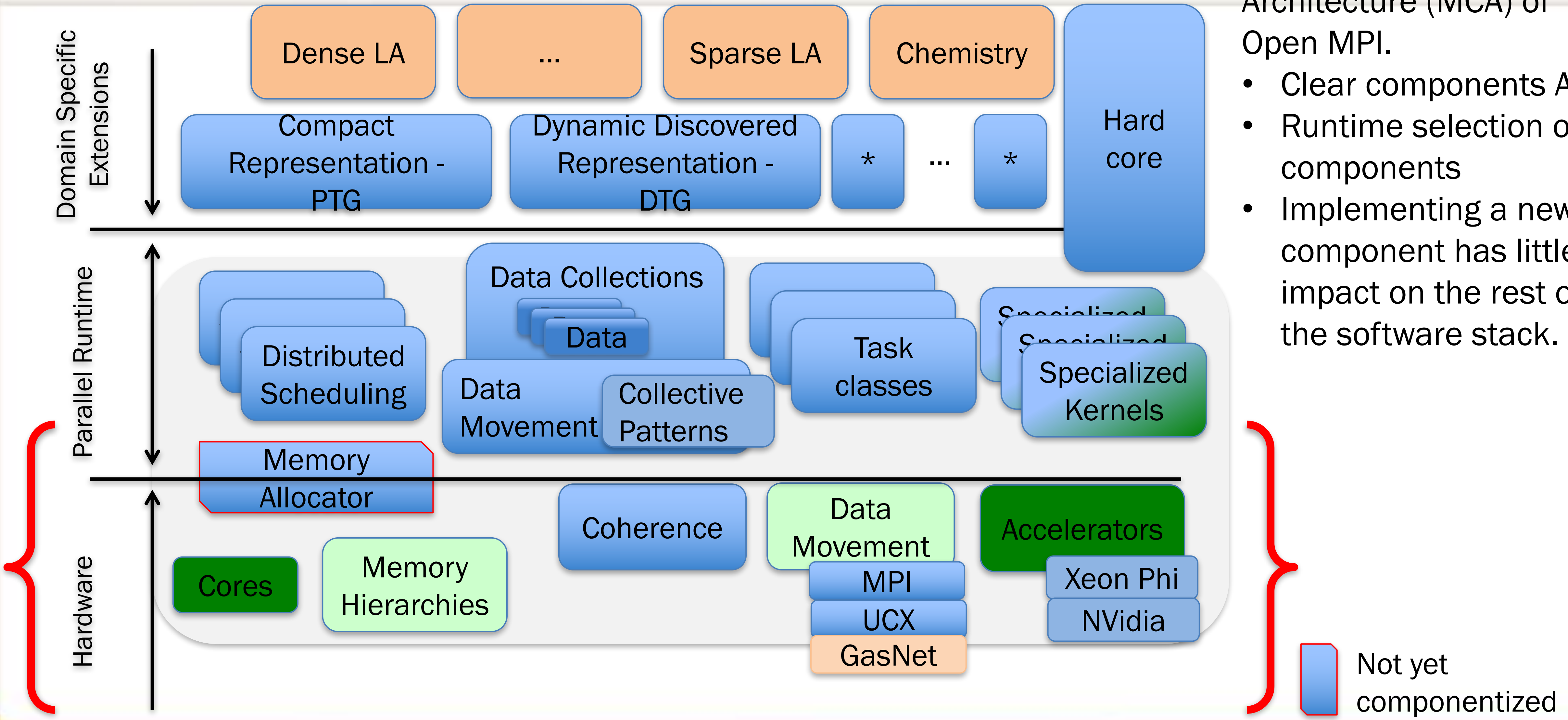

DGEQRF performance strong scaling
Cray XT5 (Kraken) - N = M = 41,472
Dense Linear Algebra
SLATE
Systolic QR over PULSAR
Systolic QR over PaRSEC (2D)
DPLASMA HQR (best single tree)
LibSCI Scalapack
PERFORMANCE (TFLOP/S)
0
5
10
15
20
768 2304 4032 5760 7776 10080 14784 19584 23868
NUMBER OF CORES


NWChem (PaRSEC)
Original 64
PaRSEC 64
Execution Time (sec)
0
500
1000
1500
2000
2500
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Cores/Node


DIP: Elastodynamic Wave Propagation
Not yet HT ready
Perfect
PaRSEC
MPI
efficiency
0.2
0.4
0.6
0.8
1
1 2 4 8 16 32 60 120 240
number of threads


SPARSE DIRECT SOLVER PaSTIX
internal runtime
with PaRSEC
CORES 3 6 9
Performance (GFlop/s)
20
40
60
80
afshell10 (D,LU)
FilterV2 (Z,LU)
Flan (D,LLT)
audi (D,LLT)
MHD (D,LU)
Geo1438 (D,LLT)
pmlDF (Z,LDLT)
HOOK (D,LU)
Serena (D,LDLT)

PaRSEC Architecture



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

- Clear components API
- Runtime selection of components
- Implementing a new component has little impact on the rest of the software stack.

Wish List (1/2)

- Never block, fail gracefully and provide support for reissuing the operation
- Architectural Information (currently HWLOC)
  - This info is used for thread placement (communication thread, accelerator managing threads), scheduling
  - Hyper-thread management (used by the runtime for low computational tasks) or gives to the tasks
  - Energy ?
- Data Management
  - Datatypes (memory layouts)
    - Regular (vector/index) and irregular (struct)
    - MPI is providing a good interface
  - Data versioning and coherency (software MOESI)
  - Efficient in intra and inter-node mode
  - Memory allocator (pinned memory, arena, NUMA placement, …)
  - More flexible memory management: expose page-tables to the runtime would be interesting, global pinned memory (that can be used with all hardware devices)
- Accelerators management
  - Abstract portable interface (with or without support for datatypes) for data transfers
  - Tasks submission
  - Serialization per stream/flow provide certain benefits, but required a infallible scheduling (unexpectedly NP-hard)

Wish List (2/2)

- Communication layer
  - Event based system
    - Short messages for coordination
    - RMA for large data transfers
    - Active Message offers a desirable interface (but gray areas still exists)
      - Threading support ?
      - What level of portability (MPI) ?
      - How do we interoperate with 1) legacy applications (assuming libraries developed using PaRSEC); 2) other runtimes ?
    - Collective pattern (different than most other implementations as async and non-blocking)
- Resilience
  - Detecting data corruption. Who detects (hardware or software) and who/how to propagates ?
    - A task-based runtime has more information about what task used what memory …
  - A more flexible and efficient mechanism that async signals
  - Serialized direct access to stable storage (NVRAM, local SSD, buddy, …)

Priority based Wish List

- Portable and efficient API/library for accelerator support
- Portable, efficient and interoperable communication library (UCX, libFabric, …)
- Moving away from MPI will require an efficient datatype engine
  - Also supported by rest of the software stack (for interoperability)
- Resource management/allocation system
  - PaRSEC supports dynamic resource provisioning, but we need a portable system to bridge the gap between different programming runtimes
- Memory allocator: thread safe, runtime defined properties, arenas (with and without sbrk). (memkind?)
- Generic profiling system, tools integration
- Any type of task-based debugger and performance analysis

Q & A