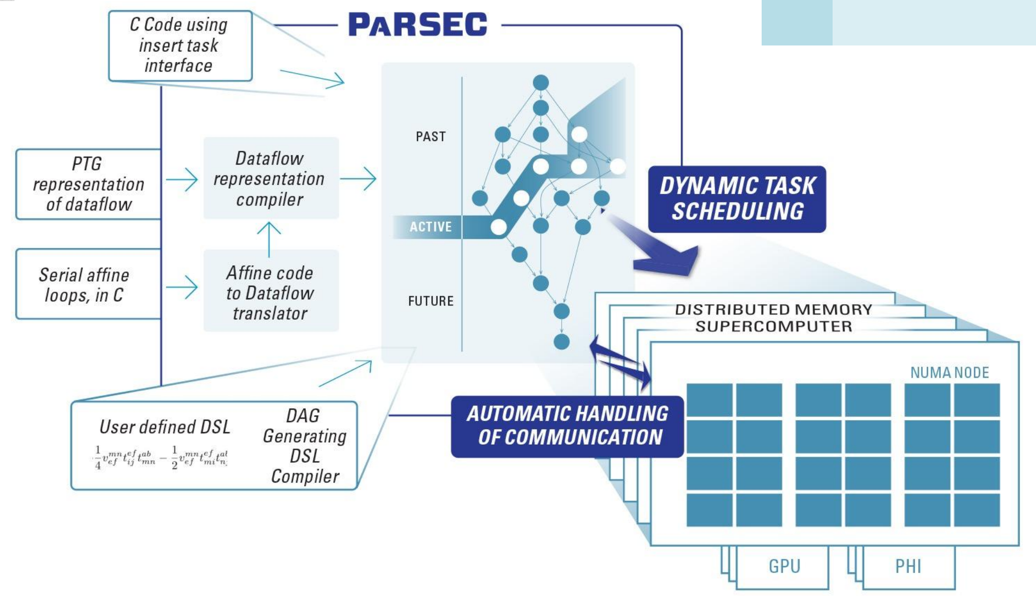
DSLs and APIs for Dataflow Programming (over the PaRSEC runtime)

Thomas Herault, Joseph Schuchart, George Bosilca, Robert Harrison, Ed Valeev, Poornima Nookala, et al

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

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



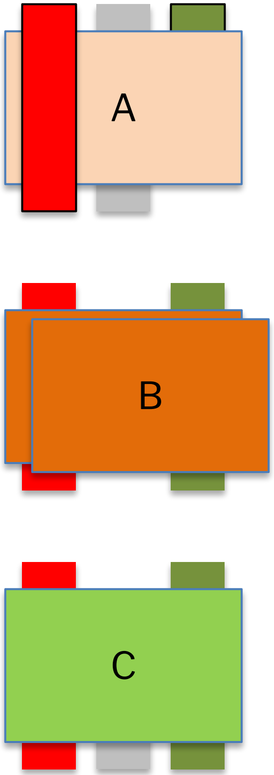
Concepts

- 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/TTG, Python, insert_task, fork/join, …)
- Separate algorithms from data distribution
- Remove unnecessary control flow

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

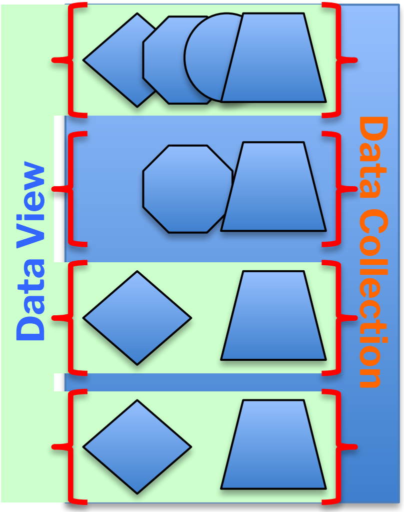
PaRSEC concepts: Tasks / Collections / Contexts



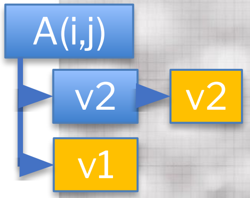
- A task-class is somewhat a familiar concept, a pure function with a well-defined number of terminals (input and outputs)
  - Terminals are tagged with properties R/RW/W/T
  - Depending on the DSL the outputs might be made available at any time
  - Task-classes can be extended with multiple incarnations (CPU, GPU, hierarch, OpenCL, JIT, …)
  - The execution device is dynamically selected at runtime among available incarnations
  - Specialized terminals exists (IO, redistributed, compress, low-rank, push/pull, validate)

- A task is a particular instance of a task-class (i.e. a task class with a unique task identifier)
  - The runtime was designed for tasks with ~10 μsec granularity
  - A collection of tasks and their dependencies is a taskpool
    - DSLs generate or populate taskpools

PaRSEC concepts: Tasks / Collections / Contexts



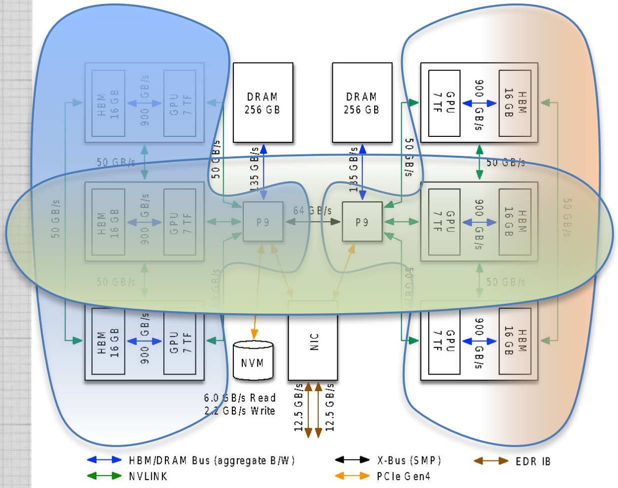
- A data is the basic logical element used in the description of the dataflow
  - Locations: have multiple coherent copies (remote node, device, checkpoint)
  - Shape: can have different memory layout
  - Visibility: only accessible via the most current version of the data
  - State: can be migrated / 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
  - Can be virtual (no content),
- **Data View** a subset of the data collection used in a particular algorithm (aka. submatrix, row, column,…)
- A data (version) is a promise, a data collection is a promise, a data view is a promise
- The promise will be delivered where it is expected by the task that will use it (distributed, GPU task on GPU, …)



INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

PaRSEC concepts: Tasks / Collections / Contexts



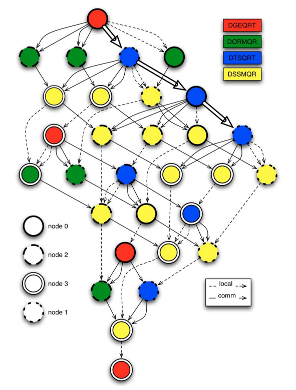
- A PaRSEC context is a distributed executor extended with a set of resources (core(s), accelerators, networks), memory allocators, and task schedulers
  - Multiple contexts could exist simultaneously, but the runtime does not police their use of resources
  - A given taskpool belongs to a specific context, and its tasks execute only on the resources belonging to the context

PaRSEC: task-based runtime system

- PaRSEC:
  - A runtime system
    - Distributed, accelerated, with multiple communication systems
  - A programming environment
    - Tools for profiling, debugging
  - A set of Domain Specific Languages / Extensions
    - Dynamic Task Discovery (DTD)
    - Parameterized Task Graph (PTG)
    - (SLATE API)
    - *Templated Task Graph (TTG)*

From the PaRSEC runtime perspective

- The runtime is agnostic to the domain specific language (DSL)
- Different DSL interoperate through the data collections
- The DSL share the infrastructure
  - Distributed schedulers
  - Communication engine
  - Hardware resources
  - Data management (coherence, versioning, …)
- They don’t share
  - The task structure
  - The internal dataflow depiction



Dynamic Task Discovery (DTD) aka. insert_task

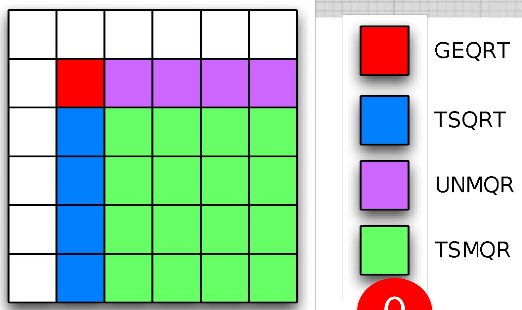
```
int task_hello(parsec_execution_stream_t *es,
               parsec_task_t *this_task)
{
  int *i;
  parsec_dtd_unpack_args( this_task, UNPACK_VALUE, &i);
  printf("Hello World, my index is %d\n", *i);
  return PARSEC_HOOK_RETURN_DONE;
}
int discover_tasks()
{
  for(int i = 0; i < 10; i++) {
    parsec_dtd_taskpool_insert_task( dtd_tp, task_hello,
                                     0, "hellow_world_task",
                                     sizeof(int), &i, VALUE,
                                     0); /* No more arguments
*/
  }
```

- Possible for each process to only discover local tasks, but data consistency **must** be maintained globally
- Data versioning and caching become a requirement
- Difficult to identify **collective patterns**
- Selecting the **window size** is difficult, all data movement must be known globally (and their order is critically important)

- Dynamic Task Discovery (DTD) enables simple DAG expression through sequential task discovery
- PaRSEC DTD engine builds the DAG of tasks, based on the dependencies of the data flow
- The semantics of sequential execution (the algorithm critical path) are enforced while keeping a DAG with maximal parallelism
- For distributed execution, all computing elements need to discover the same DAG, impairing the runtime scalability
- Only local tasks are kept, and a reference to last accessors / writers on given data to track remote dependencies
- The internal data structure representing the DAG is problem-size dependent, and task discovery window dependent

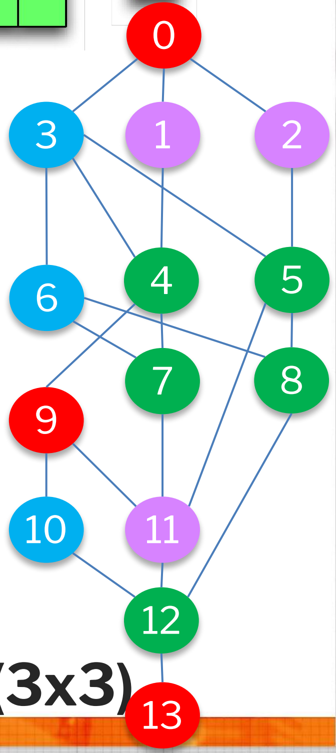
DTD: insert_task

```
for( k = 0; k < SIZE; k++ ) {
    parsec_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++ )
      parsec_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++ ) {
      parsec_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++ ) {
          parsec_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)
      }
    }
}
```



A

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2



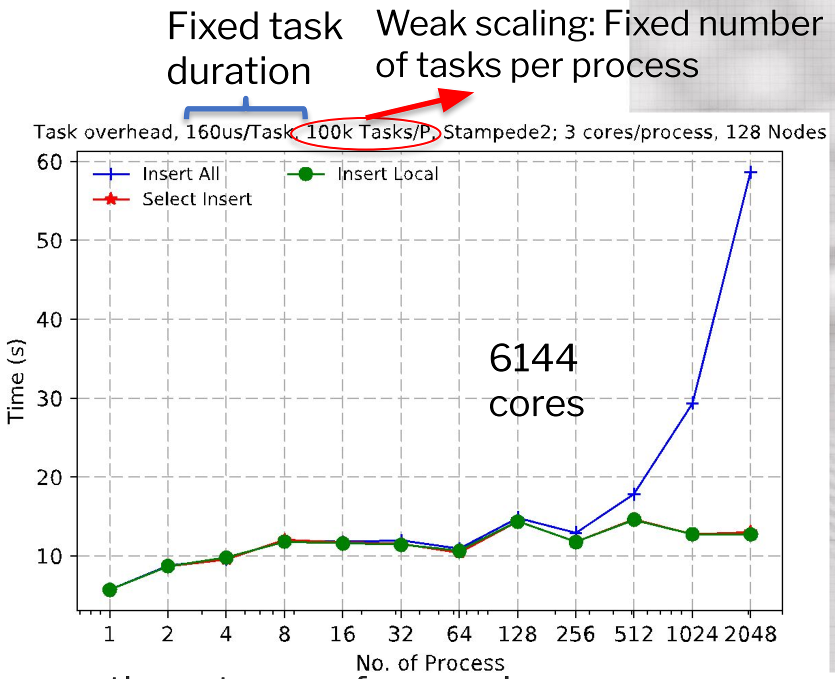
QR Factorization (3x3)

Challenge

- All participating nodes in distributed setting needs to discover the full task-graph (consistent view)
- DAG of large problem might not fit in memory

Solution: Partially Unrolling the DAG

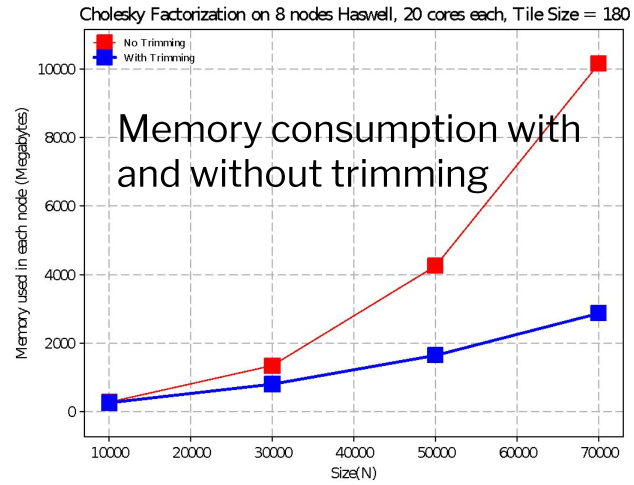
- Create partial DAG, progress, repeat (sliding window of DAG)
  - How the DAG is described directs the execution
- Memory usage is bound to the size of sliding window
- Size of window determines how far in future we can see both locally and remotely (affects performance)



- 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.

Other DTD optimizations

- Trimming (idea popularized by StarPU)
  - Removing remote tasks that do not have any impact locally
- Untying Task Insertion:
  - Users can insert task using one specific thread
  - Users can also insert task that can insert more tasks in the runtime, untying any specific thread from the responsibility of task insertion
  - Allow **recursive task insertion**
  - Allow users to generate independent tasks simultaneously
  - Eliminates performance drop in case of responsible thread being de-scheduled by OS
- Communication
  - Keep track of data version and cache them remotely to avoid sending the same version multiple times
    - What is the life expectancy of these remote copies ?
  - Recycle buffers to optimize memory usage
- PaRSEC Specific Extensions
  - Add collective communications, specialized tasks that operate on a variable number of data
  - Implement owner tracks uses – the opposite concept of tasks trimming



INNOVATIVE COMPUTING LABORATORY

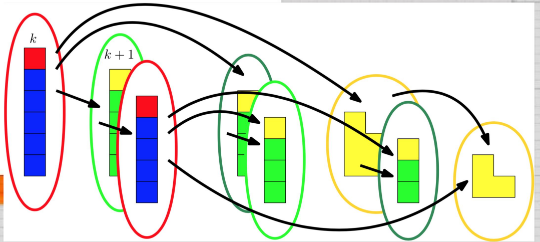
THE UNIVERSITY OF TENNESSEE KNOXVILLE

SLATE API (templated C++)

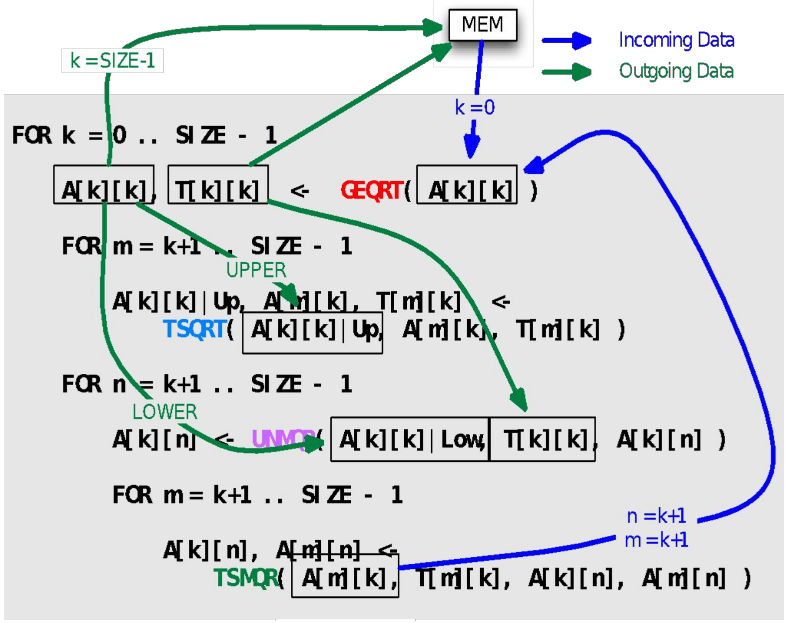
```
for (int64_t k = 0; k < A.nt(); ++k) {
  PaRSEC::potrf_panel
      <HermitianMatrix<scalar_t>, scalar_t>(A, k, A.column(k));

  if (k+1 < A.nt())
     PaRSEC::broadcast_column
       <HermitianMatrix<scalar_t>, scalar_t>
            (A, k, A.column(k), A.column(k+1), A.column(A.nt()-1));
  }
  for (int64_t n = k+1; n < k+1+lookahead && n < A.nt(); ++n) {
    // lookahead column(s)
    PaRSEC::potrf_lookahead
       <HermitianMatrix<scalar_t>, scalar_t>
         (A, k, n, A.column(k), A.column(n));
  }
  if (k+1+lookahead < A.nt()) { // trailing submatrix
    PaRSEC::potrf_trailing_update
       <HermitianMatrix<scalar_t>, scalar_t>
         (A, k, k+1+lookahead, A.column(k), A.column(k+1+lookahead),
          A.column(A.nt()-1));
  }
  PaRSEC::data_flush(A.parsec_high_level_tp,
                     A.column_range(k, k, A.nt()-1));
}
```

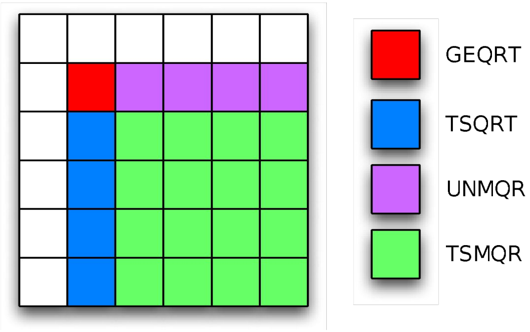
- The SLATE-ish API targets regular algorithms: tile-based task discovery algorithms with explicit synchronization and communications
- Use of templating to manage multiple precision and data representations
- Task discovery based on maintaining the sequential semantic
  - Computing elements need to discover only local tasks
- Communications and synchronizations are both implicit and explicit
- The language/API expresses a control flow
- Explicit communication happens within the progress of these containers and in the background.

ICL INNOVATIVE COMPUTING LABORATORY



Parameterized Task Graph (PTG)





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

Parameterized Task Graph (PTG)

```
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 possible but not necessary, maximum parallelism is exposed

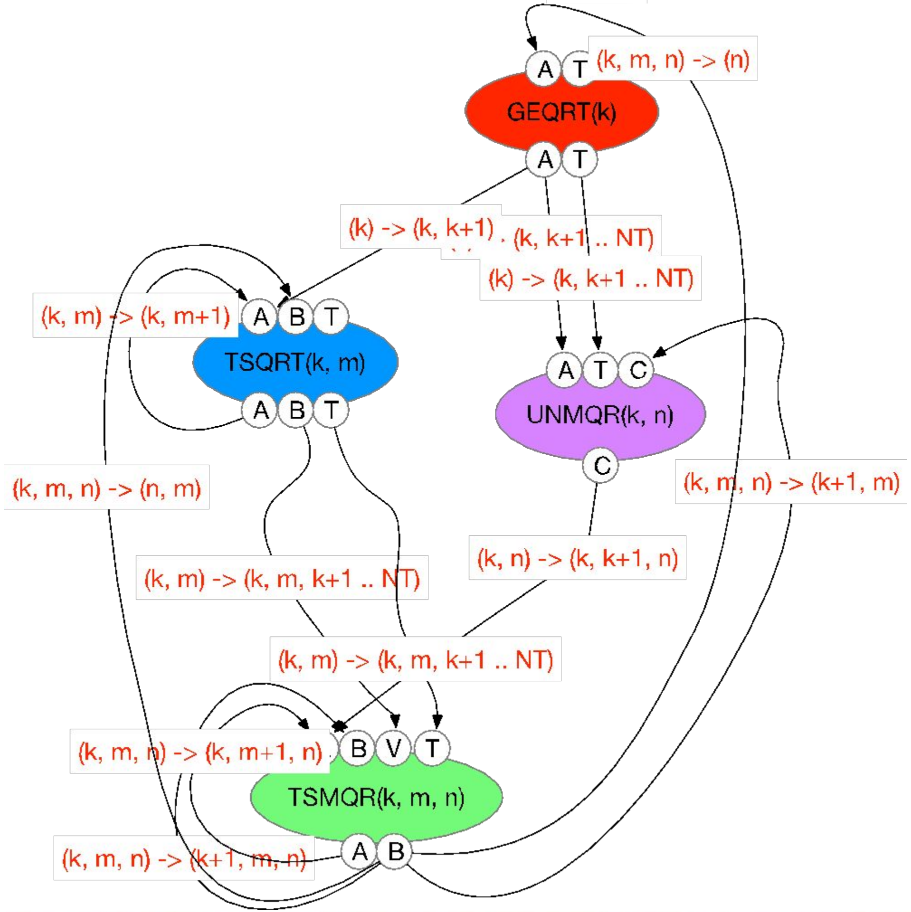
Data-dependent problems (where the DAG structure depends on the data itself) are more challenging

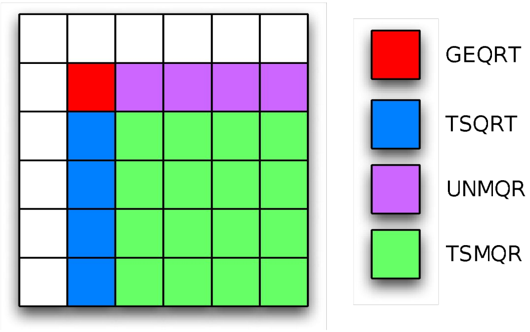
- A concise parameterized dataflow language, with non-dense iterators and extended expressions via inlined C/C++ code to augment the language
- Only local tasks are instantiated: internal data structures size is inversely proportional to the number of nodes
- The language features multiple collective communication patterns
- Data flows can be typed, to transmit variable data elements
- Tasks can be specialized to target specific devices and refined to adapt to multiple granularities
- Termination mechanism part of the runtime (counting or distributed termination detection)

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Parameterized Task Graph (PTG)

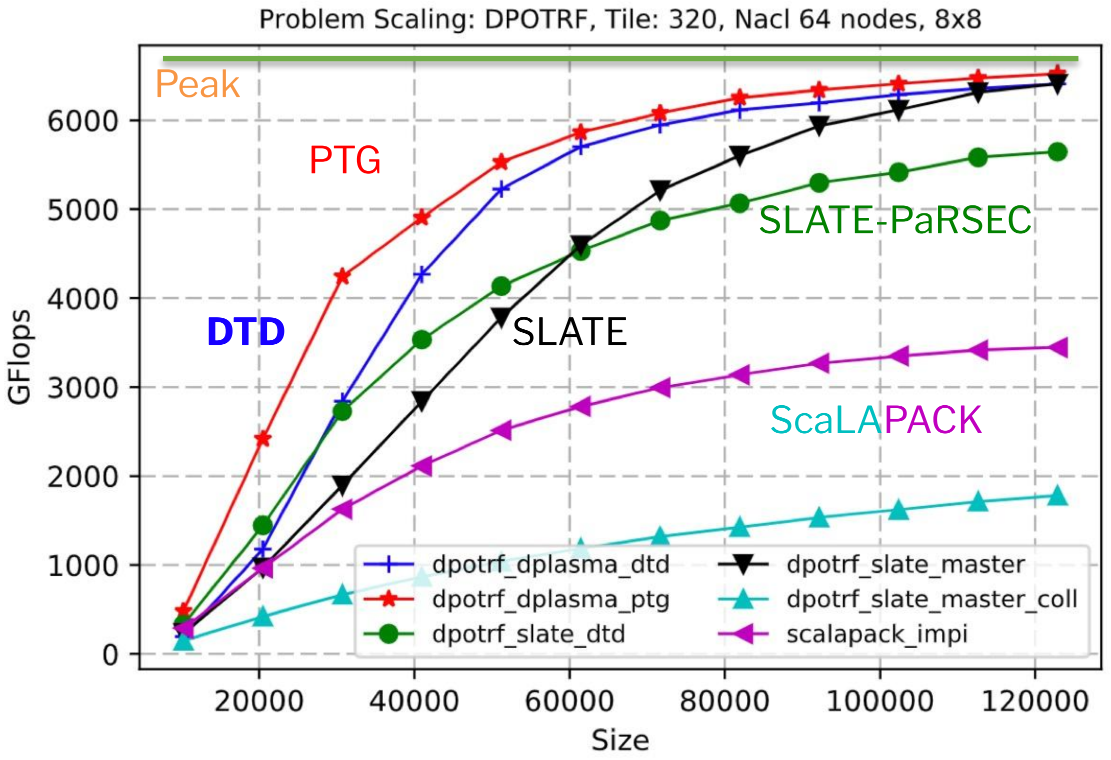




PaRSEC Domain Specific Languages

simplicity — flexibility

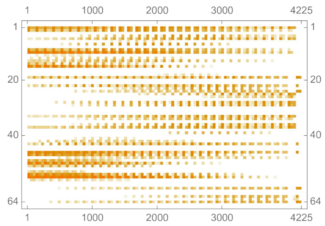
- Dynamic Task Discovery
- SLATE – C++
- Parameterized Task Graph

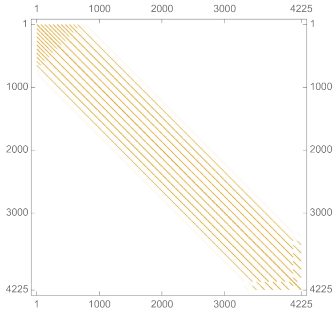


ICL INNOVATIVE COMPUTING LABORATORY

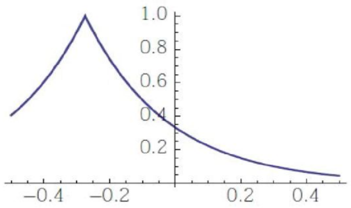
THE UNIVERSITY OF TENNESSEE KNOXVILLE

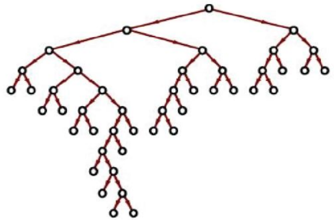
TTG: Motivation





- Some algorithms work on irregular data
  - Block-sparse matrices
  - Sparse matrices
- Others work on irregular data and the DAG is data-dependent
  - Approximative representation of functions using trees
- PTG is not well suited for the latter case (SLATE isn't either)
- DTD has scalability issues
  - All processes need to discover a consistent view of the DAG
  - DAG Pruning is sometimes complex to get right for programmers, especially if the DAG is data-dependent



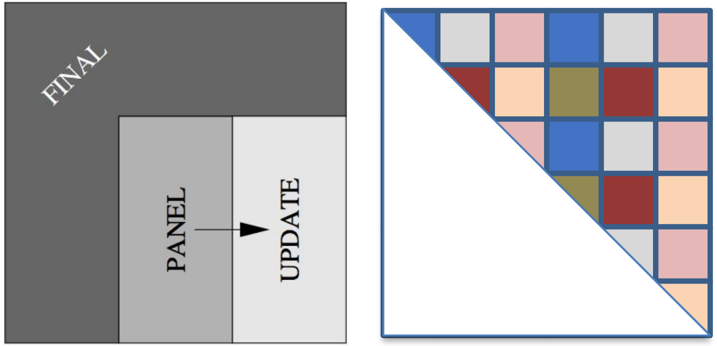


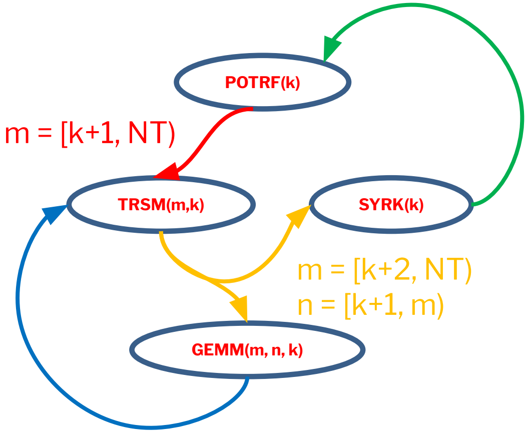
- TTG: a C++ API to dynamically discover the DAG, with process-local discovery only

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Cholesky in TTG

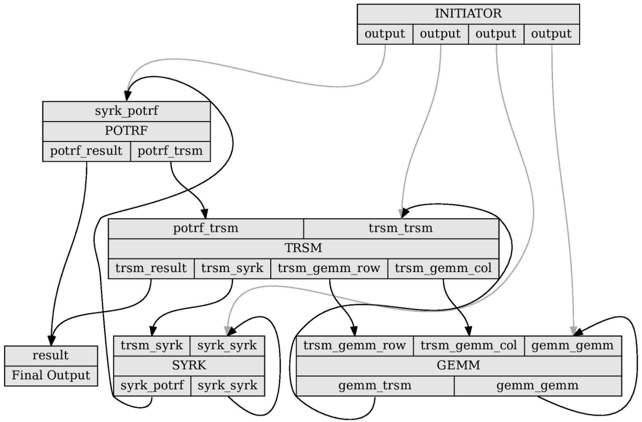





```
/* Edges with 1-tuple task IDs */
ttg::Edge<Int1, Tile> init_potrf;
/* Edges with 2-tuple task IDs */
ttg::Edge<Int2, Tile> potrf_trsm, trsm_result,
                      trsm_syrk, gemm_trsm;
/* Edges with 3-tuple task IDs, encodes the iteration K */
ttg::Edge<Int3, Tile> trsm_gemm_row, trsm_gemm_col;

auto POTRFOp = ttg::make_tt(potrf_fn /* not shown here */,
                            /* input edges */
                            ttg::edges(init_potrf),
                            /* output edges */
                            ttg::edges(potrf_results,
                                       potrf_trsm));

auto trsm_fn =
    [](const Int2& id,
       const Tile<T>&  tile_kk,
       Tile<T>&& tile_mk,
       std::tuple<ttg::Out<Int2, Tile<T>>,
                  ttg::Out<Int2, Tile<T>>,
                  ttg::Out<Int3, Tile<T>>,
                  ttg::Out<Int3, Tile<T>>>& out){
  const auto [I, J] = id;
  const auto K = J;

  /* call LAPACK library's tsrm function */
  TRSM(tile_kk, tile_mk);

  std::vector<Int3> row_ids, col_ids;
  /* ids for gemms row I */
  for (int n = J+1; n < I; ++n)
    row_ids.push_back(Int3(I, n, K));

  /* ids for gemms column I */
  for (int m = I+1; m < NROWS; ++m)
    col_ids.push_back(Int3(m, I, K));

  /* broadcast the result to 4 output terminals:
   * 0: to final output task writing back the tile;
   * 1: to the SYRK kernel;
   * 2: to the gemm tasks on in row I;
   * 3: to the gemm tasks in column K; */
  ttg::broadcast<0, 1, 2, 3>(
    std::make_tuple(id, Int2(I, K), row_ids, col_ids),
    std::move(tile_mk), out);
};
auto TRSMOp = ttg::make_tt(trsm_fn,
                           /* input edges */
                           ttg::edges(potrf_trsm, gemm_trsm),
                           /* output edges */
                           ttg::edges(trsm_result, trsm_syrk,
                                      trsm_gemm_row,
                                      trsm_gemm_col));
```

Cholesky in TTG

- Dense regular matrix
- Tile-based algorithm
- Comparisons:
  - Chameleon: runtime system StarPU; ‘Sequential Task Flow’ DAG representation (equivalent to DTD in PaRSEC)
  - DPLASMA: PaRSEC runtime with PTG DAG representation
  - SLATE: native SLATE implementation
  - ScaLAPACK: machine-provided ScaLAPACK implementation

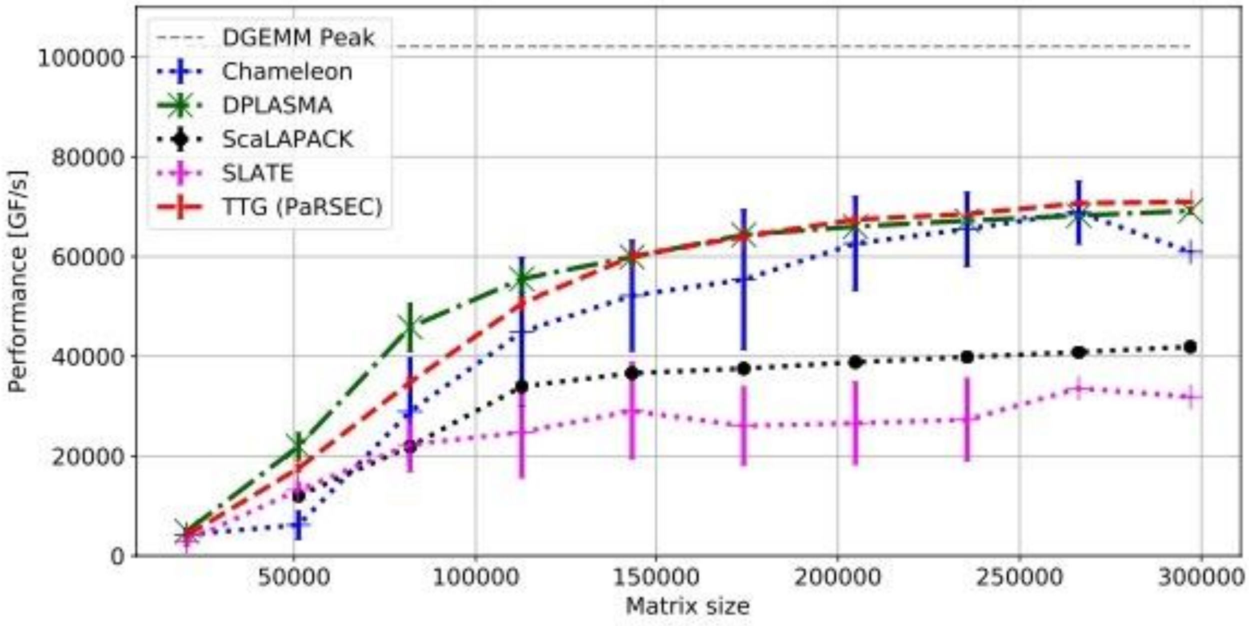


PaRSEC Domain Specific Languages

simplicity ⟵ ⟶ flexibility

- Dynamic Task Discovery
- SLATE – C++
- Parameterized Task Graph
- Template Task Graph

Problem scaling on 64 nodes / 3840 cores (tile size 512x512)



ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Conclusion

- PaRSEC is a distributed task-based runtime system targeting hybrid large scale platforms
- It supports multiple DAG of tasks input languages / APIs
  - Centered around the idea of a task class that features multiple alternative implementations and can be instantiated into tasks by providing an identifier
    - Build a graph of task classes, at compile time or at runtime
    - Tasks instantiated during execution unfold the DAG of tasks in a distributed way
  - Data centric runtime: manages data lifecycle and movement for the user
- New interface to program task systems, TTG
  - Fully functional over PaRSEC and MADNESS
  - Targets irregular applications and C++ environments
- Performance oriented runtime for TTG: PaRSEC
  - Work in progress
  - performance is on-par with state of the art implementations at reasonable scale
  - Adding accelerator support in TTG

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE