Template Task Graph
a New Programming Paradigm
for Task-Based Applications

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

\[
m = \lceil k + 1, NT \rceil
\]

\[
n = \lceil k + 1, m \rceil
\]
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
- Dynamic Task Discovery
- SLATE – C++
- Parameterized Task Graph
- Template Task Graph

Problem scaling on 64 nodes / 3840 cores (tile size 512x512)
PaRSEC Domain Specific Languages

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

Weak scaling on with submatrix of 30k x 30k per node (tile size 512x512)
Block-Sparse Matrix-Matrix Multiply (BSPMM)

\[ C = A \times A \]

A: block-sparse matrix, 48% density, real use case (Yukawa integral operator \(e^{-r/12}/r^{12}\) in the cc-pVDZ-RIFIT Gaussian atomic orbital basis for the main protease of the SARS-CoV-2 virus in complex with the N3 inhibitor)

- Algorithm corresponds to 2D SUMMA
- Two levels of control flow:
  - Around communications to prevent network overloading and favor communications related to the active set of tasks
  - To group tasks in order to favor local data reuse and increase cache effects between independent tasks on the same socket
- Compare with DBCSR (MPI+Open MP, 2.5D SUMMA)
Block-Sparse Matrix-Matrix Multiply (BSPMM)

Strong scaling of block-sparse GEMM on the Yukawa Integral operator (...)

![Graph showing performance vs. number of nodes/cores for different methods: DBCSR, TTG (MADNESS), TTG (PaRSEC).]
Multi-Resolution Analysis (MRA)

- Computes adaptive the order-10 multi-wavelet representation of 3-D Gaussian functions (exponent 30,000) to precision of $10^{-8}$ with Gaussian centers distributed randomly in a $[-6, 6]^3$ volume
- Random distribution $\rightarrow$ clustering $\rightarrow$ load imbalance
- Addressed by over-decomposition
- Data structure is 3D spatial tree that extends down to about 6 levels of adaptive dyadic refinement
- Benchmark computes compression and reconstruction of the 3D tree; norm of the function is computed at the same time to provide verification mechanism
Multi-Resolution Analysis, Strong scaling on Hawk, for 400 functions
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
Floyd-Warshall All-Pairs-Shortest Path (FW-APSP)

Comparisons:
- MPI + OpenMP implementation
  - Two levels of tiling:
    - Super-tiles communicated by row/column via MPI_Bcast
  - Parallel operation on sub-tiles via OpenMP
- TTG implementation
  - Single level of tiling
  - MADNESS runtime backend
  - PaRSEC runtime backend

```c
void fw_apsp(double **X, int N) {
    for(k=0; k<N; ++k)
        for(i=0; i<N; ++i)
            for(j=0; j<N; ++j)
                X[i][j]=min(X[i][j], X[i][k]+X[k][j]);
}
```
Strong scaling of the Floyd-Warshall benchmark using TTG and MPI+OpenMP on Hawk using 16 processes per node, 8 threads each (block sizes in square brackets)