Accelerating Cholesky-based Dense Matrix Inversion on GPUs

Ali Charara, David Keyes, Hatem Ltaief

{Ali.Charara, David.Keyes, Hatem.Ltaief}@kaust.edu.sa

Extreme Computing Research Center (ECRC) - King Abdullah University of Science and Technology (KAUST)

1. Background

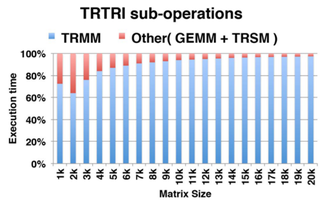
In computational sciences, explicitly inverting a covariance matrix (which is symmetric positive definite (SPD) matrix) is an operation that is frequently needed for various applications (Climate Modeling, Seismic Modeling, Computational Astronomy, etc…).

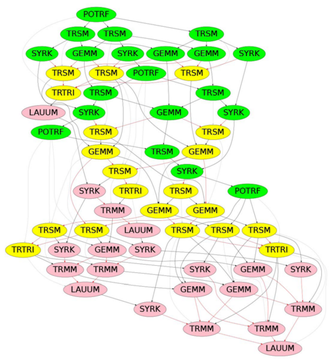
Inverting an SPD matrix based on Cholesky factorization involves two main operations:

- POTRF: Cholesky factorization.
- POTRI: Triangular matrix inversion, established by two sub-operations:
  - TRTRI: Inverting a triangular matrix.
  - LAUUM: Multiplying the triangular matrix with its transpose.

2. Motivation

- **Block Algorithm:** MAGMA[4] uses panel / update for BLAS operations.
  - MAGMA_POTRI:
    - **Low compute efficiency**, 13% of GPU peak perf. Two components:
    - TRTRI:
      - Low compute efficiency.
      - Heavily relies on TRMM operation.
    - LAUUM:
      - Moderate compute efficiency.
      - Moderate reliance on TRMM.
  - **TRMM**: triangular matrix multiply.
    - cuBLAS in-place-TRMM is a low performing kernel (version 7.0).
    - cuBLAS out-of-place-TRMM requires extra buffer allocation.
  - Block algorithm currently does not pipeline tasks across three stages of CHOLINV.
- **Tile algorithm**, implemented in PLASMA[5]:
  - DAGs for task dependencies.
  - Exposes **more parallelism**.
  - Better performance on multi-core.





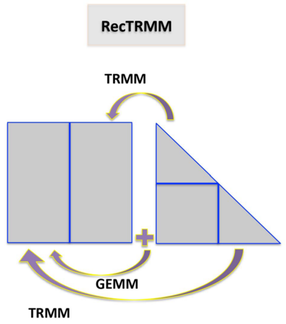
DAG of the pipelined stages of CHOLINV, for a 4x4 tiled matrix.

3. Our Contributions

- Accelerating cuBLAS-Inplace-TRMM operation.
- Accelerated CUDA kernels for core operations at small tile sizes:
  - → Recursive in-situ TRTRI & LAUUM.
- Pipelining tasks of the three stages of CHOLINV.
- Statically Scheduling the execution of the three stages of inversion.

4. Recursive in-place TRMM

- cuBLAS TRMM:
  - In-place: 11% of GPU Peak performance.
  - Out-of-place:
    - Extra buffer allocation.
    - Extra data-transfer.
    - **Violates BLAS legacy**.
- Recursive implementation:
  - Convert into **GEMM**s.
  - More **data reuse.**
  - **No extra storage.**
  - Available in KBLAS[1].

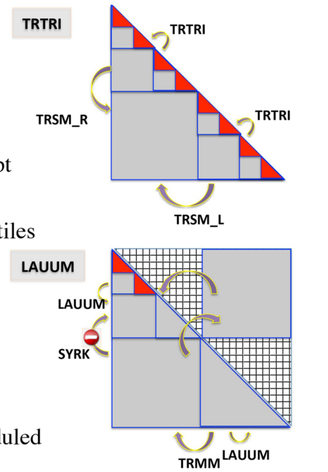


5. Recursive in-situ POTRI

- Methodology:
  - Process **bigger tiles**.
  - Convert operations into bigger GEMMs.
- Implementation:
  - cuBLAS for non diagonal operations (except TRMM).
  - Our optimized CUDA kernels for diagonal tiles (TRTRI, LAUUM).
  - Concurrent CUDA streams to run sub operations in parallel.
- Result:

→ Better performance for small tiles.

→ Used for diagonal blocks in Statically scheduled CHOLINV.



6. Static Scheduling

- **Pipeline** / Merge DAGs of three stages: POTRF / TRTRI / LAUUM.
  - CUDA runtime system executes tasks on concurrent streams ASAP.
- **Statically schedule** tasks execution based on merged DAGs:
  - Issue dependent tasks on the same thread / stream.
  - Explicitly wait for dependent tasks with CUDA events.

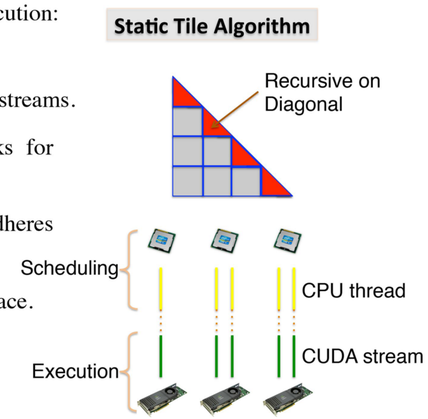
→ Minimal scheduling overhead.

- **Decouple** task scheduling from execution:
  - Scheduling on the CPU threads.
  - Execution on concurrent CUDA streams.

→ Populate GPU with more tasks for **better occupancy**.

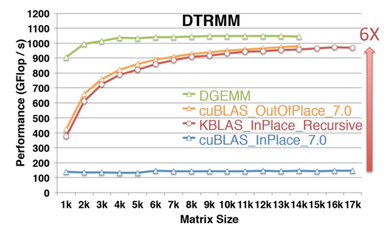
- Preserve **data locality**, a thread adheres to a tile, then moves to another:
  - Do all operations of the tile in place.
  - Wait for dependencies if needed.

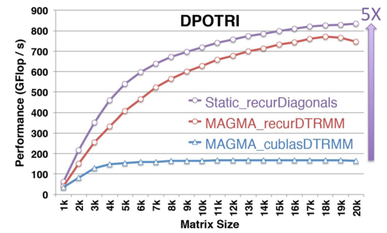
→ **Minimize data transfer**.

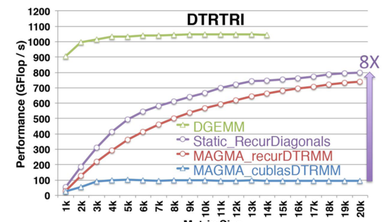


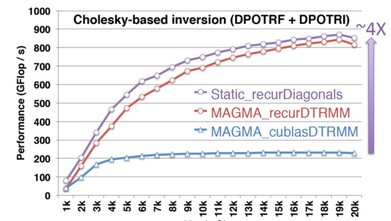
7. Results

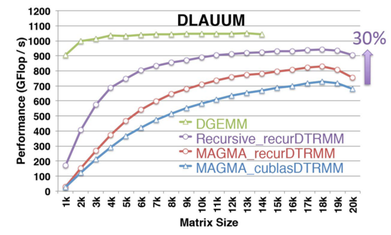
Experiments run on a shared memory dual socket Intel® Xeon® CPU E5-2650 of 8 cores each, running at 2GHz, with a total of 64GB memory, equipped with an NVIDIA Kepler K20 GPU. All computations performed in *double precision*.

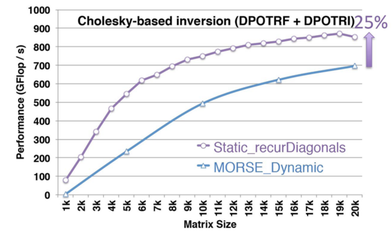












8. Ongoing / Futrue Work

- Multi-GPU statically scheduled factorization-inversion:
  - Less overhead than Dynamic Scheduling.
  - Data resides in CPU-memory → Larger matrix support.
- Support distributed memory environment.
- Investigate advantage of RecTRMM in leveraging other existing dense linear algebra algorithms.

9. Link & References:

1. RecTRMM available in KBLAS (open source Library) from KAUST
   - Download KBLAS at: http://ecrc.kaust.edu.sa/Pages/Res-kblas.aspx
2. A. Charara, H. Ltaief, D. Gratadour, D. Keyes, et al. 2014. *Pipelining computational stages of the tomographic reconstructor for multi-object adaptive optics on a multi-GPU system*. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14).
3. E. Agullo, H. Bouwmeester, J. Dongarra, et al. 2010. *Towards an efficient tile matrix inversion of symmetric positive definite matrices on multicore architectures*. In Proceedings of the 9th international conference on High performance computing for computational science (VECPAR'10). Springer-Verlag, Berlin, Heidelberg, 129-138.
4. Matrix Algebra on GPU and Multicore Architectures (MAGMA) http://icl.cs.utk.edu/magma/.
5. Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA), http://icl.cs.utk.edu/plasma/