Cholesky Factorization on Batches of Matrices with Fixed and Variable Sizes

Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra
Innovative Computing Laboratory, University of Tennessee, USA

Motivation

Many scientific applications require solving a number of independent small-size problems, such as

- Astrophysics
- Quantum chemistry
- Metabolic networks
- Image and signal processing
- CFD and resulting PDEs through direct and multi-frontal solvers

Such independent problems may have the same size (batched routine) or different sizes (vbatched routine). We address both situations.

Introduction

Taking Cholesky Factorization as a case study, we consider problems of the form,

$$\texttt{xPOTRF}(A^{(1)}_{LDA_1 \times N_1}) \rightarrow L_1 L_1^\intercal$$
$$\texttt{xPOTRF}(A^{(2)}_{LDA_2 \times N_2}) \rightarrow L_2 L_2^\intercal$$
$$\dots$$
$$\texttt{xPOTRF}(A^{(k)}_{LDA_k \times N_k}) \rightarrow L_k L_k^\intercal$$

- Matrices $A^{(1)}$ through $A^{(k)}$ can have the same size or different sizes.
- Design is based on BLAS routines, which can be used for other factorizations (e.g. LU and QR)

Algorithmic Design

Factorization Loop: Blocked left-looking Cholesky factorization with small blocking size nb.

Algorithm 1: The blocked Cholesky factorization.

for $i \leftarrow 0$ **to** m **Step** nb **do**
  $\bar{m} \leftarrow m - i$;
  if $(i > 0)$ **then**
    Panel Update $C_{\bar{m} \times nb} = C_{\bar{m} \times nb} - A_{\bar{m} \times i} \times (B^T)_{i \times nb}$;
  end
  Tile Factorize $(C_1)_{nb \times nb} \leftarrow \text{Cholesky}(C_1)$ (unblocked dpotf2);
  Panel Factorize $(C_2)_{(\bar{m}-nb) \times nb} \leftarrow C_2 (C_1^T)^{-1}$ (dtrsm);
end

Two Kernel Design Approaches:

- Loop-inclusive: All factorization steps (iterations) are executed in one kernel to maximize chances of data reuse
- Loop-exclusive: Each iteration is executed in a separate kernel launch to optimize resource utilization

Design Methodology:

- Start with fixed size problems using loop-inclusive and loop-exclusive kernels
- Performance tuning across different values of nb
- Use the best performing fixed-size kernel to support variable size problems

Testing vbatched Routines

We use batches with size distributions that follow a uniform random distribution in the interval [1:N_{max}], where N_{max} can be specified by the user.

Optimization Techniques and Performance Results

1. System Setup:

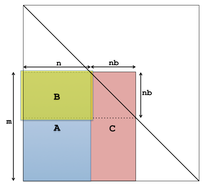
- 2 × 8-core Intel Sandy Bridge CPUs (Intel Xeon E5-2670, 2.6 GHz), 1 × Tesla K40c (745 MHz, ECC on)
- CUDA Toolkit 7.0, Intel MKL 11.3.0
- Results are shown for single and double precisions on batches of 3000 matrices

2. Key Changes to Routine Interface (in C)

- Input batches are passed as double pointer arrays
- In case of vbatched routines, matrix sizes and leading dimensions are passed as integer arrays
- Additional parameters: Batch sizes, and maximum dimension(s) across all matrices (for vbatched routines only)

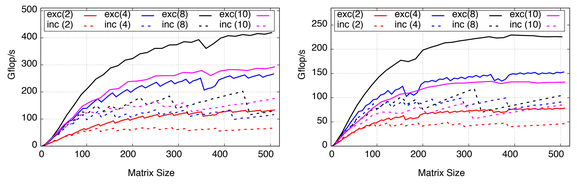
3. Symmetric Rank-k Updates

- The most dominating step (C = C - $A^T \times B$)
- We use double buffers to hide memory latency
- We also take advantage of the overlap between A and B to avoid redundant memory traffic
- Used in loop-inclusive and loop-exclusive kernels



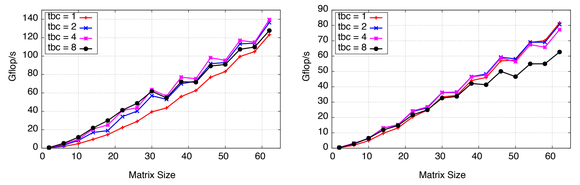
4. Performance Tuning

- Loop inclusive(inc.)/exclusive(exc.) kernels are tested against different values of nb
- Loop inclusive kernels do not utilize resources efficiently as the computation progresses, since more threads become idle



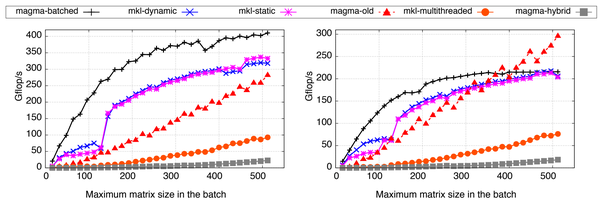
5. Thread Block (TB) Level Concurrency

- If matrices are very small, we can assign multiple matrices to a TB instead of one matrix
- Number of matrices per TB can be set dynamically during run time based on the matrix size
- Up to 2.86×/1.34× speedups in SP/DP



6. Final Fixed-size Performance

- Best competitor is a multicore CPU with dynamically unrolled OpenMP loop (one core per matrix)
- Up to 3×/2× speedups in SP/DP
- Improvement is more significant for smaller matrices



7. Adding support for vbatched factorization

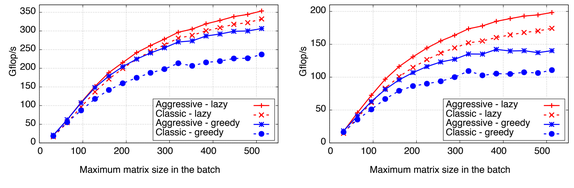
- Early Termination Scheme (ETM):
  - A vbatched kernel is always considered according to the largest matrix in the batch
  - ETMs detect and terminate threads with no work to do for smaller matrices in the batch
  - ETM-classic: can only terminate full thread blocks
  - ETM-aggressive: can also terminate idle threads in live thread blocks

8. Adding support for vbatched factorization "cont."

- Greedy vs. Lazy Scheduling
  - When should we start the factorization for smaller matrices in the batch?
  - Greedy scheduling: always start at the 0^{th}iteration.
  - Lazy scheduling: factorization of an arbitrary N×N matrix starts at iteration $\lceil \frac{N_{max}}{nb} \rceil - \lceil \frac{N}{nb} \rceil$.
  - Lazy scheduling tends to increase occupancy as the computation progresses (i.e. as the matrices gets smaller).

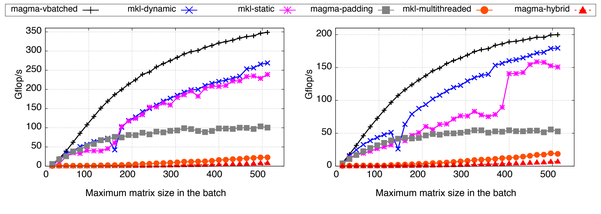
9. Impact of ETM and Scheduling Types

- With greedy scheduling, ETM-agressive is up to 50%/45% faster than ETM-classic in SP/DP
- If lazy scheduling is utilized, it improves ETM-classic by up to 87%/125% and ETM-agressive by up to 35%/90% in SP/DP



10. Final vbatched Performance

- Similarly, more performance improvement is observed in smaller matrices
- Up to 2.3×/1.88× speedups in SP/DP against the best competitor.



11. Future Directions

- Setting standard benchmarks for vbatched routines based on distributions from real applications
- Setting a standard interface for vbatched routine (e.g. like LAPACK)
- Error reporting to the user in batched routines (e.g. if the factorization succeeds except for one matrix)

References

[1] Azzam Haidar, Tingxing Dong, Piotr Luszczek, Stanimire Tomov, and Jack Dongarra. Batched matrix Computations on Hardware Accelerators based on GPUs. *International Journal of High Performance Computing Applications*, 2015.

[2] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. On the Development of Variable Size Batched Computation for Heterogeneous Parallel Architectures. In *17th Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC)*, 2016. (submitted).

[3] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. Performance Tuning and Optimization of Fixed and Variable Size Batched Cholesky Factorization on GPUs. In *International Conference on Computational Science (ICCS)*, 2016. (submitted).

Acknowledgement

This material is based upon work supported by:

- The National Science Foundation under Grant No. CSR 1514286
- NVIDIA
- The Department of Energy, and
- The Russian Scientific Foundation, Agreement N14-11-00190