GPU Acceleration of Small Dense Matrix Computation of One-Sided Factorizations

T. Dong, M. Gates, A. Haidar, P. Luszczek, S. Tomov

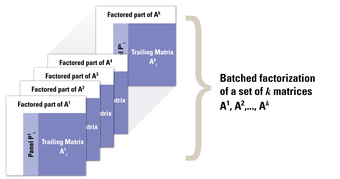
ICL UT INNOVATIVE COMPUTING LABORATORY

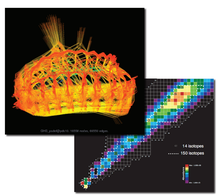
THE UNIVERSITY of TENNESSEE

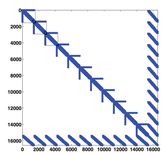
DEFINITION Batched factorization of a set of small matrices in parallel

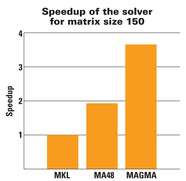
Numerous applications require factorization of many small matrices

- Structural mechanics
- Astrophysics
- Direct sparse solvers
- High-order FEM simulations









PROBLEM

- A natural way to implement a low level panel routine would be to load the panel to the GPU's shared memory and then do the entire computation before writing the result back to main memory. While this direction can be easily implemented, it cannot provide good performance for two main reasons:
  - A GPU's shared memory is currently limited to only 48KB per streaming multiprocessor (SMX) for the newest NVIDIA Kepler architecture, thereby limiting the panel size.
  - Saturating the shared memory per SMX can decrease performance, since only one thread block will be mapped to that SMX at a time, resulting in low occupancy and subsequently poor core utilization.

PROPOSED SOLUTIONS

- We implemented all the low level BLAS routines, such as dscal, dnorm, and dger, using a batched style that minimizes the use of shared memory. For example, we load only one column to the shared memory during the LU dgetf2.
- We found that such an implementation allows many thread blocks to be executed by the same SMX in parallel, taking better advantage of its resources.
- Since a CUDA warp consists of 32 threads, it is recommended to develop CUDA kernels that use a multiple of 32 threads per thread block.

OPTIMIZATION TECHNIQUES FOR DEVELOPING HIGH-PERFORMANCE BATCHED FACTORIZATIONS

PARALLEL PIVOTING IN LU

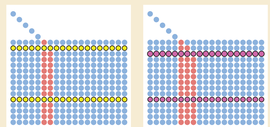
How does the LU swap work?

CLASSICAL SEQUENTIAL SWAP

Given pivot indices ipiv[1, …, ***nb***]
For row i = 1 to ***nb***
  Swap row i with row ipiv[i] of size N columns
Two rows, i_1 and i_2, can both swap with the same row, k, as shown, creating data dependency.

PARALLEL SWAP

Compute final index ppiv[i] for each row i
For column j = 1 to N
  Swap entries 1, …, ***nb*** with entries ppiv[1, …, ***nb***] in parallel



Bottlenecks:

- Swap is sequential and data dependent.
- Data transfer is not coalesced: GPU cannot read 32 values in parallel unless the matrix is stored rowwise. However, if the matrix is stored rowwise, the swap is fast, BUT other components become very slow.

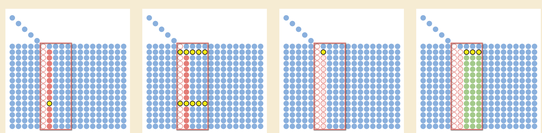
Proposition:

- Develop parallel version of swap.
- Improves write back of swapped rows as data transfer is now coalesced.

MULTI LEVEL BLOCKING

How does dgetf2 work?

① Find the max absolute value of a column below the diag "pivot"
② Swap the row of size ***nb*** columns
③ Scale the column below the diag by the inverse of the pivot
④ Update to the right of the column with dger

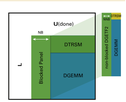


Bottlenecks:

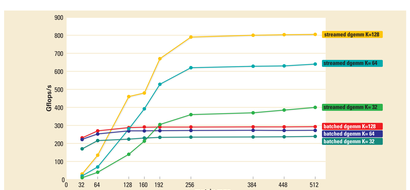
- It requires 30% of the global time.
- It is memory bound.
- Synchronization between each kernel launch.

Proposition:

- Nested blocking fashion: Develop a nested blocking technique that also blocks the panel in order to replace dger kernel with dgemm kernel.



USING GPU STREAM



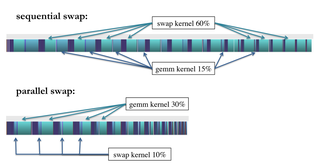
Bottlenecks:

- Batched gemm kernel from cuBLAS is well suited for small matrix sizes (<128), but stagnates for large sizes (>128).

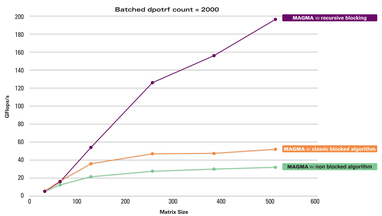
Proposition:

- Streamed gemm can provide higher performance for large matrix sizes (>128), thus we propose switching between streamed and batched gemm according to the size of the trailing matrix update.

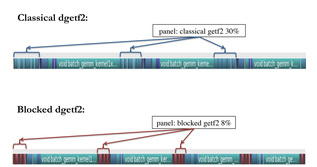
PROFILER TRACING



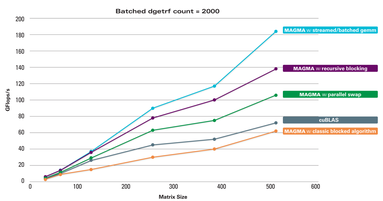
CHOLESKY PERFORMANCE



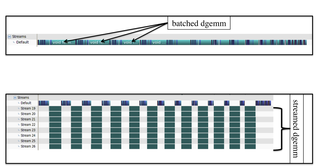
PROFILER TRACING



LU PERFORMANCE



PROFILER TRACING



QR PERFORMANCE