Heterogeneous band LU factorization in MAGMA

Razvan Carbunescu

Constant bandwidth with increasing N

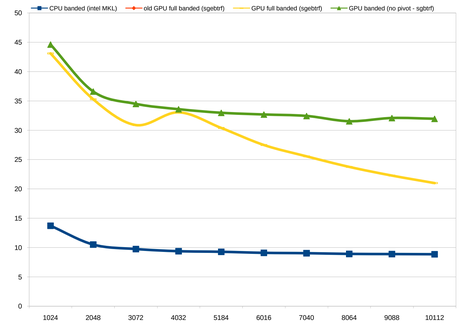

Figure 1. Gflops/s for b = 500

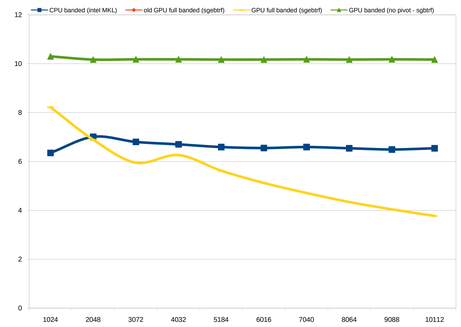

Figure 2. Gflops/s for b = 200

Timing breakdown of algorithms

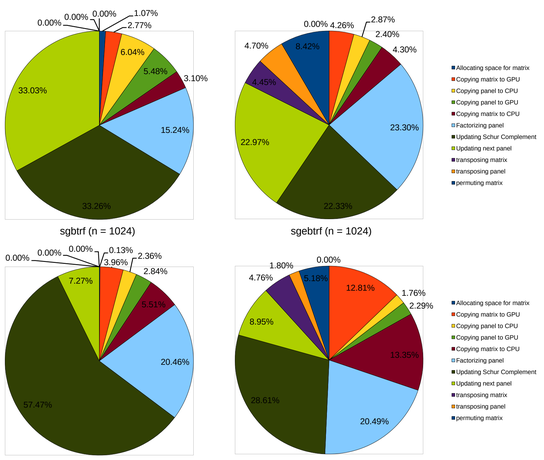

Figure 3. Timing breakdown for b = 10%

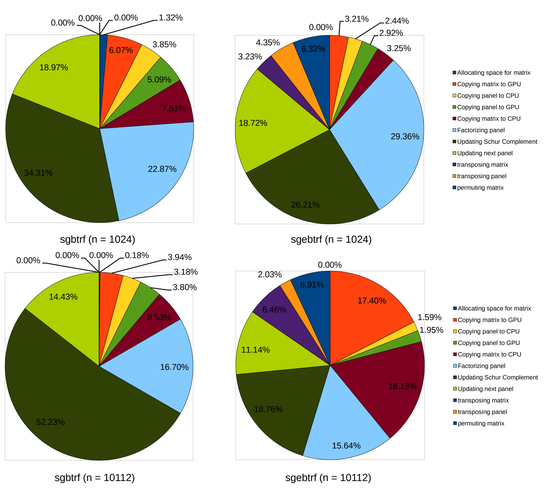

Figure 4. Timing breakdown for b = 500

Introduction of recent work

In the past 6 months we have worked on finishing the implementation of the new algorithm which was presented in the winter and have implemented a restricted version of that code that works for diagonally dominant matrices. This limitation of the code is due to work in progress on the pivoting for the new algorithm that is not yet complete.

The results of the timing breakdown however show that even in the worse cases tested pivoting would only take up to 15% of the total time so while a full implementation of the new algorithm (magma_**sgbtrf**) is not fully available the results can be considered representative of the speed of the final implementation.

The previous code (magma_**sgebtrf**) has also been improved with ideas from the BEBOP group at Berkeley to have 32-bit aligned memory to improve the copying times to and from the GPU; this was achieved by simple use of the cudaMallocHost commands and showed a 2 to 3x improvement in the transfer speed both for the initial and final matrix copies and also for the intermediate panel transfers.

The previous implementation was also corrected to ignore some extraneous flops that were being done on null data but maintains the full copy of the matrix (N^2) which represents a giant drawback for small bandwidth matrices.

All results presented were run on a Xeon dual socket quad-core E5405 machine with an attached GTX280 GPU.

Explanation of constant bandwidth results

The graphs to the left present two cases of constant bandwidth as the dimension of the matrix increases.

Both the new algorithm **(sgbtrf)** and the reference CPU code (intel MKL) seem to reach a constant Gflop rate with the GPU code having greater performance for the bandwidths presented but it is obvious that a crossover point exists where the CPU code is faster than the GPU implementation.

The GPU full banded algorithm **(sgebtrf)** loses performance as the size of the matrix increases since it copies on the order of $O(n^2)$ data versus MKL and sgbtrf which only copy O(n b) data and therefore would not represent a good choice for this problem.

Explanation of timing breakdown

The pie charts on the left show the timing breakdown for the two algorithms implemented (**sgbtrf** being the new heterogeneous LU algorithm and **sgebtrf** being the previously implemented but improved code)

The parts of the graph that do the work that would also be seen in the intel MKL version are the panel factorization (light blue) and Schur complement update(dark green, light green) pies seen in the bottom of the charts. To this would also be added the pivoting time (dark blue) for the non-panel part of the matrix. As can be seen from the charts the older algorithm was spending a much larger time in the extraneous parts related to CPU-GPU memory transfers and setups.

In the b=10% bandwidth case the copying and setup costs of the matrix to the GPU can be seen to be relatively hidden. However they become significant as the size of the matrix reaches 10112 (they reach ~30% of the cost versus only 10% for sgbtrf)

In the fixed b=500 bandwidth case the data is even worse with the parts reaching almost 45-50% of the cost which is due in part to the copying of the entire matrix to and from the GPU

Acknowledgements

I would like to thank Prof Jim Demmel from UC Berkeley and Prof Stan Tomov from University of Tennessee for their help during this project.

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, Oracle, and Samsung.

Explanation of proportional bandwidth results

The codes being compared both here and in the constant bandwidth are:

CPU banded (intel MKL) – MKL sgbtrf code running on CPU's only
Old GPU full banded (sgebtrf) – previously presented sgebtrf algorithm
GPU full banded **(sgebtrf)** – updated old algorithm with pinned memory transfer and reduction of extraneous flops
GPU banded **(sgbtrf)** – new algorithm presented at last retreat but implemented without pivoting

The older GPU sgebtrf is presented for reference to show the improvement in the code since the last version and therefore is only present in a few of the graphs.

As can be seen in most of the graphs on the right side the improved **sgebtrf** code remains the better choice for larger bandwidth problems even surpassing the **sgbtrf** implementation for b=25% and b=10% (in the latter case the numbers presented are about the same but the sgbtrf code would suffer a slight hit due to the latter addition of pivoting)

However it becomes quickly obvious from the data presented that as the size of the bandwidth decreases and the extra work spent on copying a full matrix when only a narrow band is operated upon the **sgebtrf** algorithm loses ground and **sgbtrf** becomes the correct choice.

There is still a crossover point however even for the new algorithm as can be seen by the b=1% bandwidth data where the CPU algorithm is a better choice because of the much smaller data available to work on.

Work in progress and Future work

At the moment the next step is the correction of the pivoting scheme for **sgbtrf** (partially implemented but error bounds outside norms).

The future work for this algorithm would be to apply some of the concepts presented by A. Gearhart and G. Ballard on heterogeneous CA lower bounds to the LU algorithm and implementation of an algorithm to attain that bound.

Current ideas have a TSQR type pivoting and split for Panel and some sort of block recursive or 2D non-uniform split between the heterogeneous processors based on the factors given by their specifications for the next panel and Schur complement update.

References

(1) PLASMA webpage: http://icl.cs.utk.edu/plasma/
(2) MAGMA webpage: http://icl.cs.utk.edu/magma/

Increasing Bandwidth as a % of N

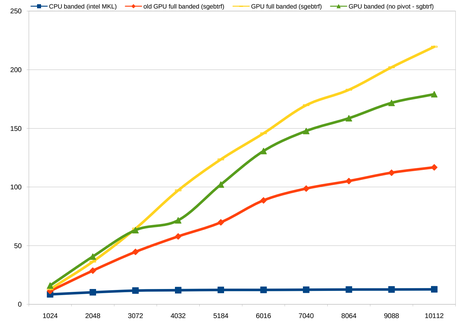

Figure 5. Gflops/s for b = 25%

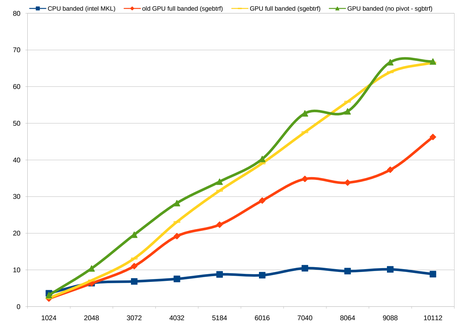

Figure 6. Gflops/s for b = 10%

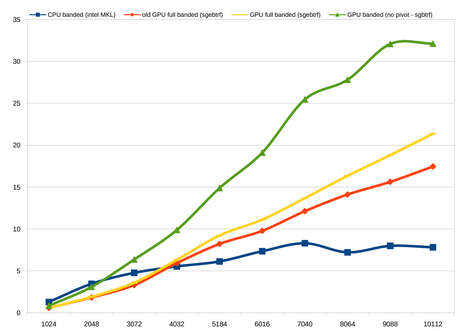

Figure 7. Gflops/s for b = 5%

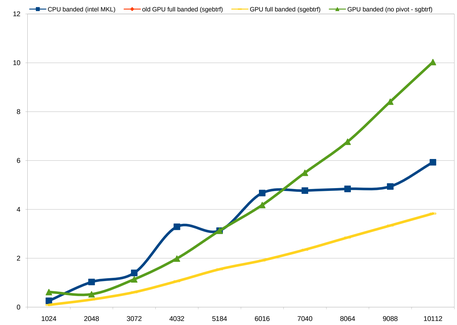

Figure 8. Gflops/s for b = 2%

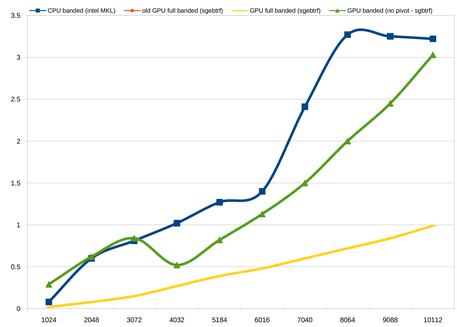

Figure 9. Gflops/s for b = 1%