

EM-TR-11: CUDA Benchmarks with Dense Hierarchical Linear System Solver

Ilgis Ibragimov

Elegant Mathematics Ltd, Hanauer Mühle, 66564, Ottweiler, Germany

Benchmarks with NVIDIA 260 GTX hardware for solution large dense linear systems with hierarchical structures are discussed.

A linear system with 163,840 unknowns was generated and solved in GPU with 25 times speedup in regards to Quad Core Xeon 2.66HGz.

Matrix generation shows 20 times speedup with a peak performance of 70 GFlop/s.

The iterative solver and compressed matrix multiplication algorithm produce up to 50 times speedup with the peak performance 6 GFlop/s = 45 GB/s memory bandwidth.

1 Motivation

Let us consider a solution of Stokes flow with boundary element method. The physics of Stokes flow assume that:

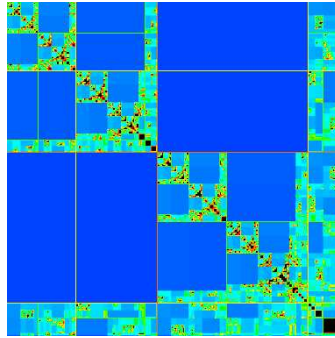
- low Reynolds number $Re \ll 1$,
- convection is neglected (linear case),
- pseudo-steady:

$$\begin{cases} \nabla \bar{\sigma} &= -\nabla p + \mu \nabla^2 \bar{v} = 0 \\ \nabla \bar{v} &= 0 \end{cases}$$

If we consider 3-dimensional domain, one need to use a lot of discretization points (order of 10^8) to get correct results for driven cavity with complicated boundary by finite element method (FEM). Here we get a large sparse matrix with 10-20 nonzeros in each column/row. Instead of FEM, one can use a boundary element method (BEM) and discretize only boundary elements. It leads to fully populated matrix with the total amount of unknowns equal to the amount of boundary elements.

The main problem of BEM is the system matrix with fully populated entries. It is reasonably smaller than in the FEM (order of 10^5), however it is not sparse since in BEM each element interacts with other one. Hence, a straightforward implementation requires huge amount of memory.

Last time results and developments in the field of BEM method allow to approximate fully populated BEM matrixes with the hierarchical structures and low rank blocks. The matrix is divided to several blocks, each block can be approximated with low rank LU, QR or SVD decomposition. If block's rank is smaller then the minimal dimension, one can store it with smaller memory requirements as if it is the dense matrix.



This picture corresponds to one example of dense hierarchical matrix. The blue color corresponds to high compression and red — almost no compression. Black blocks correspond to dense blocks without compression.

The theory of H-matrix approach and related dense compressed topic were heavily studied by Kolotilina in 1986 [1], Rohlin in 1987 [2], and Hackbusch in 1999 [3] and many others and nowadays it is well known robust dimensionality reduction tool.

2 Implementation

A set of 2D triangular grids for boundary elements was considered. The benchmark grids are constructed by the discretization of a sphere. The grid sizes are shown in the following table:

Exp. No.:	1	2	3	4	5
# Unknowns:	1280	5120	20480	81920	163840
# Points:	642	2562	10242	40962	81922
# Triangles:	1280	5120	20480	81920	163840

Based on this grid, one need to compute a cluster tree based on geometrical nested dissection algorithm.

The cluster tree construction takes so small computational time, that it is not very important to discuss CUDA implementation of this part. The biggest benchmark with 81920 unknowns needs only a few seconds for cluster tree generation.

Based on the cluster tree, we need to

- generate matrix elements,
- compress them in blocks,
- construct matrix by vector multiplication algorithm, and
- solve the system.

We cannot generate all matrix elements and start compression, the total amount of data required for storage of Example No 4, is about 25GB. Hence, modern algorithms, like adaptive cross approximation and incomplete rank revealing QR algorithms, are used for compression together with generation.

The compression rate of these examples are shown in the following table:

Exp. No.:	1	2	3	4	5
# Unknowns:	1280	5120	20480	81920	163840
Mememory Usage:	3.8Mb	19Mb	82Mb	360Mb	740Mb
Compression Rate:	61%	19%	5.0%	1.4%	0.7%
Memory for Fully Populated Matrix:	6.3Mb	100Mb	1.6Gb	25Gb	100Gb

where the memory for fully populated matrix is an estimation.

Hence, we expect that NVIDIA 260 GTX has enough memory to generate and solve a system with compressed dense matrix of 82K unknowns.

The following table shows a timing of all stages during the solution of problem on Quad Core Xeon 2.66 with heavy quad core optimization:

Exp. No.:	1	2	3	4	5
# Unknowns:	1280	5120	20480	81920	163840
Elements:	18s	78s	5.7m	27m	55m
Compression:	2.5s	4.2s	9.8s	21s	39s
Matrix-by-vector:	0.02s	0.19s	1s	5.1s	10s
Iterations:	<0.01s	0.01s	0.03s	0.1s	0.2s
Total:	24s	2m	11m	69m	142m

where

- **Elements** — a computational time for matrix element generations,
- **Compression** — a computational time for a low rank compression of matrix blocks,
- **Matrix-by-vector** — a computational time for matrix by vector multiplications that is used in the iterative method,
- **Iterations** — a computational time for one iterative step of BiCGStab.
- **Total** — a total computational time.

Let us remark that matrix element generation and low rank compression algorithms can be performed in a single precision, however an iterative part requires double precision arithmetics.

Let us consider the NVIDIA 260 GTX implementation (preliminary benchmarks) for this problem.

The results are very preliminary, so, for example, matrix element generation was installed in the same kernel, which generates low rank blocks; after generation, low rank blocks are permuted to other data structure using CPU; there are only measurement time for one matrix-by-vector operation and etc. Hence, the following results are benchmarks and represent the main asymptotics of

this GPU implementation. Additionally, many other important features not yet tested and implemented, for example, iterative methods with several simultaneous matrix-by-vector multiplications.

Exp. No.:	1	2	3	4	5
# Unknowns:	1280	5120	20480	81920	163840
Elements+Compression:	23s	28s	44s	110s	3m
Matrix-by-vector:	0.03s	0.04s	0.05s	0.12s	0.2s
Iterations:		<0.01s		0.01s	0.01s
Total:	28s	37s	62s	3m	5.5m

Let us write it in GFlop/s and speed terms and join in two main parts of the algorithm — matrix generation and iterative parts:

Exp. No.:	1	2	3	4	5
# Unknowns:	1280	5120	20480	81920	163840
Matrix Generation and Compression					
GFlop/s	3	12	34	59	71
SpeedUp	0.9	3	8	15	20
Iterative Part (BiCGStab and Matrix-by-Vector)					
GFlop/s	0.3	0.9	3.3	6.0	6.3
SpeedUp	0.6	5	20	43	47
Complete Solver					
SpeedUp	0.9	3	11	23	26

The matrix generation requires $10^4 - 10^5$ Flops per one computed word in compressed matrix. So, we can achieve here maximal possible performance. Indeed we have 60 GFlop/s, which is considerably good result for preliminary test, compared with 1 GFlop/s per core on Xeon.

The iterative methods deal with large vectors, which do not fit in cache of CPU or shared memory of GPU. In addition, matrix-by-vector multiplication requires the same amount of memory accesses as the Flops's amount. So, we expect here no better performance than the memory bandwidth speed. Indeed we reach $45\text{GB/s} = 6\text{GFlop/s}$ for the largest example, and see that the same example on CPU gives only $800\text{MB/s} = 100\text{MFlop/s}$. It is a property of iterative solvers, and it is impossible to improve it without clever methods, like several simultaneous matrix-by-vector multiplications.

The similar performance in iterative methods can be achieved with a cluster of workstations with 32-64 nodes, where the total memory bandwidth is comparable with NVIDIA GPU memory bandwidth.

3 Conclusion

The preliminary benchmarks with CUDA dense hierarchical solver provide good speedup, that is compared with a speedup of small cluster of workstations with 32-64 nodes. The computational problem with dense 164K unknowns is solved in core main memory of NVIDIA 260 GTX. Estimated biggest possible matrix size with C1060 usage should be around of 500K with dense matrices.

Arithmetic complexity fluctuates depending on the complexity of domain, and for large problems remains achieved values.

References

- [1] Kolotilina. Explicit preconditioning of systems of linear algebraic equations with dense matrices. *J. Sov. Math.*, 43:2566-2573, 1988. English translation of a paper first published in *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo im. V.A. Steklova AN SSSR* 154 (1986) 90-100.
- [2] Greengard L, Rokhlin V. A fast algorithm for particle simulation. *J. Comput. Phys.* 1987; **73**:325–348.
- [3] Hackbusch W, Nowak Z. On the fast matrix multiplication in the boundary element method by panel clustering. *Numer. Math.* 1989; **54**(4):463–491.
- [4] CUDA 2.1 Reference Manual. NVIDIA Corporation. <http://www.nvidia.com/>