Conjugate Gradients on Graphic Hardware: Performance & Feasibility

Serban Georgescu[1] and Hiroshi Okuda[2]

[1] University of Tokyo, Dept. of Quantum Engineering and Systems Science, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8654
[2] University of Tokyo, Research into Artifacts, Center for Engineering (RACE), 5-1-5 Kashiwa-no-ha, Kashiwa, Chiba 277-8568

Abstract. The Conjugate Gradient method (CG), one of the most commonly used iterative methods for solving very large systems of equations, has a history of running at less than 10% of peak processor performance, because its memory bounded nature and irregular access patterns. Due to their low cost and very large bandwidth, one solution that becomes more and more attractive is using GPUs as accelerators. In this research, we investigate how last generation GPUs perform as accelerators for the CG method by studying the performance and convergence on a large set of test matrices. Although an average 3-5x speedup is obtained even after applying correction iterations to compensate for the smaller precision of the GPU, we find the solver does not work for the majority of the test matrices. Based on a condition number analysis, we find a threshold above which no matrix will converge and discuss alternatives for making the system more feasible.

1 Introduction

Preconditioned Krylov iterative solvers are widely used in the field of finite element computation or for other problems where the matrix of the system to be solved is too large for the direct methods to work. Among these iterative solvers, due to its small memory footprint, the Conjugate Gradient (CG) method is one of the most commonly used. Composed only of BLAS 1 and 2 operations, the CG is a clear example of memory bounded application. Moreover, dominated by the sparse matrix vector multiplication (SpMV) kernel, the CG has to cope with an irregular memory access pattern.

Motivated by the performance advantage (in particular by the difference in sustainable bandwidth) and the recent entrance of Nvidia and AMD in the HPC market, we investigate the feasibility of last generation graphic card as accelerators for the CG method. By feasibility we understand both cost-performance and the ability of GPUs working in single precision to converge, in double precision accuracy, for a wide range of real work problems. This later aspect is very important since the graphic cards working in full double, which are starting to become available, cost a few times more than their single precision versions. Put shortly, the question we are trying to answer is whether and for which problems can one use a cheaper, single precision GPU and where must one buy the much more expensive double precision ones.

The first GPGPU implementation of the CG method for unstructured grids, without considering accuracy issues, was reported in [1] while similar work, on modern hardware, was reported in [2]. The mixed-precision approach we use here to obtain double precision accuracy while doing most of the work in single precision on the GPU was first presented in [3] and later scaled up to a GPU cluster in [4]. While convergence for ill-conditioned matrices is analyzed here, they only do so for a CG solver preconditioned by a multigrid preconditioner. This setup is particular to their solver does not apply for general unstructured grid solvers.

The most important contribution of this work is the finding of a threshold for the condition number of the system matrix above which no matrix will converge and showing that, even using powerful preconditioners, the convergence does not improve much. Besides this, we estimate the real world performance of a GPU-accelerated CG solver by comparing it with powerful BLAS libraries on the CPU on a large set of test matrices.

2 Solver performance

We tested the GeForce 8800GTX GPU against a system equipped with an 2.4GHz Core2Duo processor. On the CPU side we used the Intel-supplied MKL 10.0 library, the standard (NIST) implementation and the optimized sparse kernels from OSKI. On the GPU side we used Nvidia's CUBLAS combined with our implementations of the SpMV kernel. To improve generality, we took all matrices from the University of Florida Sparse Matrix Collection which were symmetric positive-definite and with more than 10,000 rows.

Performance results for all the kernels involved in implementing the CG method are shown in Fig.1. GPU performance increases with problem size, which provides enough work to keep busy the graphic pipelines.

Using the aforementioned kernels, a CG solver was implemented and benchmarked with and without iterative refinement. The only preconditioner used was diagonal scaling. Performance results are shown in Fig. 2. While on an iteration-preformance basis the GPU is up to 13x faster than the CPU, after running the iterative refinement iterations, the overal average speedup was about 3-5x.

3 Convergence

In order to achieve double precision accuracy while running the CG solver in single precision, we used iterative refinement. We discovered that under diagonal preconditioning only 20% of the matrices could converge. By doing a condition number analysis, we discovered that all matrices with condition number (computer after preconditioning) smaller than roughly 10^5, can successfully converge, in double precision accuracy, while matrices above this threshold fail to do so.

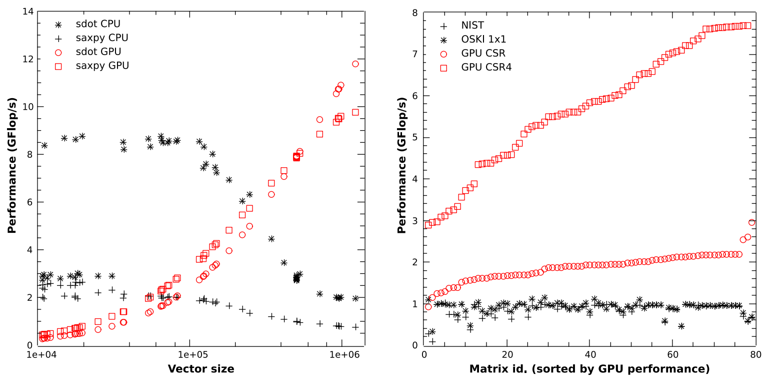

Fig. 1. Performance comparison between CPU and GPU for single precision vector operations (left) and SpMV (right). In the left, the AYPX operation is not shown as it has the same performance as AXPY. In the right, the performance MKL is not shown since it is provided only in double precision. CSR4 represents CSR padded and read four elements at a time, which greatly improves the performance of the GPU.

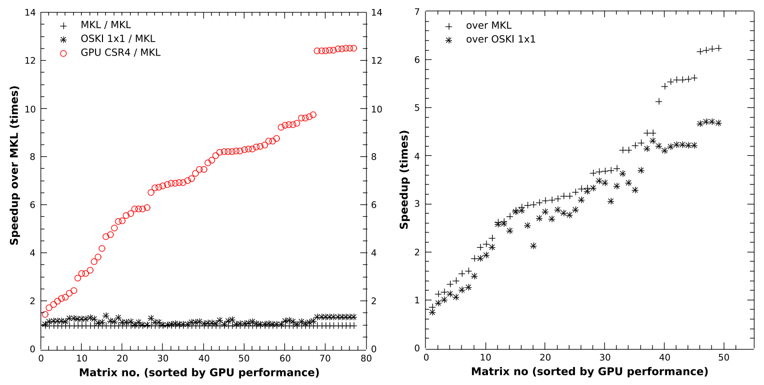

Fig. 2. Solver speedup for one iteration (left) and overall, including iterative refinement (right)

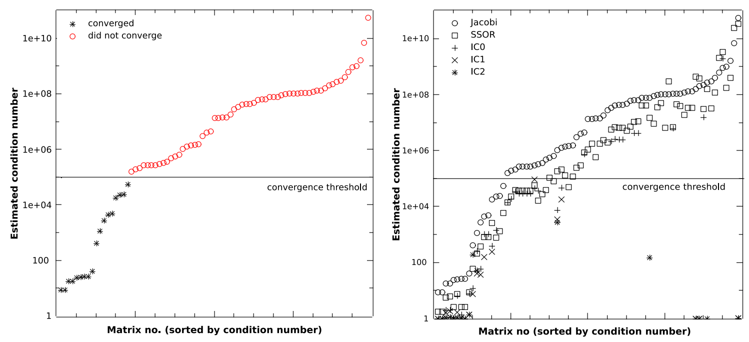

Fig. 3. The condition number threshold (left) and condition numbers after applying a series of preconditioners (right)

This behavior can be clearly seen in Fig.3 (left). Trying to find ways to get below this threshold, we computed the condition number resulted after applying SSOR and IC0-2 preconditioning (Fig.3, right). It was found that, while applying a powerful preconditioner tends to reduce the condition number with and order of magnitude or so, in most cases, this is not enough to get below the threshold and therefore to converge. Besides, actually implementing such preconditioner on the GPU is a very difficult task.

4 Conclusions

Tests conducted on a large collection of real-world matrices showed that single precision GPUs make very good accelerators for the CG method if and only if the matrix is well behaved, meaning that its condition number, computed after preconditioning, should be below 10^5. For ill-conditioned matrices, in the current solver setting, double precision is necessary. We are currently investigating two possible ways of overcoming this problem: using emulated-double in conjuction with iterative refinement and using the single-precision CG as a preconditioner for an outer solver.

References

1. J. Bolz et al, "Sparse matrix solvers on the GPU: conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 917–924, 2003.
2. L. Buatois et al, "Concurrent number cruncher: an efficient sparse linear solver on the GPU," *Lecture Notes in Computer Science*, vol. 4782, pp. 358–371, 2007.
3. D. Goddeke et al, "Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 22, no. 4, pp. 221–256, 2007.
4. D. Goddeke et al, "Exploring weak scalability for fem calculations on a gpu-enhanced cluster," *Parallel Computing*, vol. 33, no. 10-11, pp. 685–699, 2007.