A fast GPU Implementation of the Deflated Preconditioned Conjugate Gradient method

List of authors:

C. Vuik [1]

R. Gupta [2]

C.W.J. Lemmens [3]

Large sparse linear systems are mostly solved by Preconditioned Krylov Methods combined with some form of coarse grid acceleration. We consider the Deflated Preconditioned Conjugate Gradient Method. Most of the important building blocks of this algorithm could be optimized for execution on parallel architectures. Recently, with the advent of General Purpose Computing on the Graphical Processing Unit (GPU) it is possible to achieve $10 - 100$ times reduction in computing times. However, many preconditioners involved in the solution of the systems of interest do not run optimally on the GPU. A large class of preconditioners is based on incomplete Cholesky decompositions. Solving the lower and upper triangulars system is inherently sequential. So the challenge is how to make these preconditioners suitable for the GPU which can only be used with parallel methods. In this paper we show that it is possible to achieve a 20 fold speedup for a combination of suitable building blocks.

We use two levels of Preconditioning with the CG method to solve a linear system [1]. Our system arises from the discretization of the Pressure Correction Equation. This equation is the most time-consuming step in the solution of the Incompressible Navier-Stokes Equation using the Level Set Method. This method, as suggested in [2], is of interest in modeling bubbly flows. The Partial Differential Equations describing the Pressure Correction have been discretized through the use of finite differences. The linear system is of the form

$$Ax = b, A \in \mathbb{R}^{n \times n}, \tag{1}$$

where n is the number of degrees of freedom. We assume that A is symmetric positive definite.

With the advent of the Component Unified Device Architecture (CUDA) paradigm of computing available on NVIDIA GPU devices, it has become easier to write numerical software. Characteristic features of GPU devices are: one should use vectorizable algorithms (fine scale parallelism), reuse the fast shared memory as much as possible, if statements etc. lead to severe degradation of the performance, minimize communication between CPU and GPU, and the obtained accuracy and error checking is less on the GPU as compared to the CPU.

We perform our experiments using 5 point Laplacian matrix resulting from a 2D square grid. The same experiments have been done for a matrix originating from a two phase flow. The

[1]Delft University of Technology, Faculty of EEMCS, Mekelweg, Delft, The Netherlands
[2]Delft University of Technology, Faculty of EEMCS, Mekelweg, Delft, The Netherlands
[3]Delft University of Technology, Faculty of EEMCS, Mekelweg, Delft, The Netherlands

methods below have been implemented and tested in the given order. We start with the standard CG method, add a standard preconditioner (BIC) and combine it with a second level preconditioner (deflation). After that we replace the standard preconditioner by a parallel preconditioner and optimize the code for the GPU which leads to the fastest method: DIPCG2.

- (CGVV) Conjugate Gradient - Vanilla Version,
- (CGBIC) Conjugate Gradient - Block Incomplete Cholesky Preconditioning,
- (DPCG) Conjugate Gradient - Deflation and Block Incomplete Cholesky Preconditioning,
- (DPCG1) Conjugate Gradient - Deflation (Optimized - Level 1) and Block Incomplete Cholesky Preconditioning,
- (DIPCG1) Conjugate Gradient - Deflation (Optimized - Level 1) and Incomplete Poisson Preconditioning,
- (DPCG2) Conjugate Gradient - Deflation (Optimized - Level 2) and Block Incomplete Cholesky Preconditioning,
- (DIPCG2) Conjugate Gradient - Deflation(Optimized - Level 2) and Incomplete Poisson Preconditioning.

We use a Q9650 Intel Quad Core CPU however we only utilize a single core. We optimize it to use SSE instructions, unrolling loops and vectorizing using compiler switches. We also use the Meschach Blas Library for the Blas routines on the CPU. The GPU we use is a Tesla C1060 from NVIDIA. We use CUDA for writing our code on the GPU. The *CUBLAS* and *MAGMA* libraries are used when Blas functions are needed in the GPU version. Table 1 shows the results for a 512×512 grid. Note that the combination of the IP preconditioner and the deflation approach leads to a low number of iterations. Furthermore, after optimization, the gain in computing time is around a factor 20. For further details we refer to [3].

	Execution Times		No. of Iterations	
Code Version	CPU	GPU	CPU	GPU
CGVV	5.9	0.5004	652	649
CGBIC	5.1	5.59	327	327
DPCG	110.5	4.45	42	42
DPCG1	1.6	1.01	41	41
DIPCG1	1.7	0.25	49	49
DPCG2	1.6	0.89	41	41
DIPCG2	1.8	0.10	49	49

Table 1: Comparison GPU vs. CPU. Number of iterations required for convergence and execution times.

References

[1] J.M. Tang, C. and Vuik, Efficient Deflation Methods applied to 3-D Bubbly Flow Problems *Electronic Transactions on Numerical Analysis*, 26:330–349, 2007.

[2] S.P. Van der Pijl, A. Segal, C. Vuik, and P. Wesseling, A mass conserving Level-Set Method for modelling of multi-phase flows *International Journal for Numerical Methods in Fluids*, 47:339–361, 2005.

[3] R. Gupta, C. Vuik, and C.W.J. Lemmens, GPU Implementation of Deflated Preconditioned Conjugate Gradient submitted