3D Helmholtz Krylov Solver Preconditioned by a Shifted Laplace Multigrid Method on Multi-GPUs

H. Knibbe, C.W. Oosterlee, and C. Vuik

Abstract We are focusing on an iterative solver for the three-dimensional Helmholtz equation on multi-GPU using CUDA (Compute Unified Device Architecture). The Helmholtz equation discretized by a second order finite difference scheme is solved with Bi-CGSTAB preconditioned by a shifted Laplace multigrid method. Two multi-GPU approaches are considered: data parallelism and split of the algorithm. Their implementations on multi-GPU architecture are compared to a multi-threaded CPU and single GPU implementation. The results show that the data parallel implementation is suffering from communication between GPUs and CPU, but is still a number of times faster compared to many-cores. The split of the algorithm across GPUs limits communication and delivers speedups comparable to a single GPU implementation.

1 Introduction

As it has been shown in paper [5] the implementation of numerical solvers for indefinite Helmholtz problems with spatially dependent wavenumber, such as Bi-CGSTAB and IDR(s) preconditioned by shifted Laplace multigrid method on a GPU is more than 25 times faster than on a single CPU. Comparison of single GPU to a single CPU is important but it is not representative for problems of realistic size.

H. Knibbe (✉) · C. Vuik
Delft University of Technology, Delft, Netherlands
e-mail: hknibbe@gmail.com; c.vuik@tudelft.nl

C.W. Oosterlee
Dutch National Research Centre for Mathematics and Computer Science (CWI), Delft University of Technology, Delft, Netherlands
e-mail: c.w.oosterlee@cwi.nl

A. Cangiani et al. (eds.), *Numerical Mathematics and Advanced Applications 2011*,
DOI 10.1007/978-3-642-33134-3_69,

By realistic problem size we mean three-dimensional problems which lead after discretization to linear systems of equations with more than one million unknowns. Such problems arise when modeling a wavefield in geophysics.

Problems of realistic size are too large to fit in the memory of one GPU, even with the latest NVIDIA Fermi graphics card (see [6]). One solution is to use multiple GPUs. The currently widely used architecture consists of a multi-core connected to one or at most two GPUs. Moreover, in most of the cases those GPUs have different characteristics and memory size. A setup with four or more identical GPUs is rather uncommon, but it would be ideal from a memory point of view. It implies that the maximum memory is four times or more than on a single GPU. However GPUs are connected to a PCI bus and in some cases two GPUs share the same PCI bus, this creates data transfer limitation. To summarize, using multi-GPUs increases the total memory size but data transfer problems appear.

The aim of this paper is to consider different multi-GPU approaches and understand how data transfer affects performance of a Krylov solver with shifted Laplace multigrid preconditioner for the three-dimensional Helmholtz equation.

2 Helmholtz Equation and Solver

The Helmholtz equation in three dimensions for a wave problem in a heterogeneous medium is considered

$$-\frac{\partial^2\phi}{\partial x^2}-\frac{\partial^2\phi}{\partial y^2}-\frac{\partial^2\phi}{\partial z^2}-(1-\alpha i)k^2\phi=g, \tag{1}$$

where $\phi=\phi(x,y,z)$ is the wave pressure field, $k=k(x,y,z)$ is the wavenumber, $\alpha \ll 1$ is the damping coefficient, $g=g(x,y,z)$ is the source term. The corresponding differential operator has the form $\mathscr{A}=-\Delta-(1-\alpha i)k^2$, where Δ denotes the Laplace operator. The problem is given in a cubic domain $\Omega=[(0,0,0),(X,Y,Z)]$, $X,Y,Z\in\mathbb{R}$. A first order radiation boundary condition is applied $\left(-\frac{\partial}{\partial\eta}-ik\right)\phi=0$, where η is the outward normal vector to the boundary (see [2]). Discretizing equation (1) using the 7-point central finite difference scheme gives the following linear system of equations: $A\phi=g$, $A\in\mathbb{C}^{N\times N}$, $\phi,g\in\mathbb{C}^N$, where $N=n_xn_yn_z$ is a product of the number of discretization points in the $x-$, $y-$ and $z-$directions. Note that the closer the damping parameter α is set to zero, the more difficult it is to solve the Helmholtz equation. We are focusing on the original Helmholtz equation with $\alpha=0$.

As a solver for the discretized Helmholtz equation we have chosen the Bi-CGSTAB method preconditioned by shifted Laplace multigrid method with matrix-dependent transfer operations and a Gauss-Seidel smoother, (see [3]). It has been shown in [5] that this solver is parallelizable on CPUs as well as on a single GPU and provides good speed-up on parallel architectures. The prolongation in

this work is based on the three dimensional matrix-dependent prolongation for real-valued matrices described in [7]. This prolongation is also valid at the boundaries. The restriction is chosen as full weighting restriction. As a smoother the multi-colored Gauss-Seidel method has been used. In particular, for 3D problems the smoother uses eight colors, so that the color of a given point will be different from its neighbours.

Since our goal is to speed up the Helmholtz solver with the help of GPUs, we still would like to keep the double precision convergence rate of the Krylov method. Therefore Bi-CGSTAB is implemented in double precision. For the preconditioner, single precision is sufficient for CPU as well as GPU.

3 Multi-GPU Implementation

For our numerical experiments NVIDIA [6] provided a Westmere based 12-cores machine connected to 8 GPUs Tesla 2050 as shown on Fig. 1. The 12-core machine has 48 GB of RAM. Each socket has 6 CPU cores Intel(R) Xeon(R) CPU X5670 @ 2.93 GHz and is connected through 2 PCI-buses to 4 graphics cards. Note that two GPUs are sharing one PCI-bus connected to a socket. Each GPU consist of 448 cores with clock rate 1.5 GHz and has 3 GB of memory.

In the experiments CUDA version 3.2[1] is used. All experiments on CPU are done using a multi-threaded CPU implementation (pthreads).

In general GPU memory is much more limited than CPU memory so we chose a multi-GPU approach to be able to solve larger problems. The implementation on a single GPU of major components of the solver such as vector operations, matrix-vector-multiplication or the smoother has been described in [5]. In this section we focus on the multi-GPU implementation.

There are two ways to do computations on multi-GPU: push different Cuda contexts to different GPUs (see [6]) or create multiple threads on the CPU, where each thread communicates with one GPU. For our purposes we have chosen the second option, since it is easier to understand and implement.

Multiple open source libraries for multi-threading have been considered and tested. For our implementation of numerical methods on a GPU the main requirement for multi-threading was that a created thread stays alive to do further processing. It is crucial for performance that a thread remains alive as a GPU context is attached to it. Pthreads has been chosen as we have total control of the threads during the program execution.

[1] During the work on this paper, the newer version of CUDA 4.0 has been released. It was not possible to have the newer version installed on all systems for our experiments. That is why for consistency and comparability of experiments, we use the previous version

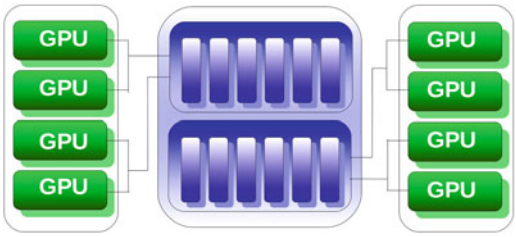

Fig. 1 NVIDIA machine with 12 Westmere CPUs and 8 Fermi GPUs, where two GPUs share a PCI bus connected to a socket

There are several approaches to deal with multi-GPU hardware:

1. *Domain-Decomposition approach*, where the original continuous or discrete problem is decomposed into parts which are executed on different GPUs and the overlapping information (halos) is exchanged by data transfer. This approach can however have difficulties with convergence for higher frequencies (see [4]).
2. *Data-parallel approach*, where all matrix-vector and vector-vector operations are split between multiple GPUs. The advantage of this approach is that it is relatively easy to implement. However, matrix-vector multiplication requires exchange of the data between different GPUs, that can lead to significant data transfer times if the computational part is small. The convergence of the solver is not affected.
3. *Split of the algorithm*, where different parts of the algorithm are executed on different devices. For instance, the solver is executed on one GPU and the preconditioner on another one. In this way the communication between GPUs will be minimized. However this approach requires an individual solution for each algorithm.

Note that the data-parallel approach can be seen as a method splitting the data across multi-GPUs, whereas the split of the algorithm can be seen as a method splitting the tasks across multiple devices. In this paper we are investigating the data-parallel approach and the split of the algorithms and make a comparison between multi-core and multi-GPUs. We leave out the domain decomposition approach because the convergence of the Helmholtz solver is not guaranteed. The data parallel approach is more intuitive and is described in detail in Sect. 4.

3.1 Split of the Algorithm

The split can be unique for every algorithm. The main idea of this approach is to limit communication between GPUs but still be able to compute large problems.

One way to apply this approach to the Bi-CGSTAB preconditioned by shifted Laplace multigrid method is to execute the Bi-CGSTAB on one GPU and the multigrid preconditioner on another one. In this case the communication only between the Krylov solver and preconditioner is required but not for intermediate results.

The second way to apply split of the algorithm to our solver is to execute the Bi-CGSTAB and the finest level of shifted Laplace multigrid across all available GPUs using data parallel approach. The coarser levels of multigrid method are executed on only one GPU due to small memory requirements. Since the LU-decomposition is used to compute an exact solution on the coarsest level, we use the CPU for that.

3.2 Issues

Implementation on multi-GPUs requires careful consideration of possibilities and optimization options. The issues we encountered during our work are listed below:

- Multi-threading implementation, where the life of a thread should be as long as the application. This is crucial for the multi-threading way of implementation on multi-GPU. Note that in case of pushing contexts this is not an issue.
- Because of limited GPU memory size, large problems need multiple GPUs.
- Efficient memory reusage to avoid allocation/deallocation. Due to memory limitations the memory should be reused as much as possible, especially in the multigrid method. In our work we create a pool of vectors on the GPU and reuse them during the whole solution time.
- Limit communications CPU→GPU and GPU→CPU.
- The use of texture memory on Multi-GPU is complicated as each GPU needs its own texture reference.
- Coalescing is difficult since each matrix row has a different number of elements.

4 Numerical Results on Multi-GPU

4.1 Vector- and Sparse Matrix-Vector Operations

Vector operations such as addition, dot product are trivial to implement on multi-GPU. Vectors are split across multiple GPUs, so that each GPU gets a part of the vector. In case of vector addition, the parts of a vector remain on GPU or can be send to a CPU and assembled in a result vector of original size. The speedup for vector addition on 8-GPUs compared to a multi-threaded implementation (12 CPUs) is about 40 times for single and double precision. For the dot product, each GPU sends its own sub-dot product to a CPU, where they will be summed into the final result. The speedup for dot product is about 8 for single precision and 5 for double precision. Note that in order to avoid cache effects on a CPU and to make a fair comparison, the dot product has been taken from two different

Table 1 Matrix-Vector-Multiplication in single (SP) and double (DP) precision

Size	Speedup (SP) 12-cores/1 GPU	Speedup (SP) 12-cores/8 GPUs	Speedup (DP) 12-cores/1 GPU	Speedup (DP) 12-cores/8 GPUs
100,000	54.5	6.81	30.75	5.15
1 Mln	88.5	12.95	30.94	5.97
20 Mln	78.87	12.13	32.63	6.47

Table 2 Speedups for Bi-CGSTAB in single (SP) and double (DP) precision

Size	Speedup (SP) 12-cores/1 GPU	Speedup (SP) 12-cores/8 GPUs	Speedup (DP) 12-cores/1 GPU	Speedup (DP) 12-cores/8 GPUs
100,000	12.72	1.27	9.59	1.43
1 Mln	32.67	7.58	15.84	5.11
15 Mln	45.37	15.23	19.71	8.48

vectors. The speedups for vector addition and dot product on multi-GPU are smaller compared to the single GPU because of the communication between CPU and multiple GPUs.

The matrix is stored in a CRS matrix format (Compressed Row Storage, see e.g. [1]) and is split row-wise. In this case a part of the matrix rows is transferred to each GPU as well as the whole vector. After matrix-vector multiplication parts of the result are transferred to a CPU where they are assembled into the final resulting vector. The timings for matrix-vector multiplication are given in Table 1.

4.2 Bi-CGSTAB and Gauss-Seidel on Multi-GPU

Since the Bi-CGSTAB algorithm is a collection of vector additions, dot products and matrix-vector multiplications described in the previous section, the multi-GPU version of the Bi-CGSTAB is straight forward. In Table 2 the timings of Bi-CGSTAB on many-core CPU, single GPU and multi-GPU are presented. The stopping criterion is 10^{-5}. It is easy to see that the speedup on multi-GPUs is smaller than on a single GPU due to the data transfer between CPU and GPU. Note that for the largest problem in Table 2 it is not possible to compute on a single GPU because there is not enough memory available. However it is possible to compute this problem on multi-GPUs and the computation on multi-GPU is still many times faster than 12-core Westmere CPU.

As mentioned above, the shifted Laplace multigrid preconditioner consists of a coarse grid correction based on the Galerkin method with matrix-dependent

Table 3 Speedups for colored Gauss-Seidel method on different architectures in single precision

Size	12-cores/1 GPU	12-cores/8 GPUs
5 Mln	16.5	5.2
30 Mln	89.1	6.1

prolongation and of a Gauss-Seidel smoother. The implementation of coarse grid correction on multi-GPU is straight forward, since the main ingredient of the coarse grid correction is the matrix-vector multiplication. The coarse grid matrices are constructed on a CPU and then transferred to the GPUs. The matrix-vector multiplication on multi-GPU is described in Sect. 4.1.

The Gauss-Seidel smoother on multi-GPU requires adaptation of the algorithm. We use 8-colored Gauss-Seidel, since the problem (1) is given in three dimensions and computations at each discretization point should be done independently of the neighbours to allow parallelism. For the multi-GPU implementation the rows of the matrix for one color will be split between multi-GPUs. Basically, the colors are computed sequentially, but within a color the data parallelism is applied across the multi-GPUs. The timing comparisons for 8-colored Gauss-Seidel implementation on different architectures are given in Table 3.

5 Numerical Experiments for the Wedge Problem

This model problem represents a layered heterogeneous problem taken from [3]. For $\alpha \in \mathbb{R}$ find $\phi \in \mathbb{C}^{n\times n\times n}$

$$-\Delta\phi(x,y,z) - (1-\alpha i)k(x,y,z)^2\phi(x,y,z) = \delta\left((x-500)(y-500)z\right), \quad (2)$$

$(x,y,z) \in \Omega = [0,0,0] \times [1000,1000,1000]$, with the first order boundary conditions. We assume that $\alpha = 0$. The coefficient $k(x,y,z)$ is given by $k(x,y,z) = 2\pi f l/c(x,y,z)$ where $c(x,y,z)$ is presented in the Fig. 2. The grid size satisfies the condition $\max_x(k(x,y,z))h = 0.625$, where $h = \frac{1}{n-1}$. Table 4 shows timings for Bi-CGSTAB preconditioned by the shifted Laplace multigrid method on the problem (2) with 43 millions unknowns. The single GPU implementation is about 13 times faster than a multi-threaded CPU implementation. The data-parallel approach shows that on multi-GPUs the communication between GPUs and CPUs takes a significant amount of the computational time, leading to smaller speedup than on a single GPU. However, using the split of the algorithm, where Bi-CGSTAB is computed on one GPU and the preconditioner on the another one, increases the speedup to 15.5 times. Figure 3 shows the real part of the solution for 30 Hz.

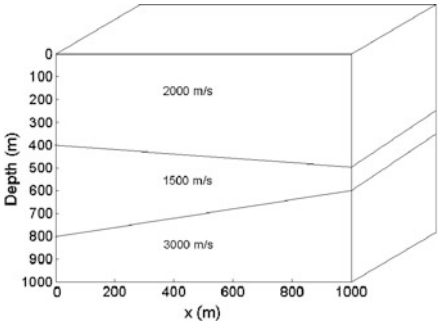

Fig. 2 The velocity profile of the wedge problem

Table 4 Timings for Bi-CGSTAB preconditioned by shifted Laplace multigrid

Size	12-cores/1 GPU Bi-CGSTAB (DP)	12-cores/8 GPUs Preconditioner (SP)	Total	Speedup
12-cores	94 s	690 s	784 s	1
1 GPU	13 s	47 s	60 s	13.1
8 GPUs	83 s	86 s	169 s	4.6
2 GPUs+split	12 s	38 s	50 s	15.5

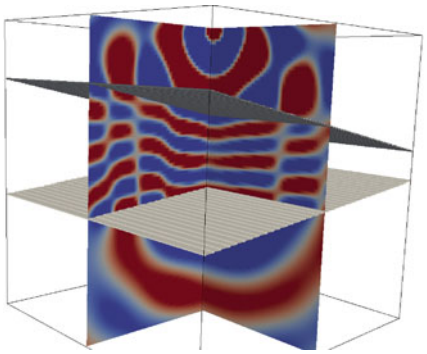

Fig. 3 Real part of the solution, $f = 30\,\text{Hz}$

6 Conclusions

In this paper we presented a multi-GPU implementation of the Bi-CGSTAB solver preconditioned by a shifted Laplace multigrid method for a three-dimentional Helmholtz equation. To keep the double precision convergence the Bi-CGSTAB method is implemented on GPU in double precision and the preconditioner in

single precision. We have compared the multi-GPU implementation to a single-GPU and a multi-threaded CPU implementation on a realistic problem size. Two multi-GPU approaches have been considered: data parallel approach and a split of the algorithm. For the data parallel approach, we were able to solve larger problems than on one GPU and get a better performance than multi-threaded CPU implementation. However due to the communication between GPUs and a CPU the resulting speedups have been considerably smaller compared to the single-GPU implementation. To minimize the communication but still be able to solve large problems we have introduced split of the algorithm. In this case the speedup on multi-GPUs is similar to the single GPU compared to the multi-core implementation.

The autors thank NVIDIA Corporation for access to the latest many-core-multi-GPU architecture.

References

1. J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia (1991).
2. B. Engquist and A. Majda. Absorbing boundary conditions for numerical simulation of waves. *Math. Comput.*, 31:629–651 (1977).
3. Y. A. Erlangga, C. W. Oosterlee, and C. Vuik. A novel multigrid based preconditioner for heterogeneous Helmholtz problems. *SIAM J. Sci. Comput.*, 27:1471–1492 (2006).
4. O. Ernst and M. Gander. Why it is difficult to solve Helmholtz problems with classical iterative methods. In *Durham Symposium 2010* (2010).
5. H. Knibbe, C. W. Oosterlee, and C. Vuik. GPU implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method. *Journal of Computational and Applied Mathematics*, 236:281–293 (2011).
6. www.nvidia.com (2011).
7. E. Zhebel. *A Multigrid Method with Matrix-Dependent Transfer Operators for 3D Diffusion Problems with Jump Coefficients*. PhD thesis, Technical University Bergakademie Freiberg, Germany (2006).