Speeding up a marine 3D CSEM code with GPU

M. Sommer[1], S. Hölz[1], M. Moorkamp[2], A. Swidinsky[1], B. Heincke[1], C. Scholl[3] and M. Jegen[1]
[1]GEOMAR, Helmholtz Centre for Ocean Research Kiel, Wischhofstr. 1-3, 24148 Kiel, Germany
[2]University of Leicester, Department of Geology, University Road, Leicester, LE1 7RH, UK
[3]Fugro, Fugro Electro Magnetic GmbH, Wolfener Strasse 36V, 12681 Berlin, Germany

SUMMARY

One main problem in 3D time domain controlled source electromagnetic (CSEM) inversion is the high runtime of forward modeling codes. We reduced the runtime of the 3D time domain finite difference CSEM code TEMDDD by the GPU-parallelization of expensive algorithms. The code solves the electromagnetic diffusion equation by discretization of spatial operators and subsequent calculation of eigenpairs. These eigenpairs are found by approximation of the eigenspace in a Krylov subspace using the Spectral Lanczos Decomposition Method. This algorithm was not parallelizable in its original form due to implementation of the surface boundary condition. We show for the marine case that replacing the original boundary condition at the air-sea surface by a discretized air layer allows GPU parallelization of every time consuming algorithm of the code. Speedups between 20 to 60 have been achieved compared to the original code for a large 3-D model. Currently, we are developing a 3D-inversion based on this code.

Keywords: forward modeling, marine CSEM, GPU

INTRODUCTION

The setup of the marine CSEM system at GEOMAR is demonstrated in Fig.1. The receivers are arranged as an array on the seafloor measuring both horizontal E-field components.

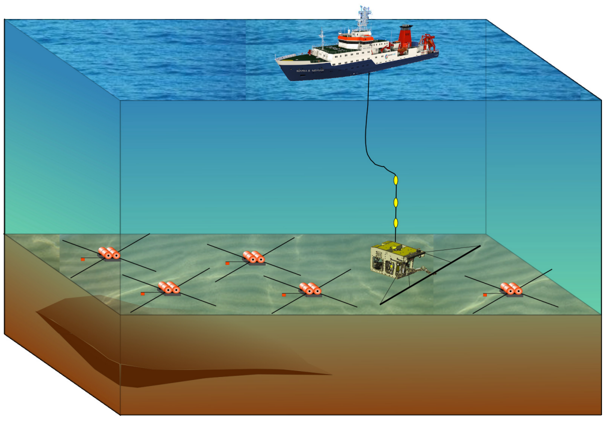

Figure 1. Design of marine CSEM-system from GEOMAR. A Source field is emitted from a dipole antenna, which is mounted on an ROV. Horizontal field components are measured at a number of receivers that are deployed at the seafloor.

Unlike most CSEM setups, where a source is dragged through the water, here it is a 10m long dipole mounted on a remotely operated vehicle (ROV). Transient, rectangular signals are sent with periods of four to 40 seconds with currents up to 70 ampere. Several datasets have already been gathered by this approach.

Currently, we are developing an inversion algorithm to handle the datasets. One main problem in an inversion is the high runtime of the forward code. In the scope of this work we investigate the potential for parallelization of the time domain code TEMDDD that was originally designed for modeling of land based EM-surveys. For average sized 3-D models this code has runtimes of 20 minutes ($50 \times 50 \times 40$ grid cells) up to one hour ($70 \times 70 \times 50$ grid cells) on one standard CPU kernel for a single forward calculation. One way to speedup a code is to use Graphics Processing Units (GPUs). GPUs, originally developed for graphic rendering, have a massively parallel structure and have a high computational power that can also be used for scientific computing.

In this paper we investigate possibilities to speed up the most time consuming algorithms of TEMDDD. We show that for the marine case, the two most time consuming algorithms can be skipped because the seawater layer above the source and receivers allows a simplification of the water-air boundary condition. Another advantage of such a modified boundary is that the remaining algorithms can be speeded up by implementating GPU parallelization, which was not possible before. By means of synthetic models we demonstrate that the runtime for typical model sizes can be reduced drastically by these modifications.

THE FD TIME DOMAIN CODE

The original code, developed by Knútur Árnason, solves the electromagnetic diffusion equation

$$\nabla \times \nabla \times (\sigma\mu)^{-1}\vec{E} = -\frac{\partial \vec{E}}{\partial t} \tag{1}$$

with the initial value $\vec{E}(t=0) = \vec{E}_0$. The spatial operators becomes discretized with central finite differences

$$\mathbf{A}\vec{E} = -\frac{\partial \vec{E}}{\partial t} \tag{2}$$

With the exponential ansatz

$$\vec{E} = \vec{E}_0 exp(-t\mathbf{A}) = \sum_{i=1}^{n} E_{0i} exp(-t\lambda_i)\vec{z}_i. \tag{3}$$

the problem reduces to an eigenproblem, where the integer n is the number of rows of $\mathbf{A}$, E_{0i} is the i'th components of $\vec{E}_0$, λ_i is the i'th eigenvalue of $\mathbf{A}$ and $\vec{z}_i$ the corresponding eigenvector. Because of the size and sparsity of $\mathbf{A}$, its eigenpairs are found by reduction of its eigenspace by projection with a Krylov subspace basis. This is done by the spectral Lanczos decomposotion method (SLDM) $\mathbf{A} \approx \mathbf{QHQ}^T$, with $\mathbf{H} \in \mathbb{R}^{\mathbf{k}} \times \mathbb{R}^{\mathbf{k}}$ is a tridiagonal matrix and its eigenvalues are the first k of $\mathbf{A}$ and the columns of $\mathbf{Q}$ are the orthogonalized Krylov vectors $\mathbf{K}^k = span\{\vec{E}_0, \mathbf{A}\vec{E}_0, \mathbf{A}^2\vec{E}_0, \ldots, \mathbf{A}^{k-1}\vec{E}_0\}$.
Critical are the electric field vectors at the air boundary, because the high conductivity contrast causes numerical instability. Deriving eq. 1 with $\sigma = 0$ results in the Laplace equation such that upward continuation is possible. To solve the Laplace equation, a Householder tridiagonalization, an eigensolver and a convolution in the SLDM are required.

SPEEDING UP THE CODE

Three algorithms turned out to be computationally expensive: 1) the Householder tridiagonalization used to solve the discretized Laplace operator, 2) the SLDM and 3) the tridiagonal symmetric full eigensolver, which is performed twice; for the surface matrix and the tridiagonal matrix gained by SLDM.
In the marine case, the Laplacian boundary condition can be replaced by a discretized air layer, for sufficient water depth. This allows to skip the expensive Householder tridiagonalization as well as one eigensolver. This also allows us to massively parallelize the SLDM on a graphics device. Its computationally most expensive part is a sparse matrix times vector multiplication, which is well parallelizable. The eigensolver required for the matrix $\mathbf{H}$ can be replaced by the eigensolver of the CULA (CUDA LAPACK) library.

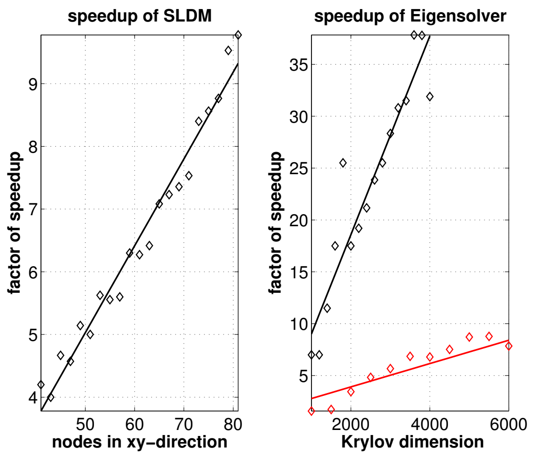

Figure 2. Speedups of SLDM calculations (left, GPU-version relative to original CPU-version) and the QR eigensolver (right, CULA-QR-Eigensolver relative to LAPACK QR-algorithm. A linear trend (black lines) is evident for both algorithms.

Fig. 2 shows the speedup for both algorithms. As a graphics device, the GeForce GTX 275, has been compared with an Intel i7 with 2.80GHz. The speedup of the SLDM mainly depends on the number of nodes of the grid in xy-direction. It ranges from four to ten for 40 to 80 cells in xy-direction. It has been benchmarked on a single core. The speedup of the eigensolver, here a QR-algorithm, only depends on Krylov dimension. It ranges up to ten. Note that this benchmark has been compared with a CPU optimized LAPACK QR-algorithm parallelized on four cores.

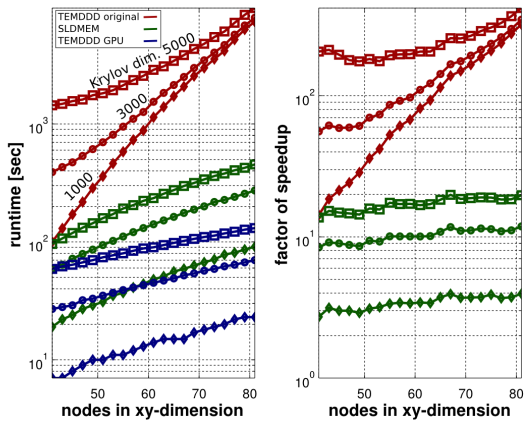

Figure 3. (Left) total runtimes of original code (red) and the modified GPU-parallelized code (blue). Models for the parallelized code include additional grid cells in z-direction for the air layer. (Right) total speedup of modified relative to original code. Krylov dimensions are indicated directly on the curves. Note that speedups are overestimated, since GPU-parallelized code requires use of higher Krylov dimensions for numerically accurate results.

While the Krylov dimension has only a minor impact on the original code's runtime for a large number of cells in the horizontal directions, it is increasingly important for the new code's runtime (Fig. 3, left). For models sizes of $70 \times 70 \times 50$ up to $80 \times 80 \times 50$ cells, runtimes of the original code are on the order of one to two hours. For numerically accurate results of the parallelized code, higher Krylov dimensions of 3000 up to 5000 are needed leading to calculation times of about 1 to 1.5 minutes and speedups in the order of 25-60. Additionally, we have also inluded a speed comparison to the highly optimized (but not parallelized) SLDMEM code (s. Fig. 3 green curves) from Druskin & Knizhnermann, which can be considered a standard industry code. Here, speedups fall in a range between 2–20.

CONCLUSION

The attempt to significantly reduce runtime of the 3D time domain CSEM code TEMDDD has been achieved. Air discretization turned out to be a proper method to avoid computationally expensive surface algorithms. SLDM was parallelized with speedups up to a factor of 10, for the eigensolver the CULA-library has speedups up to a factor of 40. Even when using a CPU-parallelized eigensolver (CPU, 4 cores), the CULA library still outperforms it by a factor of 10. This work was done with the aim to have a forward code suitable for 3D-MCSEM inversion, which is our goal in future.

ACKNOWLEDGMENTS

We like to thank Knútur Árnason, for allowing us to use his source-code.