Adaption and GPU based parallelization of the code TEMDDD for the 3D modelling of CSEM data

IFM-GEOMAR
Leibniz-Institut für Meereswissenschaften
an der Universität Kiel

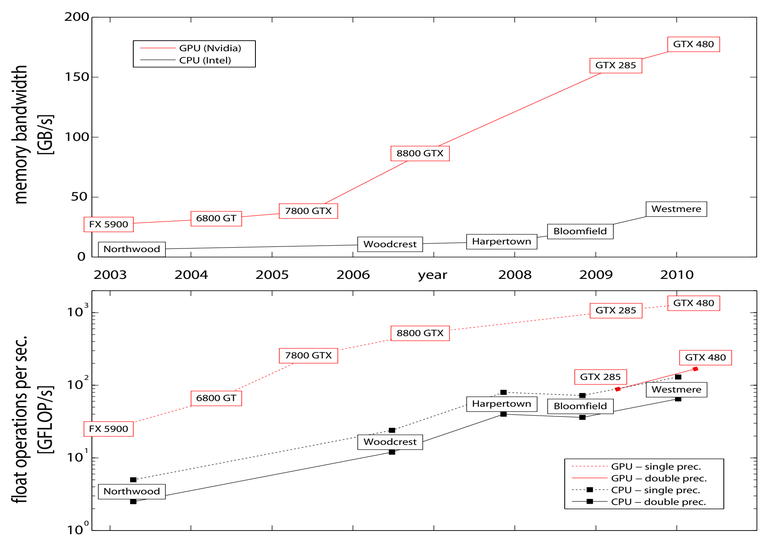
Malte Sommer, Björn Heincke, Max Moorkamp, Marion Jegen, Sebastian Hölz

Abstract

We have analyzed and are currently enhancing the finite difference time domain code TEMDDD [1] for the forward modelling of marine CSEM (controlled source EM) data. Changes in the code were implemented to include EM sources into a conductive medium required to account for marine CSEM geometries. Currently, its performancy is increased using the massively parallel programming language CUDA (Compute Unified Device Architecture) to handle the most expensive numerical operations. These are the Spectral Lanczos Decomposition Method (SLDM, s.[2]), Householder tridiagonalization and the estimation of eigenvalues and eigenvectors. The Householder tridiagonalization as well as parts of the eigensolver have been parallelized successfully. As a result the performance has increased by a factor of 10 to 13 for the considered matrix sizes. Parallelization of SLDM and Inverse Iteration have been realized and are currently implemented into the code. The code will be applied to a 3D data set, which was acquired during a research cruise in the WND-project (West Nile Delta).

GPGPU

GPGPU (General Purpose Computation on Graphics Processing Unit) programming uses the GPU of a graphics card instead of the computer's CPU for computations. In comparison to regular computations on a CPU, GPGPU computations benefit from the GPU's architecture, which is designed for massively parallelization via several multiprocessors (housing multiple stream processors) and a greatly increased memory bandwidth (Fig. 1, top). Currently, a single GPU may handle up to 15360 computational threads in parallel (Nvidia 480 GTX) as compared to a maximum of 16 parallel threads on a CPU (e.g. Intel Xeon X7560), which leads to a greatly increased number of floating point operations per second (fig. 1, bottom).



For this work, the programming language CUDA (Compute Unified Device Architecture), released by NVIDIA in November 2006, has been used and tested on a GeForce GTX 275 graphics card.
Generally, CUDA is a C-type programming environment, which enables the user to manage the data transfer between system- and GPU-memory and to program and initiate computational kernels on the GPU, which carry the main workload of computations in parallel.

Fig. 1: Comparison, how CPUs and GPUs memory bandwidth (top) and FLOPS/s (bottom) increased during the last decade.

TEMDDD

Calculations in TEMDDD, developed from Knútur Árnason [1], are based on Maxwell's equations:

$$\nabla \times H = \epsilon \dot{E} + J \tag{1}$$

$$\nabla \times E = -\dot{B} \tag{2}$$

Assuming isotrope mediums and neglecting displacement currents, these equations reduce to the quasi static approximation:

$$\nabla \times (\nabla \times E) = -\epsilon\mu\ddot{E} - \sigma\mu\dot{E} \approx -\sigma\mu\dot{E} \tag{3}$$

The spatial dependency of this differential equation is formulated via finite differences on a staggered grid [7]. Generally, the formulation on a staggered grid, in which the electrical conductivity σ is discretized on a rectilinear grid and is taken to be constant within grid cells, yields a system matrix $\mathbf{A}$, which relates each component of the E-field on a grid node to field components at adjacent grid nodes:

$$\nabla \times (\nabla \times E) \rightarrow \mathbf{A}E = -\dot{E} \tag{4}$$

Assuming initial conditions to be $E(t=0) = E_0$, this matrix equation system describes exponential decay modes of the initial field, which may be spectrally decomposed:

$$E = exp(-t\mathbf{A})E_0 = \sum_{i=1}^{n} E_{0i} exp(-t\lambda_i) z_i \tag{5}$$

Here, λ_i are the eigenvalues and z_i the corresponding eigenvectors of $\mathbf{A}$. To solve this eigenvalue problem, the large, sparse matrix $\mathbf{A}$ is reduced to a small, tridiagonal matrix similar to $\mathbf{A}$ by the Spectral Lanczos Decomposition Method (SLDM) [5]. Consecutively TEMDDD solves the eigensystem via SVD. The boundary conditions at the surface are incorporated via Laplacian upward continuation:

$$E(-z) = \mathbf{A_s} E_0 \tag{6}$$

It's solution can be found by spectral decomposition of the Laplace operator, discretised in the same way mentioned above:

$$a_{ij} = \sum_{k}^{n} z_{\lambda_k}(i) exp(-t\lambda_k) z_{\lambda_k}(j) \tag{7}$$

The spectrum is obtained by Householder tridiagonalization and SVD.

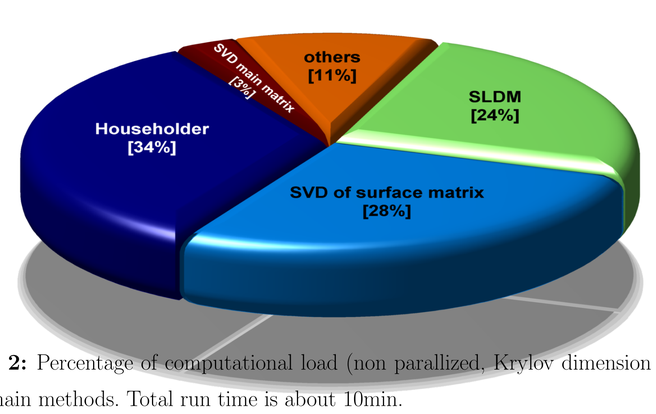
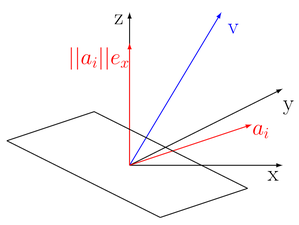


Fig. 2: Percentage of computational load (non parallized, Krylov dimension of 1000) for main methods. Total run time is about 10min.

Householder

Householder tridiagonalization reflects the coulumn vectors of a matrix on its coordinate axes to zero out certain components.



The Householder matrix, performing such mapping, is:

$$\mathbf{H} = \mathbf{I} - \frac{2}{v^T v} v v^T \tag{8}$$

whereby v is normal to the reflecting plane. Performing it for every column yields:

$$\mathbf{Q} = \mathbf{H_1 H_2} \dots \mathbf{H_n} \tag{9}$$

That tridiagonalizes $\mathbf{A}$ in case of symmetry:

$$\mathbf{Q A Q}^T = \mathbf{T} \tag{10}$$

Eigenvalues remain and Eigenvectors can be found by $\mathbf{Q^{-1}}z = \mathbf{Q^T}z$ backtransformation, performed via **WY-factorization**. Expensive operations are: matrix-vector-products, dyadic products, scalar products, matrix-vector additions. Householder tridiagonalization is suitable for parallelization, because all operations are independent except for the scalar product ($\approx \sqrt{n}$ cycles).

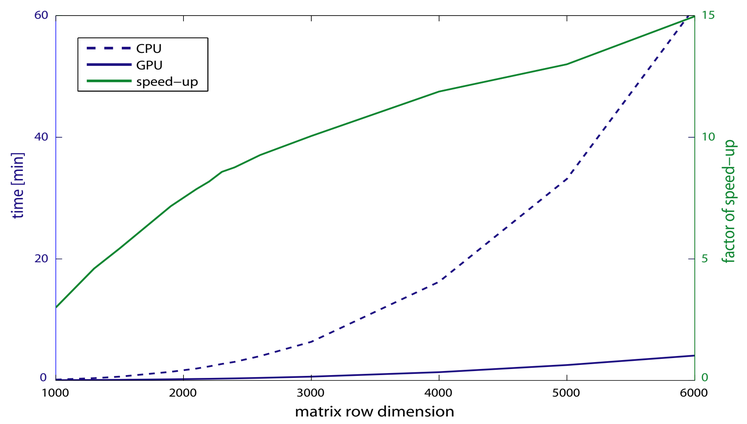


Fig. 3: Comparison of serial CPU Householder tridiagonalization (blue, dashed) with GPU based one (blue, solid). For real geometries, matrix row dimensions of 2500 up to 5000 are encountered, resulting in speed-ups by factors 10-13x.

Eigenproblem and SLDM

In the serial code, the eigenpairs are estimated via QR decomposition. On parallel architectures, bisection with consecutive eigenvector estimation by inverse iteration is much more powerful. **Bisection** on Gershgorin intervals allows solving each eigenvalue independent from the others and is therefore suitable for parallel architectures. **Inverse Iteration** solves $(\mathbf{A} - \lambda\mathbf{I})z = q, q \rightarrow z$ iteratively by **cyclic reduction**.

The Spectral Lanczos Decomposition Method (SLDM) is a Krylov subspace method, that reduces $\mathbf{A}$ to a tridiagonal Matrix:

$$\mathbf{H} = \begin{pmatrix} \alpha_1 & \beta_1 & \dots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \beta_{m-1} & \alpha_m \end{pmatrix} \tag{11}$$

of a desired rank by the orthonormalizing recursion:

$$\beta_i q_{i+1} = \mathbf{A}q_i - \beta_{i-1}q_{i-1} - \alpha_i q_i \tag{12}$$

$$\alpha_i = q_i * \mathbf{A}q_i, \beta_i = \|\mathbf{A}q_i - \beta_{i-1}q_{i-1} - \alpha_i q_i\| \tag{13}$$

$$\beta_0 q_0 = 0, q_1 = \frac{E_0}{\|E_0\|} \tag{14}$$

Those operations can be parallelized directly.
Some problems remain with the stability of the inverse iteration and the SLDM. Therefore, no reliable estimates for speed-up can be given yet.

Literature

[1] Knútur Árnason, "Consistent Discretization of Electromagnetic Fields and Transient Modeling", *Three-Dimensional Electromagnetics, SEG,pp. 103-118* 1999.

[2] V. Druskin & L. Knizhnerman, "Spectral approach to solving three-dimensional Maxwell's diffusion equations in the time and frequency domains", *Radio Science* 1994.

[3] G. H. Golub & Charles F. van Loan, "Matrix Computations - Third Edition", *The Johns Hopkins University Press* 1996.

[4] "CUDA C Programming Guide", *NVIDIA Corporation* 2010.

[5] B. N. Parlett, "The Symmetric Eigenvalue Problem", *SIAM* 1980.

[6] H. Y. Chang, 'A parallel Householder Tridiagonalization stratagem using scattered row decomposition", *International Journal for numerical methods in engeneering* 1988.

[7] Kane S. Yee, 'Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media", *IEEE Transactions on antennas and propagation* 1966.