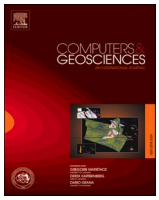
Computers & Geosciences 184 (2024) 105518

Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo



Hybrid CPU-GPU solution to regularized divergence-free curl-curl equations for electromagnetic inversion problems

Hao Dong [a,*], Kai Sun [b], Gary Egbert [c], Anna Kelbert [d], Naser Meqbel [e,f]

[a] China University of Geosciences, Beijing, China
[b] NVIDIA Corporation, Beijing, China
[c] Oregon State University, Corvallis, OR, USA
[d] United States Geological Survey, Golden, CO, USA
[e] 3D Consulting-GEO GmbH, Berlin, Germany
[f] Observatório Nacional, RJ, Brazil

ARTICLE INFO

Keywords:
Curl-curl equations
EM geophysics
GPU
Divergence-free
Inversions

ABSTRACT

The Curl-Curl equation is the foundation of time-harmonic electromagnetic (EM) problems in geophysics. The efficiency of its solution is key to EM simulations, accounting for over 95% of the computation cost in geophysical inversions for magnetotelluric or controlled-source EM problems. However, most published EM inversion codes are still central processing unit (CPU)-based and cannot utilize recent computational developments on the graphic processing units (GPUs). Based on a previously proposed divergence-free algorithm developed on CPUs, this study demonstrates the current limits of the CPU-based inversion procedure. To exploit the high throughput capability of GPUs, we propose a hybrid CPU-GPU framework to solve forward and adjoint problems required for EM inversions. The large sparse linear systems arising from the staggered-grid finite difference approximation of the Curl-Curl equation are solved with a mixed-precision Krylov subspace solver implemented on a GPU. The algorithm is then tested in EM forward and adjoint calculations, with real-world three-dimensional numerical examples. Test results show promising 30× kernel-level speed-ups over the conventional CPU algorithm. This approach may further take the complex frequency domain EM inversions onto the next, practical stage on small affordable GPU platforms.

1. Introduction

The Curl-Curl equations define the core underlying physics of most frequency-domain electromagnetic (EM) geophysical problems, including magnetotellurics (MT) and controlled source electromagnetics (CSEM). Accurate and rapid solution of these equations is therefore the key to practical EM simulations and inversions, especially in complex three-dimensional (3D) cases. The Curl-Curl equations are commonly discretized with finite element or finite difference approximations leading to large sparse linear systems, which need to be solved efficiently in forward and adjoint calculations.

It is well known that most basic operations required for solving large sparse linear systems, like matrix-vector multiplication or scaled vector addition, are memory bandwidth-bound (e.g., Boland and Constantinides, 2011). In other words, the efficiency of those computations in EM forward/inversion problems is not limited by the number of floating-point operations per second (FLOPS) of the computer central processing unit (CPU), but by the speed of data throughput of the computer memory. Previous studies have indicated that a few CPU cores in the computation may be enough to saturate the memory bandwidth, so using more physical CPU cores may only achieve marginal speed-up (Rogers et al., 2009). While modern computation nodes may contain 48 to 128 physical CPU cores, the memory bus between the shared memory and the CPU are normally limited to 4–16 channels, which restricts the total memory throughput of the machine. This is sometimes referred as the "bandwidth wall" or "memory wall" (Rogers et al., 2009). Upon reaching this wall, concurrent accesses from multiple CPU cores must compete for the limited memory bandwidth, resulting in the deterioration of per-core performance when running bandwidth-bound tasks.

As such, the performance of the overall memory throughput for CPU platforms can only be effectively extended by adding more interconnected shared-memory machines. Modern supercomputers and

* Corresponding author.
E-mail address: donghao@cugb.edu.cn (H. Dong).

https://doi.org/10.1016/j.cageo.2024.105518
Received 19 September 2023; Received in revised form 7 December 2023; Accepted 3 January 2024
Available online 13 January 2024

clusters may consist of tens to hundreds of computational nodes, which is costly in both financial and ecological terms. Alternatively, graphic processing units (GPUs) have become an inexpensive solution to offer high memory throughput (Owens et al., 2007) for partial differential equation (PDE) problems. Modern professional GPUs can provide memory bandwidth up to a few terabytes per second, which is comparable to that of several conventional high performance CPU nodes. Machines equipped with GPUs may therefore become potential platforms for large-scale scientific computation problems, including 3D EM modeling.

Indeed, numerical computations using GPUs have become increasingly important in scientific high-performance computation and machine learning studies (Mudigere et al., 2022; Wang et al., 2021). Specifically, GPU accelerations have long played an essential role in large-scale geophysical computations, such as the seismic reverse time migration (e.g., Foltinek et al., 2009), full wave inversion (Yang et al., 2015) and 3D gravity inversion (Shangbin Liu et al., 2022). In EM geophysics, Xiao and Liu (2015) proposed a GPU-based Gaussian elimination method to solve the forward and sensitivity problems in two-dimensional (2D) MT Occam inversion. GPUs are also successfully utilized to accelerate the direct solution in 3D transient EM simulations (Sheng Liu et al., 2022) and magnetotellurics (Varılsüha, 2020). However, the acceleration of GPUs in real-world 3D electromagnetic inversions has not been extensively discussed, probably due to the scale of the problem and the representation of frequency domain EM governing equations in complex form.

In this article, we introduce a hybrid CPU-GPU approach for the EM forward/adjoint problem, based upon the divergence-free formulation of the Curl-Curl equations (Dong and Egbert, 2019). We use the Compute Unified Device Architecture (CUDA) to implement a GPU-based mixed-precision algorithm to efficiently solve the regularized Curl-Curl equations arising from the magnetotellurics and controlled-source problems. To overcome the effects of the CPU overheads, we also design a non-preemptive scheduling scheme to reduce the GPU wait phase, which helps to speed up the parallel calculation. To manage the computational resources with multiple CPUs and GPUs, we implement a hybrid approach with a two-layered Message Passing Interface (MPI, Gropp and Lusk, 1994) parallel inversion framework. Finally, through 3D numerical examples from real-world cases, we demonstrate the substantial efficiency enhancement that our new approach may add to complex EM problems on small-scale GPU platforms.

2. Divergence-free curl-curl equations

For most frequency-domain EM problems like MT and CSEM, the time-harmonic Maxwell's equations can be expressed as:

$$\begin{aligned} \nabla \times \boldsymbol{E} &= -i\omega\mu\boldsymbol{H} \\ \nabla \times \boldsymbol{H} &= i\omega\varepsilon\boldsymbol{E} + \boldsymbol{J} + \boldsymbol{J}^{ext} \end{aligned} \tag{1}$$

where E and H are the electric and magnetic fields, ω is angular frequency, ε is the electric permittivity of the model domain, μ is the permeability. $\boldsymbol{J} = \sigma\boldsymbol{E}$ denotes the electric conduction currents in the domain, where σ is the electrical conductivity. And $\boldsymbol{J}^{ext}$ stands for the external current forcing. Equation (1) in the quasi-static approximation (displacement currents are negligible) can be reduced to a second-order Curl-Curl problem based on electric fields:

$$\nabla \times \nabla \times \boldsymbol{E} + i\omega\mu\sigma\boldsymbol{E} = \boldsymbol{J}^{ext}. \tag{2}$$

However, when the angular frequency ω in the second term of Eq. (2) is small (i.e., at longer periods) or σ vanishes (in the air), any field from any electrical potential $\boldsymbol{\varphi}$ ($\boldsymbol{E}_p = \nabla\varphi$, which is annihilated by the Curl-Curl operator) can be added to the actual solution. These spurious modes (Jiang et al., 1996) can result in inaccurate solutions and slow convergence of iterative solvers. To suppress the spurious modes, divergence-free conditions have been commonly used in EM geophysical modelling and inversion codes to improve the efficiency and accuracy of numerical solutions, with the "divergence correction" method (Mackie et al., 1994; Smith, 1996) or with augmented unknowns like magnetic/electrical potentials (Schwarzbach, 2009; Weiss, 2013) most common. An alternative approach is to impose the divergence condition on current density

$$\nabla \cdot \boldsymbol{J} = 0, \tag{3}$$

by essentially adding this constraint to the Curl-Curl equation. First introduced by Weiland (1985), a gradient-divergence (Grad-Div) term can be applied by taking the gradient of Eq. (3) and adding it to Eq. (2) (with appropriate scaling). This yields an equation which simultaneously enforces the Curl-Curl equations and the divergence condition (hereby referred to as CCGD formulation, i.e., Curl–Curl equations modified by Grad-Div operator). Dong and Egbert (2019) showed that the specific form

$$\nabla \times \nabla \times \boldsymbol{E} - \nabla\sigma^{-1}(\nabla \cdot \sigma\boldsymbol{E}) + i\omega\mu\sigma\boldsymbol{E} = \boldsymbol{J}^{ext} \tag{4}$$

can efficiently deflate the null-space of the curl operator, ensure a unique solution, and accelerate convergence of iterative solvers (also refer to Mackie and Watts, 2012; Weaver et al., 1999).

In MT, the source is commonly provided on the boundaries of the computational domain, e.g., obtained through a series of 2D forward problems (as in Siripunvaraporn and Egbert, 2009). For the CSEM problem in the frequency domain, sources are internal to the domain and are usually treated with a secondary field formulation (e.g., Wait, 1982; Zhdanov and Keller, 1994). In all cases the modified forward modeling problem in Eq. (4) can be discretized with a staggered grid finite difference formulation as:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{5}$$

and solved numerically (Sadiku, 2000). Here $\boldsymbol{A}$ is the sparse matrix, $\boldsymbol{x}$ is the solution vector, while $\boldsymbol{b}$ is the right-hand side vector (RHS). Similarly, the sensitivity information required in the inversion can be obtained with "adjoint" solutions, using essentially the same set of equations, but with an appropriately modified right-hand side. Refer to section 3 in Dong and Egbert (2019) for details.

The linear system (Eq. (5)) arising from the Curl-Curl problem is often solved using a Krylov Subspace (KSP) or direct solver (DS; lower and upper triangular (LU) decomposition and Gaussian elimination) method. The solution can then be used to calculate the transfer functions to be used in the evaluation of inversion objective functions. The full LU factorization required by the direct solvers mostly utilizes the supernodal strategy (Rothberg and Gupta, 1990), which ends up requiring (potentially large) direct factorizations of dense matrices at the end of the computation. As such, the efficiency of the direct solvers is not memory bandwidth-bound and may utilize more CPUs than iterative methods (Kordy et al., 2016; Yang et al., 2017).

However, the time and computational resource costs of the DS approach can be orders of magnitudes higher than the iterative KSP methods, requiring specialized large scale computational infrastructure. As such, the KSP method is often preferred in large scale 3D computational geophysics problems (Mackie et al., 2001; Newman, 2014). The conventional divergence correction method periodically restarts the main iteration, which resets the Krylov subspaces and lowers the efficiency of the KSP iterations (Dong and Egbert, 2019). On the other hand, the divergence-free regularization formulation (Eq. (4)) does not require the interruption of the Curl-Curl iterations, leading to a better convergence rate. In the following sections, we will discuss implementation of a solver for the CCGD problem formulation on GPUs in detail.

3. Hybrid CPU-GPU implementations

3.1. Workflow for hybrid forward/adjoint calculations

The hybrid CPU-GPU algorithm to solve the Curl-Curl equations is based upon the conventional CPU version of the CCGD formulation, as in the sparse matrix branch of ModEM (Dong and Egbert, 2019). The program is coded in the Fortran 95 language, which expands the modular framework of the original ModEM software (Kelbert et al., 2014). As with the CPU implementations, the forward/adjoint calculations for time-harmonic problems involve several essential operations or steps (Fig. 1), namely.

1) Assembly/update of the system matrix,
2) Source/boundary conditions to define the right-hand side,
3) Setup of the preconditioning matrices,
4) Solution of the linear system,
5) Calculation of the transfer functions (TFs) from electric field solutions.

With conventional CPU-based algorithms, the solution of the sparse linear system arising from the discrete Curl-Curl equations (step 4) is by far the most computationally intensive component, typically representing more than 95% of the total run time. Accelerating this part of the calculations is thus critical to improving efficiency. As the rest of the operations take up less than 5% of the total time consumption, they are referred as solver "overheads" in the rest of the paper.

The assembly of the CCGD matrix (step 1) requires the building of the basic curl, divergence, and gradient operators to form the Curl-Curl and Grad-Div components in Eq. (4). Additionally, the diagonal matrix term $i\omega\mu\sigma\boldsymbol{E}$ also need to be formed to construct the final system matrix. For each given system of equations (different for each frequency), the preconditioning matrices also needs to be formed, through the incomplete LU (ILU) decomposition (step 3).

The preparation of the RHS (step 2) involves the calculations of the sources (either external for controlled source problems, or reciprocal for adjoint calculations) and boundary values. After the solution of the system equations, the curl operator is applied to the discrete electric fields to compute the magnetic fields. The computed $\boldsymbol{E}$ and $\boldsymbol{H}$ fields are then used to calculate the transfer functions (step 5), which are required for the evaluation of the misfit and computing the gradient of the objective function in an inversion.

Steps 1 and 3 only need to be performed once in a given forward/adjoint problem. The system and decomposed ILU matrices can be stored in the memory for the entire solution procedure. They are only partially updated when the model resistivity and response period changes in an inversion. The calculations for the RHS and TFs also need to be calculated just once for each forward solution. Considering the time consumption of these "overhead" operations compared with the iterative solution of the system, we find that only marginal gains in efficiency could be achieved by implementing these steps with GPUs. Furthermore, all of these other steps involve a great deal of complex computer code, so that conversion would represent a major programming effort. Thus, steps 1, 2, 3, and 5 are kept on the CPU side. As we will demonstrate in the numerical experiments section, keeping these operations on (multiple) CPUs can also help distribute the effects of the time consumption of overheads onto multiple processes.

To summarize, in our hybrid CPU-GPU approach, only the solution of the linear system will be conducted on GPUs, whereas all other operations are performed by CPUs. Therefore, before the GPU phase, the system and preconditioning matrices, as well as the RHS vectors need to be copied to the GPUs; after the solution is obtained, the calculated fields are copied back to the CPU side for the following calculation of TFs. Later in this section we will primarily focus on the details of the implementation for the GPU phase.

3.2. Solution of the linear system

Among the KSP methods, the Quasi-Minimal Residual (QMR) method is widely used in the forward and adjoint calculations in EM inversions (e.g., Kelbert et al., 2014; Siripunvaraporn and Egbert, 2009). However, as QMR is based on the non-symmetric Lanczos process (Chan et al., 1998), it requires two matrix-vector products per iteration ($\boldsymbol{Ax}$ and $\boldsymbol{A}^T\boldsymbol{x}$). The CCGD formulation of the system (Eq. (5)) is no longer symmetric (Dong and Egbert, 2019), leading to extra storage/computation costs associated with the transpose of $\boldsymbol{A}$ in $\boldsymbol{A}^T\boldsymbol{x}$ operation. As such, KSP variants that do not require the $\boldsymbol{A}^T\boldsymbol{x}$ operation in the iterations (e.g., Transpose-Free QMR or Bi-Conjugate Gradient, BiCG) are preferred in this case (Freund, 1993). In this study, we use a preconditioned version

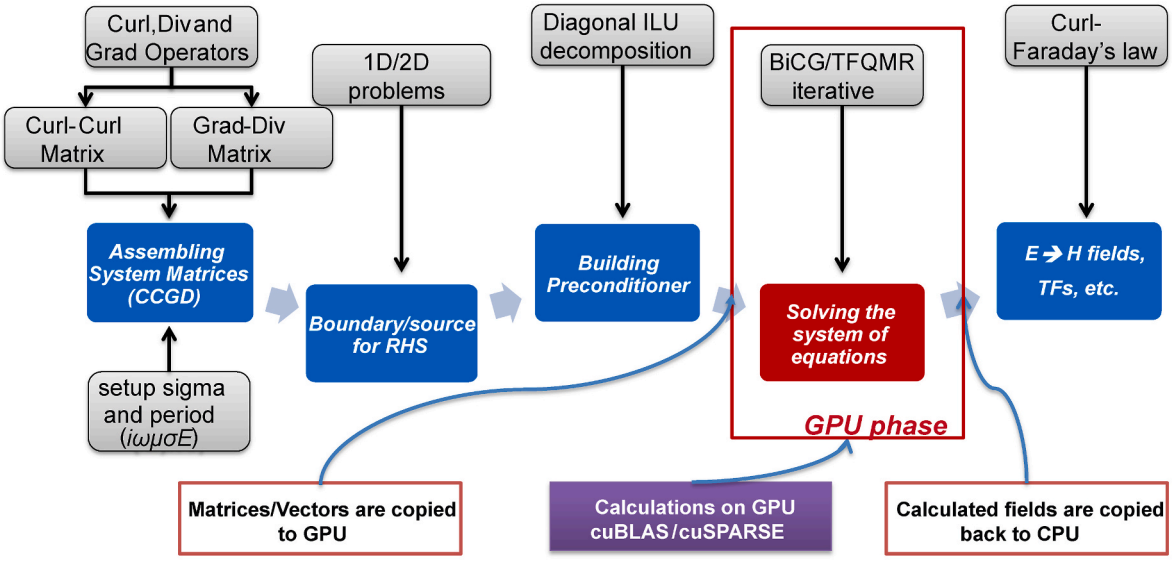


Fig. 1. Flowchart for the major operations in the Hybrid CPU-GPU algorithm for Curl-Curl equations. The red box indicates the GPU phase, whereas the other calculations are done on CPUs. CPU: central processing unit; GPU: graphical processing unit; ILU: incomplete uower and upper triangular matrix; RHS: right-hand side; CCGD: Curl–Curl equations modified by Grad-Div operator; BiCG: bi-conjugate gradient; TFQMR: transpose-free quasi minimal residual; 1D: one-dimensional; 2D: two-dimensional.

of the BiCG (van der Vorst, 1993) for the GPU implementation. The pseudo code (for both CPU and GPU) is shown as follows:

Algorithm 3.1. Preconditioned Bi-Conjugate Gradient Method

a) Initialization:

1) Set $x = initial\ guess$

2) Evaluate initial residual

$r = b - Ax$, (one SpMV operation)

3) if $norm(r) < tol$ then

stop

end

4) $\tilde{r} = r$

b) For n = 1, 2, … , maxiter, do:

1) Evaluate $\rho = \tilde{r} \cdot r$

2) Update p

if n = 1 then

$p = r$

else

$\beta = (\rho/\rho_1)(\alpha/\omega)$

$p = r + \beta(p - \omega v)$

end

3) Preconditioning, solve $\hat{p}$ in $M\hat{p} = p$, (one SpSV operation)

$v = A\hat{p}$, (one SpMV operation)

$\alpha = \rho/(\tilde{r} \cdot v)$

evaluate residual $s = r - \alpha v$

4) Preconditioning, solve $\hat{s}$ in $M\hat{s} = s$, (one SpSV operation)

$t = A\hat{s}$, (one SpMV operation)

$\omega = (t \cdot s)/(t \cdot t)$

evaluate residual $r = s - \omega t$

5) update $x = x + \alpha\hat{p} + \omega\hat{s}$

if $norm(r) < tol$ then

stop

end

update $\rho_1 = \rho$

As shown in the above algorithm, the initial phase (a) of BiCG involves one sparse matrix-vector multiplication of Ax (referred here as sparse matrix-vector multiply (SpMV)) and one reduction operation of norm(r). However, as a typical BiCG solution normally takes tens to a few hundreds of iterations, the time consumption of the initial phase can normally be ignored. The second phase (b) involves two SpMVs and five parallel reduction operations (dot product and two-norm) in a single BiCG iteration. It should also be noted that the ILU preconditioning scheme results in two triangular sparse matrices (L/U). In that case, the first preconditioning procedure in b-3 becomes:

Algorithm 3.2. two-step preconditioning with L/U matrices

Preconditioning, solve p_t in:

$Lp_t = p$, **(one SpSV operation)**

then solve $\hat{p}$ in:

$U\hat{p} = p_t$ **(one SpSV operation)**

which involves two sparse matrix triangular solution (SpSV) operations per preconditioning operation. Likewise, the second preconditioning also turns into two SpSVs. As such, the most computation-intensive operations in a single BiCG iteration reside in the four SpSVs, two SpMVs and five reduction operations. We thus focus on the GPU implementation of these basic operations to improve the efficiency of

the solution.

3.3. Implementation of the GPU solver

As in our CPU algorithm, the GPU implementation utilizes the Compressed Sparse Row (CSR) format to efficiently store sparse matrices like $\boldsymbol{A}$ and $\boldsymbol{L}/\boldsymbol{U}$ in GPU memory. In this study, the solution of the system of equations is carried out through a CUDA implementation of the BiCG solver. The CUDA platform was originally developed by NVIDIA Corporation to support general computation tasks on its line of graphics cards. After years of developments, it has gradually become one of the most important GPU programming frameworks that allow the use of GPUs for highly parallel scientific computational tasks (e.g., Pandey et al., 2022). The native CUDA instructions are an extension of the C/C++ computer language (CUDA-C), where the basic parallel functions (e.g. the scaled vector addition operation, commonly referred as axpy) need to be written as individual "kernels" to work on heterogeneous GPU infrastructures.

Writing efficient GPU kernels is a non-trivial task. Consider the matrix-vector multiplication operation SpMV as an example. In the kernel function, parallel threads should be organized into multiple blocks, where each block is in charge of the multiplication of a number of rows. Each row is then assigned to a group of threads for parallelization (McCourt et al., 2015). Because the sparse matrix arising from the finite difference discretization of Curl-Curl equations contains no more than 13 non-zero elements in each row, it is relatively easy to balance the workload among the threads. However, for maximum efficiency, the kernel should make sure the memory is aligned with threads assigned on different rows for "coalescing" (i.e., to ensure that the address for the memory accessed by different threads is continuous; refer to e.g., Bell and Garland, 2009 for details).

The CUDA libraries, like the cuBLAS and cuSPARSE, provide most basic building blocks for GPU kernels for linear algebra and sparse matrix operations, reducing the need for user written kernel functions. For example, the sparse matrix-vector multiply can be performed by calling the **cusparseSpMV** function, while the complex axpy operation can be realized by the **cublasZaxpy** function. However, the CUDA libraries still lack some basic kernels for the efficient implementation of the BiCG solver. In this study, we exploit the Fortran-C interoperability to develop a handful of custom CUDA-C kernels for operations like the conversions of precision (double to single precision, and vice versa), the Hadamard product ($\circ$) for vectors, or the quaternary scaled vector addition (axpby).

As each GPU card has its own memory, which is not directly addressable by the CPU, the computation on GPUs would naturally involve the communications between the two. The CPU and its attached memory are commonly referred as the *Host*, while the GPU side is referred as the *Device*. As the communication speed from or to the *Device* through the PCIe bus (64 GB/s for PCIe 4.0) can be magnitudes slower than the *Device* memory access ($\sim$1000 GB/s for GDDR6x GPU memory), it is preferable to minimize the host-device communication and let the GPU handle most memory manipulations. As such, we only use the communication to copy necessary matrices and vectors to the *Device*, and then to copy the calculated solution back to the *Host* after KSP iterations are completed. This communication uses the **cudaMemCpy** function in both directions.

To validate the efficiency of our new GPU BiCG solver, a forward modeling comparison was performed with a GPU workstation featuring a single hex-core Intel® Xeon W2133 CPU with four channels of DDR4 2400 memory (76.8 GB/s bandwidth). The GPU is a consumer-grade NVIDIA® RTX 3080Ti graphic card, which features 80 stream multiprocessors and 912 GB/s memory bandwidth. The source codes are compiled with the gfortran compiler from the gcc 9.1 package, with the CUDA 11.4 platform.

We use the 3D inverse model of the Cascadia region, USA (Patro and Egbert, 2008) to conduct the forward modeling experiments. Note that this model and related transfer function files can be found as part of the test dataset included with the publicly available ModEM software package (Kelbert et al., 2014). The entire model domain has the dimension of 1460 $\times$ 1590 $\times$ 550 km, with 78 $\times$ 80 $\times$ 44 discrete mesh cells. The discretized linear equations have about 0.8 million unknowns (DoFs). The forward calculations are performed at 10 periods from 11.64 to 18724.6 s. For both the CPU and GPU algorithms, the tests are conducted with a single CPU core, with the terminating criterion of the BiCG iterations set to 10^{-10} for the relative residual. The calculations for both the CPU and GPU were performed in double precision to ensure the accuracy of the solution.

Fig. 2 shows the comparison of the time consumption for the Cascadia forward calculation with our new GPU algorithm, relative to the CPU algorithm described in Dong and Egbert (2019). The wall-time is calculated using results averaged from three runs for two polarizations (XY/YX). Both the GPU and CPU results show a general trend of the increasing wall-time as the data periods become longer, which is due to weakening of the divergence constraint as the angular frequency ω vanishes. On the other hand, it is apparent that the computation time of the GPU solver is substantially reduced from about tens of seconds to $\sim$1.4 s, when compared with the CPU counterpart. This translates into a speed-up of about 20$\times$ (Fig. 2). On the other hand, the time consumption of the overheads (e.g., the assembly of the system matrix, calculation of boundary conditions) remains similar for the two. This is reasonable as the overhead operations are conducted on CPUs only for both approaches. As the time consumption of an EM inversion is commonly dominated by the forward/adjoint calculations, our new hybrid CPU-GPU implementation has the potential to accelerate the

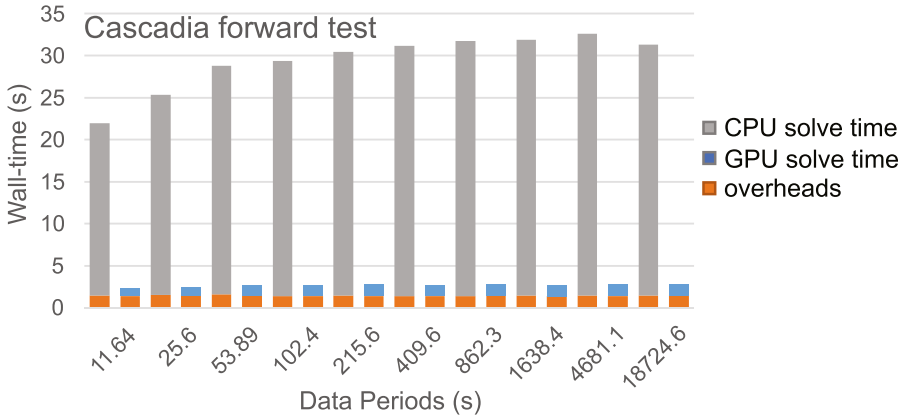


Fig. 2. Comparison on the Cascadia forward test performance for CPU and GPU solvers at different periods. CPU: central processing unit; GPU: graphical processing unit.

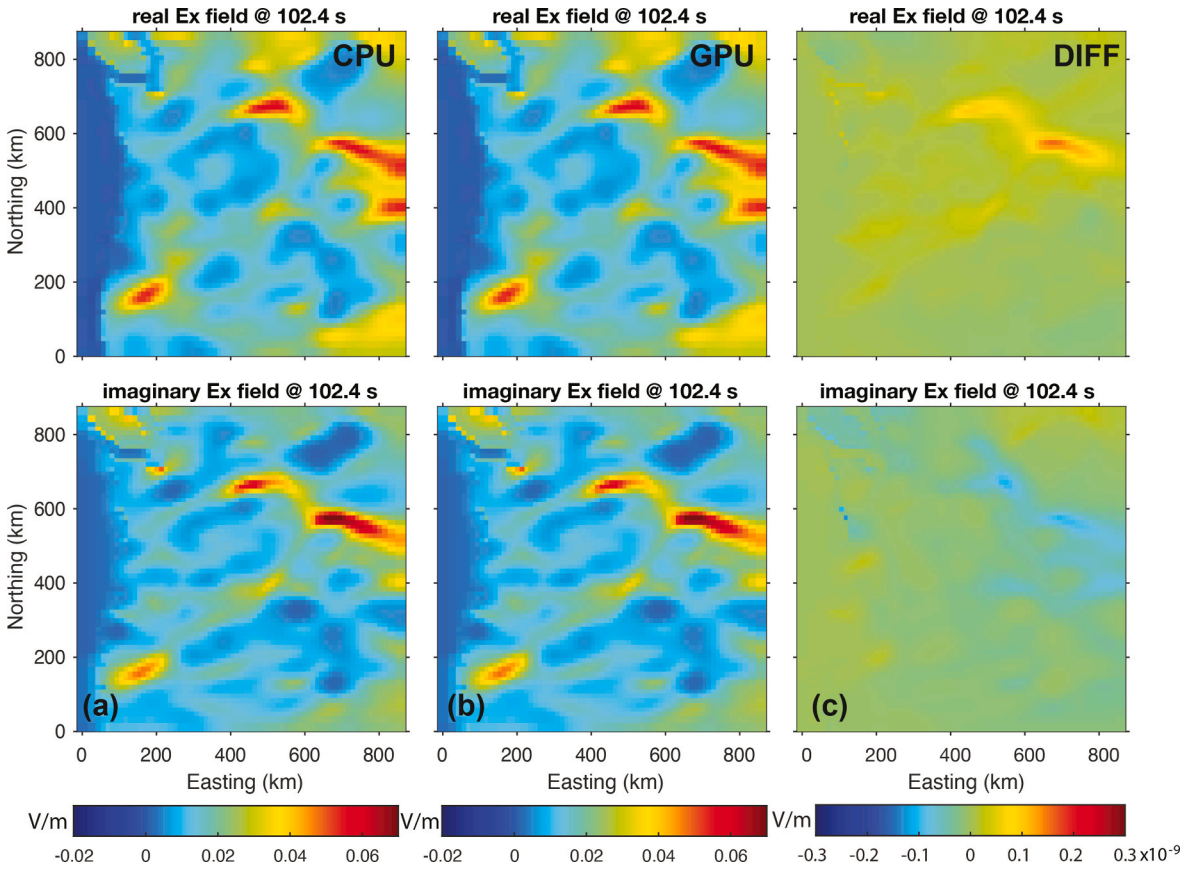


Fig. 3. A comparison on the surface electrical solutions (Ex) in the central part of the Cascadia model with XY polarization at a period of 102.4 s from (a) the CPU solver (Dong and Egbert, 2019) and (b) the GPU solver (this study). Panel (c) shows the solution difference of a–b. CPU: central processing unit; GPU: graphical processing unit; DIFF: differences.

inversions.

Although the performance of the new GPU solver is encouraging, it can be observed that the GPU solution may not have identical results to the CPU counterpart. This is due to the nature of fine-grain parallel mechanisms in GPUs. Taking the GPU matrix-vector multiplication as an example, the dot product between a matrix row and the vector may be calculated in parallel. In this case, the point-by-point products of the corresponding elements are performed on different threads before those products are accumulated to form an element of the resulting vector. However, the order of this accumulation is likely to be different for each run due to the asynchronous execution of the threads in parallel. As such, different sequences of the summation may lead to non-deterministic results because the floating-point arithmetic may not follow the law of associativity (e.g., Bell and Garland, 2009).

In light of this, we compare the solutions from our GPU and CPU solvers to demonstrate the possible divergence. As shown in Fig. 3c, the calculated Cascadia surface electric field differences from the CPU and GPU solvers are merely within a fraction of nanovolt per meter, which translates into a tiny relative difference of order 5×10^{-9}. Hence, we believe that the forward results from our new hybrid CPU-GPU approach are effectively identical to that from the traditional CPU-only ones.

4. Mixed precision considerations

With the promising results shown in the previous section, we consider improvements of the basic GPU algorithms to further accelerate EM forward modelling. One popular idea to accelerate the numerical PDE computations is the so-called mixed precision method (Higham and Mary, 2022). The method involves replacing double-precision (FP64) operations with single-precision (FP32) ones or even lower precision

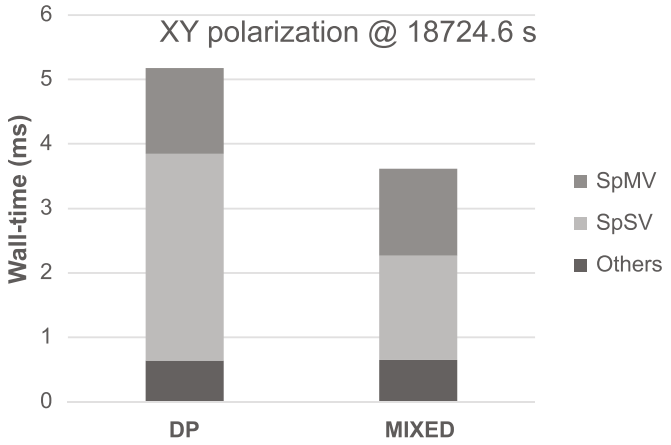


Fig. 4. Comparison of mixed and double precision GPU solver performance for an averaged single BiCG iteration with XY polarization at the period of 18724.6 s in the Cascadia forward calculation. DP: double precision; MIXED: mixed precision; GPU: graphical processing unit; BiCG: bi-conjugate gradient; SpMV: sparse matrix-vector multiply; SpSV: sparse matrix triangular solution.

counterparts (e.g., FP16) in some stages of the calculation (Carson and Higham, 2018). It may lead to higher performance due to the reduction of total memory manipulations in the calculations.

Moreover, while professional GPUs fully support double-precision calculations, in consumer-grade GPUs most processing units are single-precision. This directly leads to a higher peak performance for single precision operations than double precision ones, which also

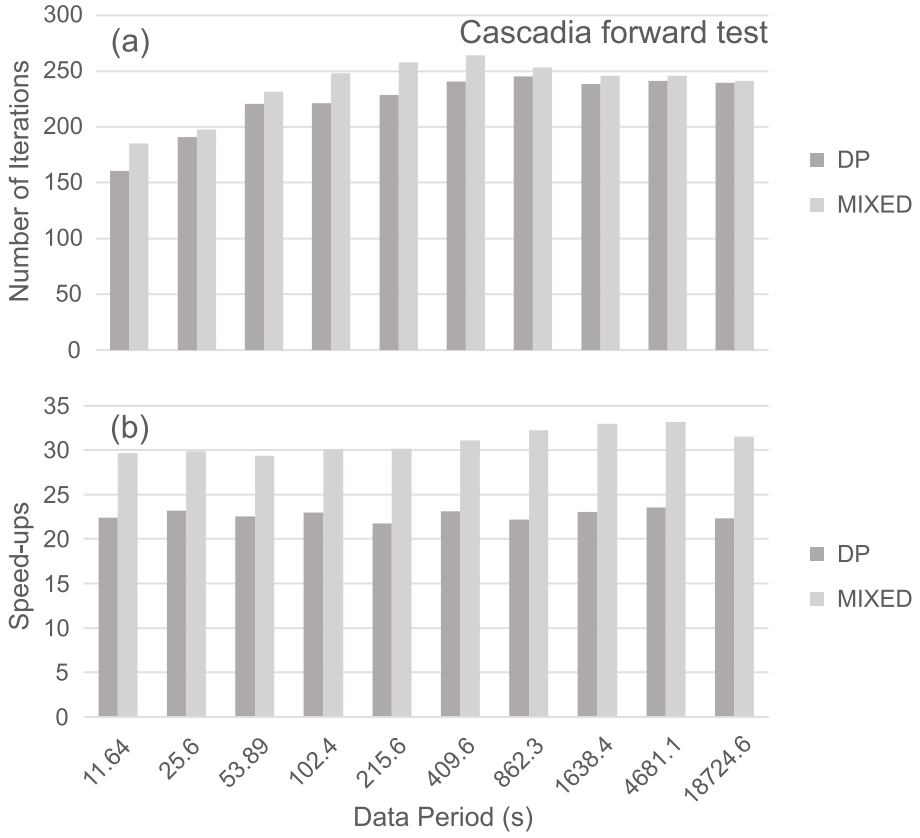


Fig. 5. Comparison on the Cascadia forward test (a) iterations and (b) speed-up times for double precision and mixed precision GPU solvers. DP: double precision; MIXED: mixed precision; GPU: graphical processing unit.

favors mixed-precision methods for consumer-grade GPUs.

However, replacing the FP64 calculations with FP32/16 counterparts may naturally lead to loss of solution precision due to increased truncation error. This may result in slow or no convergence in iterative solvers. On the other hand, in preconditioned KSP methods, as the preconditioning matrix $\boldsymbol{P}$ only needs to approximate the system equation operator $\boldsymbol{A}$, this can be stored and applied in reduced precision without sacrificing too much on the convergence rate. Following the idea of mixed-precision preconditioning proposed by Carson et al. (2022), we only perform the four SpSV operations in single precision but keep all other calculations in double precision. This involves the conversion of the $\boldsymbol{L}/\boldsymbol{U}$ matrices to single precision on the *Device*. Also, the relevant vectors stored on the *Device* need to be converted to single precision before the preconditioning step (and back to double after the step). Taking the first preconditioning step as an example, the step b-3) in Algorithm 3.2 now becomes:

Algorithm 4.1. Mixed precision modification for preconditioning

---

convert to single precision $\boldsymbol{p} \rightarrow \boldsymbol{p}_{32}$

Preconditioning, solve $\boldsymbol{p}_{t32}$ **in:**

$\boldsymbol{L}\boldsymbol{p}_{t32} = \boldsymbol{p}_{32}$,

then solve $\widehat{\boldsymbol{p}}_{32}$ **in:**

$\boldsymbol{L}\widehat{\boldsymbol{p}}_{32} = \boldsymbol{p}_{t32}$,

convert to double precision $\widehat{\boldsymbol{p}}_{32} \rightarrow \widehat{\boldsymbol{p}}$,

---

where the subscript of "32" denotes the vectors with reduced precision (FP32). The second preconditioning step needs to be modified in a similar manner. To better demonstrate the performance of mixed precision approach, we first evaluate the average time consumption of the various basic operations in a single iteration of BiCG. We adopt the same forward modelling test from Cascadia as in the previous session, using the same GPU platform and parameters. The tolerances for forward calculations are again set as 10^{-10} for the test.

Fig. 4 shows the average time comparison of the different operations in a single iteration of BiCG at the period of 18724.6 s (XY polarization). As can be expected, the time for basic SpMV and "other" operations (mostly parallel reductions) remains mostly unchanged for the double and mixed precision method. However, the time consumption of the four SpSV operations with mixed precision solver is reduced to roughly half of the double precision counterpart. As the SpSVs can take about 50% of the total computation time in double precision cases, the introduction of

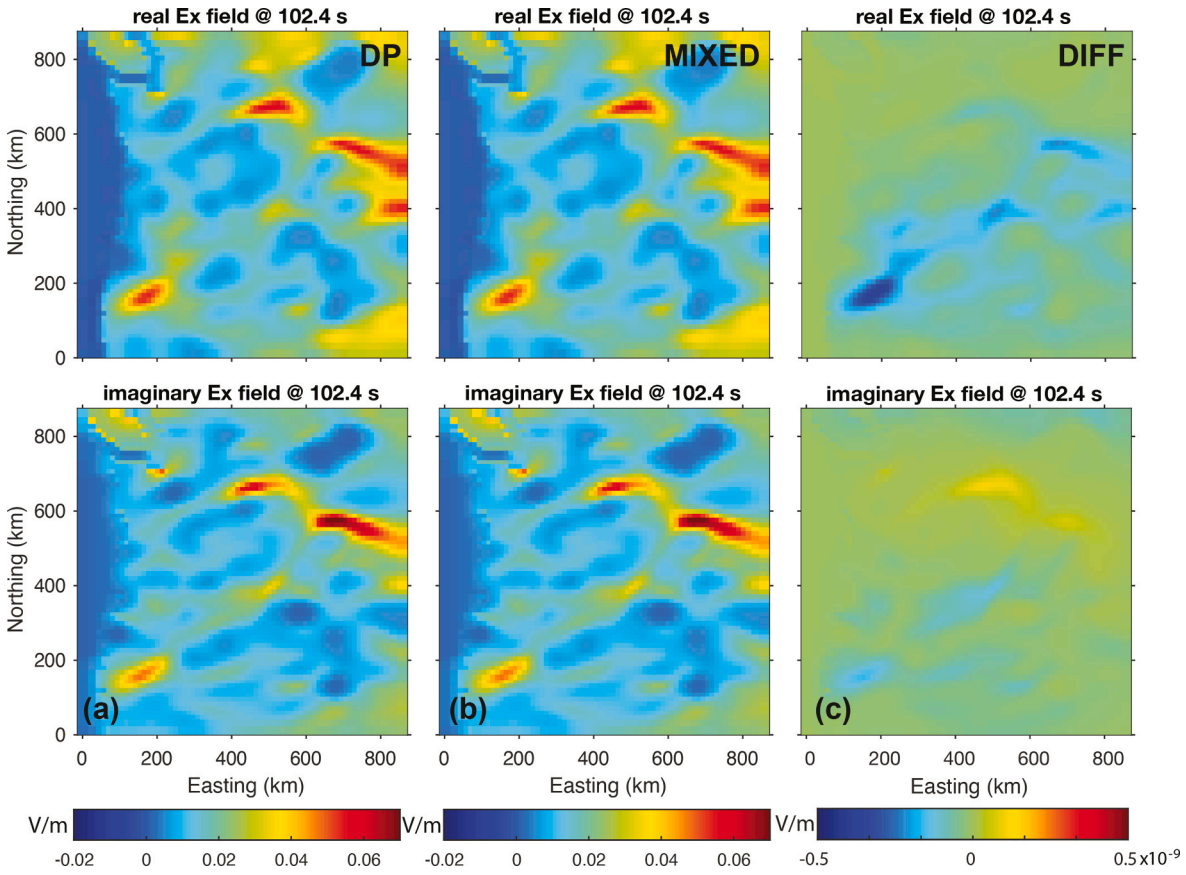


Fig. 6. A comparison on the surface electrical solutions (Ex) in the central part of the Cascadia model with XY polarization at a period of 102.4 s from (a) the double precision approach and (b) the mixed precision approach. Panel (c) shows the solution difference of a–b. DP: double precision; MIXED: mixed precision; DIFF: differences.

Table 1
Scaling test for the Cascadia forward problem with the mixed precision CPU-GPU algorithm. CPU: central processing unit; GPU: graphical processing unit; DoFs: degrees of freedom.

Test Case	Mesh Size	DoFs	Average GPU Solution Time (s)	Average CPU Overhead Time (s)	GPU Memory Consumption (MB)
Cascad 1	80x78x44	797498	0.906	1.406	966
Cascad 2	99x97x54	1515562	2.115	2.960	1405
Cascad 4	125x122x65	2911452	5.586	7.019	2363
Cascad 8	158x154x87	6248099	16.335	20.831	4409
Cascad 16	200x195x94	11750749	50.683	63.903	7746

mixed precision preconditioning can lead to an overall speed-up to 1.30–1.45×, which is quite favorable, considering the relatively simple implementation of the method.

To fully assess the effect of the mixed precision approach on efficiency, we must also consider possible effects of reduced precision on convergence rate of the KSP solver. As shown in Fig. 5a, the mixed precision solver generally does require a slightly larger number of iterations (<10%) to converge, compared with the fully double precision GPU solver. However, the mixed precision approach still maintains an overall 20–30% speed-up advantage over the double precision GPU solver. We conclude that the mixed precision BiCG implementation on GPU is even more efficient than the double precision approach described in the previous session, achieving roughly 30× speed-ups in EM forward computations when compared with conventional CPU-only algorithms (Fig. 5b).

On the other hand, there are concerns that the precision truncation in the mixed precision approach may affect the accuracy of the EM solution computed (Lindquist et al., 2022). We therefore also collate the field solutions from our mixed precision and double precision GPU solvers (Fig. 6). Similar to the results in the previous section, we show that the deviations from the full precision results are also negligible (<0.5 nV per meter).

It is nonetheless worth noting that, with the high efficiency and accuracy of the mixed precision GPU algorithm, the time consumption of the solver phase can be close to, or even less than that of the CPU overheads. The speed-up of the entire forward calculation for the hybrid CPU-GPU approach is only about 15× if the overheads are included. Apparently, the effect of the CPU overhead on the overall efficiency may depend on the performance difference between the CPU and GPU in a certain machine setup. However, it is not yet clear whether the time consumption of the overhead would remain a constant level or grow with the scale of the problem.

As such, we perform a series of forward tests to evaluate the effect of the overhead and the scaling performance of our mixed precision approach. To generate systems with different degree of freedoms (DoFs), we take the Cascadia forward model described above and refine the mesh, while keeping the geometry of the original resistivity structure. As shown in Table 1, in addition to the original forward model (Cascad 1), four forward setups are tested with increased mesh size and DoFs (Cascad 2–16). With the mesh size increasing from 80 × 78 × 44 to 200 × 195 × 94, the DoF of the system equations also grows by almost a factor of 15 (0.8 million to 11.8 million). The performance of our new hybrid algorithm scales well even for a relatively large system size (200 × 195 × 94), taking only 50.68 s for the GPU solution. It is also notable that the GPU memory consumption for the largest mesh size is less than 8 GB, which is easily manageable, even for consumer-grade graphic

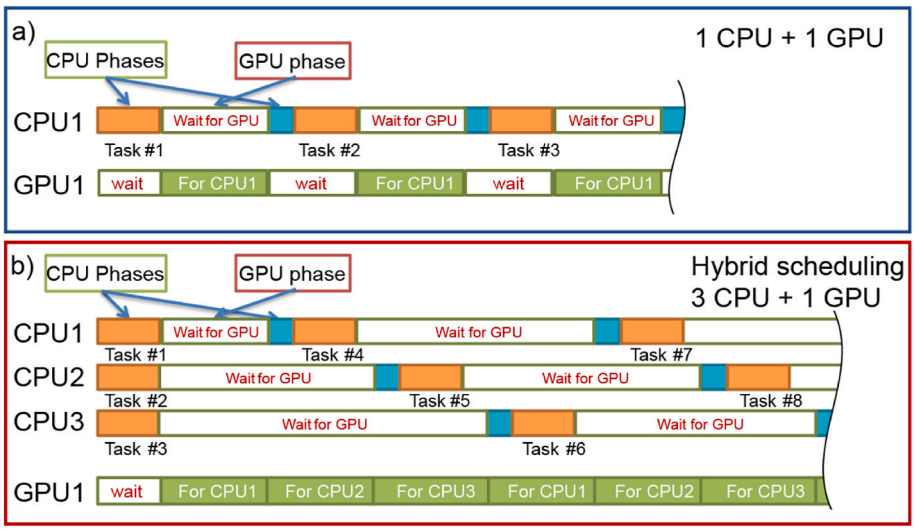


Fig. 7. Schematic diagram for scheduling methods to suppress the effect of overheads for the hybrid CPU-GPU computation algorithm, with (a) 1 CPU core + 1 GPU and (b) 3 CPU cores + 1 GPU. Note the length of the colored time slots displayed here is purely demonstrational. CPU: central processing unit; GPU: graphical processing unit.

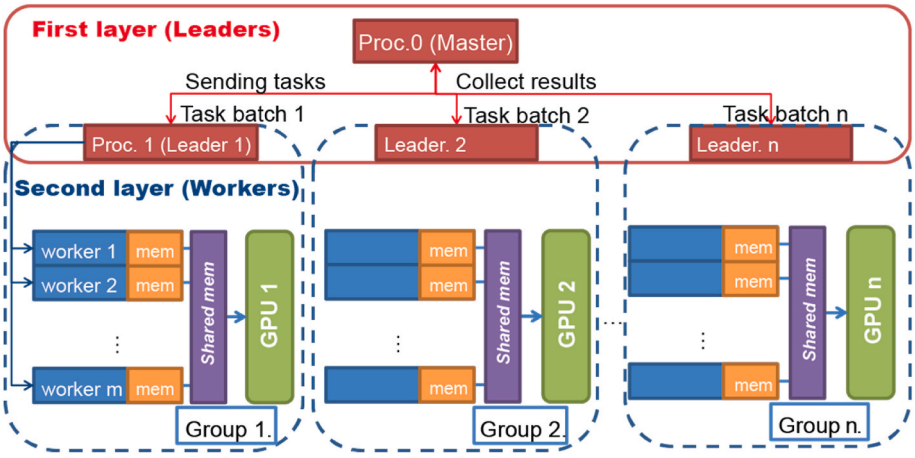


Fig. 8. Two-layered parallelization framework for the hybrid CPU-GPU computation for electromagnetic inversions. CPU: central processing unit; GPU: graphical processing unit.

cards. On the other hand, the average time for CPU overhead remains a similar ratio (~1.3 - 1.5×), over that of the GPU solution for the Cascadia test cases. Indeed, the effect of the overhead on the hybrid algorithm persists as the system scale grows. This needs to be carefully dealt with, as discussed in the next section.

5. Hybrid CPU-GPU parallel inversion framework

As shown by the numerical examples in the previous section, our new hybrid CPU-GPU approach can substantially reduce the solution time compared with conventional CPU solvers. However, according to Amdahl's law (Gustafson, 1988), the real-world speed-up of a parallel program will be limited by the portion of sequential computational tasks. In a similar sense, with the efficiency of the GPU solver increasing, the overall speed-up of our new algorithm will ultimately be determined by the time consumption of the CPU overheads (Fig. 2). The overall speed-up cannot exceed the threshold of:

$$\left(T_{sol\_cpu} + T_{overheads}\right) / T_{overheads}, \tag{5}$$

where the T_{sol_cpu} is the solver time in the CPU approach and the $T_{overheads}$ is the time used in the CPU overhead operations described in Section 3. In other words, even if the linear system can be solved by the GPU solver in an instant (zero time), it is impossible to further increase the speed-up, unless $T_{overheads}$ can be reduced. Although improvement on the efficiency of the overhead phases on the CPU is beyond the scope of the current study, some general discussion is appropriate.

First, let us focus on the scheduling scheme of the CPU and GPU phases in the hybrid approach. As introduced in Section 3.1, the five steps of the forward/adjoint calculations are strictly sequential. Specifically, the next step or operation cannot proceed until the previous one is finished. In the case of a computation platform that consists of one GPU and one CPU (or CPU core), the GPU phase has to wait until the CPU finishes steps 1–3, and then the CPU waits while the system of equations is solved by the GPU in step 4 (Fig. 7a). In this scheme, the independent forward/adjoint tasks (from different periods or transmitters) are carried out in sequence. Note that in the previous section we have shown that the actual GPU solve time may be comparable, or even less than the CPU phase (overheads). This may lead to an awkward situation, with the GPU device idly waiting for the CPU for a large fraction of the total time in the computation (Fig. 7a).

Apparently, the efficiency of the hybrid algorithm can be improved if the "waiting" phase of GPUs in Fig. 7 can be minimized. To this end, we

develop a non-preemptive scheduling scheme to leverage the resource of multiple CPUs to perform the overhead calculations for a single GPU. Fig. 7b illustrates an example with 3 CPUs and 1 GPU – in this case, 3 CPUs (or CPU cores) work in parallel to calculate the overheads for 3 independent forward tasks. The assembled matrices and RHS vectors for different tasks are then fed to the GPU in turn, which keeps the GPU from waiting for CPU overheads. As a typical EM inversion commonly involves tens to hundreds of independent tasks (i.e., forward and adjoint calculations), this scheduling scheme can efficiently mitigate the effect of the overheads by conducting those tasks in parallel. Because modern GPU platforms normally have only a few independent GPU cards but many times more CPU cores, it is also plausible to deploy the scheme by dividing the computational resources into several "groups", with each group consisting of several CPU cores and a single GPU.

To implement the aforementioned parallel scheduling scheme, we use a novel two-layered parallel structure to manage the heterogenous computational resource (Fig. 8). We first divide the computer resources into a few "worker groups," with one GPU and multiple CPU threads in a certain group. For the first layer of parallelization, the communications are realized with the MPI to send task batches to each group "Leader" and collect computed results for the Master.

For the second layer, we exploit the MPI 2.0 feature of MPI groups to let multiple CPUs (or CPU cores) calculate the overheads for individual tasks before feeding the assembled matrices/vectors to the GPU in the group. Those processes (or "workers") can share a memory pool, allowing some common data structures (e.g., grid/mesh data, basic grad or curl operators) to be efficiently stored and shared among the processors in a certain group, which reduces the overall memory consumption. The leader then collects and packs the task result upon completion of the calculation and initializes the collective communication for the harvest of results.

In this case, the conventional single-layer ModEM parallel structure can be best re-used without making fundamental changes to the tested utility routines for the distribution of model/data, the initialization/termination of tasks, or the management of job queues. The modular methods involved with the inversion operations, like the gradient evaluation and line search procedures, can also be maintained as in Egbert and Kelbert (2012), for ease of implementation. In the following section we will demonstrate the real-world performance of our new hybrid CPU-GPU inversion framework using MT test cases.

6. Real-world EM inversion experiments

In this part we perform the tests on the hybrid CPU-GPU parallel inversion framework using the 3D MT data from the central part of the Tibetan Plateau (Dong et al., 2020). The inversion dataset consists of 231 MT stations. MT and geomagnetic transfer functions at 26 periods from 0.1 to 10,000 s were inverted in the experiments. The inversion model mesh comprises 90 × 85 × 80 discrete cells (including padding and air layers), with ~1.8 million DoFs in the formulated system of equations. In order to directly compare the performance of the CPU/GPU solvers, the inversion algorithm for all the tests is set to Nonlinear Conjugate Gradient (NLCG) as described in Kelbert et al. (2014). The relative residual tolerance of the BiCG iterations is set to 10^{-8} for both the forward and the adjoint systems.

For the GPU inversion tests, we use the same workstation platform as for the forward test in the previous sections. For the comparison of conventional algorithm in a CPU-only setup, we use a four-node cluster "FST", with dual 14-core Intel® Xeon E5-2680v4 CPUs with eight channels of DDR4 2400 memory (153.6 GB/s bandwidth) in each computation node. We use only 13 cores from each node to avoid memory bandwidth saturation, which results in 52 cores and 614.4 GB/s memory bandwidth when all four computational nodes are used. The CPU-only codes are compiled with gfortran from the gcc 9.1 package, as with the GPU counterpart. We use the openMPI 4.1.4 library for the parallel communication implementation for both the CPU and hybrid versions.

We start the experiments by setting up the baseline performance of the inversion using 13 physical CPU cores from one computation node. We initialize the inversion from a starting model of a 100-Ωm homogeneous half space and use a termination condition of 100 iterations to avoid prolonged calculations. As each NLCG (Nonlinear Conjugate Gradient) inversion requires two forward and one adjoint calculation, this translates into 200 forward and 100 adjoint calculations in total, which take 4788.82 min (~80 h) to finish (Fig. 9). The wall-time of the inversion reduces to 2790.21 and 1650.3 min when using 26 and 52 CPU cores, from two and four computation nodes in parallel, respectively, showing a good strong-scaling effect. Of note, the ModEM parallel structure needs an extra "master" process to do basic input/output (I/O) and message passing tasks, which means 14, 27 and 53 physical cores are actually used in the above setups.

For the hybrid CPU-GPU framework, we first test the basic performance without the non-preemptive scheduling scheme. In other words, only one CPU core is used to prepare the system matrices and vectors for one single GPU, as illustrated in Fig. 7a. The test inversion takes 3102.79 min (~52 h) to reach 100 iterations (Fig. 9). This already achieves an 1.54× speed-up over the CPU-only setup with 13 physical cores. However, we need to consider the influence of the CPU overheads

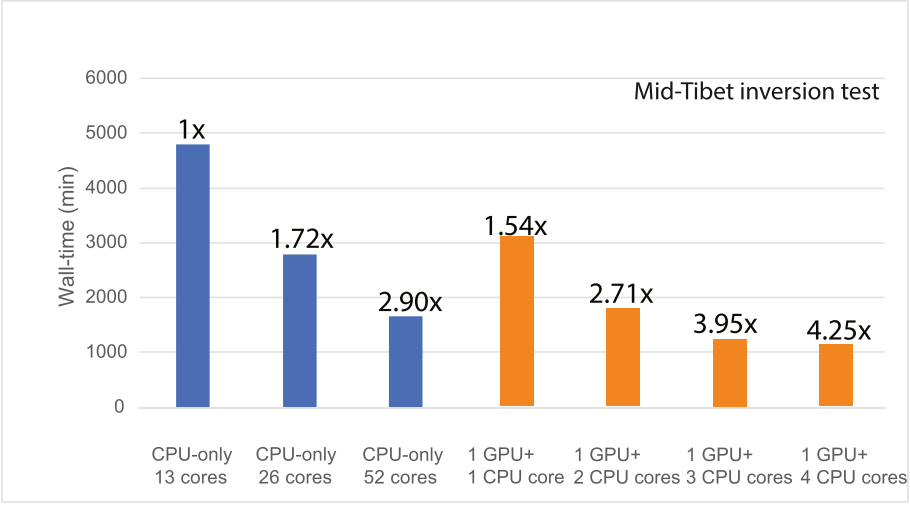


Fig. 9. Comparison of computation time with various parallel CPU and GPU set-ups. Note the result from the one-node CPU-only setup (13 cores) is set as the baseline (1×). CPU: central processing unit; GPU: graphical processing unit.

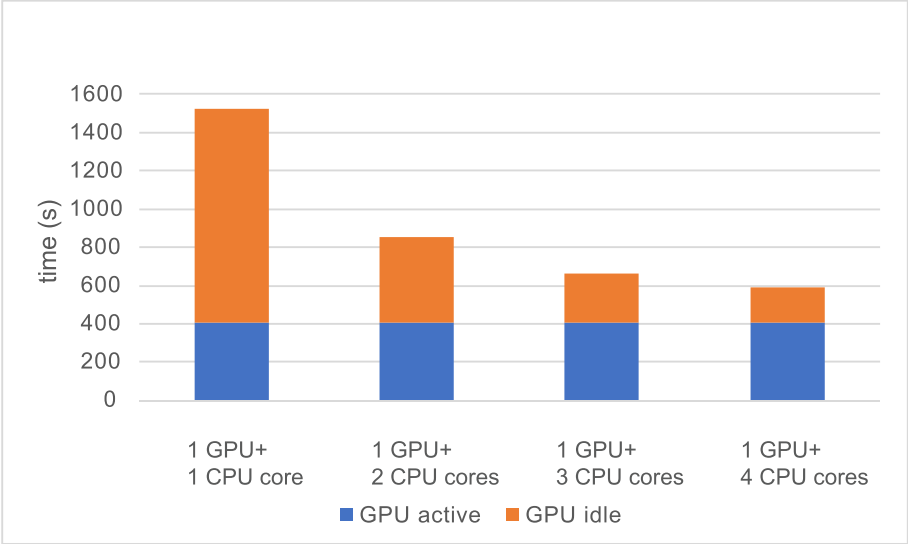


Fig. 10. Comparison of GPU activity during a NLCG iteration with and without the CPU-GPU scheduling scheme. NLCG: non-linear conjugate gradient; CPU: central processing unit; GPU: graphical processing unit.

on the final performance. As pointed out in the previous section, larger CPU overhead is directly related to longer "wait" or idle phase of the GPU, which in turn leads to lower computational efficiency.

We therefore compare the overall activity of the GPU in one single NLCG iteration to demonstrate the real-world effect of the CPU overheads (Fig. 10). However, in the ~1520 s wall-time of one NLCG iteration, the GPU is only active (when solving the system) for 407 s, which means the GPU is idly waiting for the CPU overheads for about 3/4 of the total computation time! This also explains why the GPU inversion test without the scheduling algorithm results in a far from the optimal speed-up. We further test our CPU-GPU scheduling scheme to utilize two, three and four CPU cores to feed the independent computational tasks to the GPU in parallel. It is observed that the total GPU wait or "idle" time is greatly reduced from about 1112 s (1 GPU + 1 CPU core) to only ~185 s (1 GPU + 4 CPU cores), while the GPU active working time remains at a similar level (about 405s) for different scheduling setups (Fig. 10).

Indeed, because the total amount of forward/adjoint calculation is the same for all the inversion tests, the active GPU time should be similar as all the computations are eventually done sequentially on the same GPU. We show that our scheduling scheme (Fig. 7) successfully mitigates the overhead tasks onto multiple CPU cores, which reduces the GPU wait time by a factor of four. As a result, the test inversion for the hybrid CPU-GPU scheme with the "1 GPU +4 CPU cores" setup only takes 1126.52 min to reach 100 iterations, gaining a 4.25× speed-up over the 13-core CPU-only setup (Fig. 9). Our new approach utilizing a single GPU can even outperforms the four-node, 52-core CPU-only setup with a speed-up of 1.45×, which demonstrates that our new hybrid method has the potential to greatly improve the efficiency of real-world EM inversions.

7. Discussion and conclusions

In this study we present a mixed-precision GPU accelerated method for solving the divergence-free Curl-Curl equations, based on the previously proposed algorithm on CPUs. The new method is implemented with a hybrid CPU-GPU EM inversion framework in the ModEM package and tested in a series of real-world forward and inversion experiments. The efficacy of the new hybrid approach is demonstrated by accurately solving the divergence-free Curl-Curl equations of the EM forward modelling problem. A ~30 times speed-up is achieved in the solve phase of the linear systems from the forward problem when compared with conventional CPU-only implementations. We also develop a parallel scheduling algorithm to effectively mitigate the effect of CPU overheads in the hybrid GPU computation framework. Furthermore, parallel inversion tests with MT data indicates that efficiency of the new hybrid CPU-GPU method on a workstation attached with a single consumer-grade GPU card can already outperform a cluster of four inter-connected CPU-only nodes.

Despite the fact that our hybrid approach is built on the divergence-free regularization formulation of the Curl-Curl equations, other approaches, like the conventional divergence correction method, or methods with augmented unknown parameters, could also benefit from the GPU accelerations (e.g., Mackie et al., 1994; Schwarzbach, 2009; Weiss, 2013). However, this may introduce complexities related to the implementation of those approaches. For example, the divergence correction method involves a Poisson-type equation to calculate the correction of the fields after a certain number of Curl-Curl iterations. The Poisson solver will have to be nested within the GPU Curl-Curl outer loops to avoid extra communications between the *Host* and the *Device*.

On the other hand, although our hybrid approach can utilize multiple GPUs to conduct a batch of forward/adjoint tasks in parallel, we did not have the chance to test the performance of multiple GPUs in this study due to the lack of available devices. The introduction of multi-GPU computation could help further in extending the strong scalability of the algorithm. We also have not considered the implementation that involves the use of multiple GPUs on a single, individual computational task, which can be useful when the modelling problem is so large that the memory requirements exceed that of a single GPU. Furthermore, we have not considered more heterogeneous cases and computation platforms (i.e., some computation nodes have GPUs while the others do not). As the number of elements for the system matrix does not exceed 13 in each line, the use of formats like ELL or packet (Bell and Garland, 2009) may be beneficial as memory is easier to coalesce across the GPU threads for these formats. These issues can be explored in future experiments.

It is also worth mentioning that all the experiments we performed in the study are done with a consumer-grade GPU (NVIDIA® RTX 3080Ti). Professional GPUs, which have much greater double-precision capabilities and memory throughputs, would be advantageous for large scale EM geophysical inversions. On the other hand, new computational architectures, like the new many-core processors or fused CPU-GPU systems can also provide the high memory bandwidth (Jackson et al., 2020; Mittal and Vetter, 2015), which is potentially beneficial for highly efficient EM inversions. Further experiments could be performed to evaluate the efficiency of those platforms.

Code availability section

Modular Electromagnetic Inversion Software (ModEM)

Contact: Gary Egbert, gary.egbert@oregonstate.edu; Tel:541-737-2947

Hardware requirements: Memory: depends on the problem; CPU: at least two physical cores; GPU: CUDA compute capability 6.1 or above

Program Language: Fortran, C, makefile

Software required: MPI, LAPACK

Program size: 1.8 MB (binary executive)

The ModEM software and source code are freely available to non-commercial and academic usage. The new hybrid CPU-GPU ModEM version can be accessed anonymously at publicly available GitHub repository:

https://github.com/dong-hao/ModEM-GPU

An overview for the basic routines and modules of ModEM can be found in Kelbert et al. (2014).

CRediT authorship contribution statement

Hao Dong: Conception of the research, Development of the source code (Mixed-precision algorithm, GPU kernels), Conduction of the experiments, Writing of the manuscript; Kai Sun: Development of the source code (GPU kernels), Conduction of the experiments; Gary Egbert: Development of the source code (Model and Data Operators), Writing of the manuscript; Anna Kelbert: Development of the source code (Inversion algorithms), Critical feedbacks; Naser Meqbel: Development of the source code (Parallel infrastructure)

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The datasets used for the forward and inversion tests are freely available to the public; The forward test data (Cascadia) used in this study are included as a supplemental example in the ModEM package. The inverse test data (middle Tibet) can be downloaded from: https://data.mendeley.com/datasets/svwdw67tfv/2

Acknowledgements

The authors thank Dr. Barry Smith for constructive discussions. The authors also appreciate the helpful and very detailed comments from Benjamin Murphy, Brian Shiro, Janet Carter and an anonymous reviewer. Partial support for H.D. was provided by NSFC grants 4227040447 and 4212100033. Partal support for G.E. was provided by NSF grant EAR-1948874. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

References

Carson, E., Higham, N.J., 2018. Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Sci. Comput. 40 (2), A817–A847. https://doi.org/10.1137/17M1140819.

Bell, N., Garland, M., 2009. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09. ACM Press, New York, New York, USA, p. 1. https://doi.org/10.1145/1654059.1654078.

Boland, D., Constantinides, G.A., 2011. Optimizing memory bandwidth use and performance for matrix-vector multiplication in iterative methods. ACM Trans. Reconfigurable Technol. Syst. (TRETS) 4 (3), 22. https://doi.org/10.1145/2000832.2000834.

Carson, E., Gergelits, T., Yamazaki, I., 2022. Mixed precision s-step Lanczos and conjugate gradient algorithms. Numer. Lin. Algebra Appl. 29 (3), e2425 https://doi.org/10.1002/nla.2425.

Chan, T.F., de Pillis, L., van der Vorst, H., 1998. Transpose-free formulations of Lanczos-type methods for nonsymmetric linear systems. Numeric. Algorithms 17, 51–66. https://doi.org/10.1023/A:1011637511962.

Dong, H., Egbert, G.D., 2019. Divergence-free solutions to electromagnetic forward and adjoint problems: a regularization approach. Geophys. J. Int. 216 (2), 906–918. https://doi.org/10.1093/gji/ggy462.

Dong, H., Wei, W., Jin, S., Ye, G., Jones, A.G., Zhang, L., et al., 2020. Shaping the surface deformation of central and south Tibetan Plateau: insights from magnetotelluric array data. J. Geophys. Res. Solid Earth 125 (9). https://doi.org/10.1029/2019JB019206 e2019JB019206.

Egbert, G.D., Kelbert, A., 2012. Computational recipes for electromagnetic inverse problems. Geophys. J. Int. 189 (1), 251–267. https://doi.org/10.1111/j.1365-246X.2011.05347.x.

Foltinek, D., Eaton, D., Mahovsky, J., Moghaddam, P., McGarry, R., 2009. Industrial-scale reverse time migration on GPU hardware. In: SEG Technical Program Expanded Abstracts 2009. Society of Exploration Geophysicists, pp. 2789–2793. https://doi.org/10.1190/1.3255428.

Freund, R.W., 1993. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. SIAM J. Sci. Comput. 14 (2), 470–482. https://doi.org/10.1137/0914029.

Gropp, W., Lusk, E., 1994. Implementing MPI: The 1994 MPI implementors' workshop. In: Proceedings Scalable Parallel Libraries Conference. IEEE Comput. Soc. Press, pp. 55–59. https://doi.org/10.1109/SPLC.1994.377005.

Gustafson, J.L., 1988. Reevaluating Amdahl's law. Commun. ACM 31 (5), 532–533. https://doi.org/10.1145/42411.42415.

Higham, N.J., Mary, T., 2022. Mixed precision algorithms in numerical linear algebra. Acta Numer. 31, 347–414. https://doi.org/10.1017/S0962492922000022.

Jackson, A., Weiland, M., Brown, N., Turner, A., Parsons, M., 2020. Investigating applications on the A64FX. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER). IEEE Comput. Soc. Press, pp. 549–558. https://doi.org/10.1109/CLUSTER49012.2020.00078.

Jiang, B., Wu, J., Povinelli, L., 1996. The origin of spurious solutions in computational electromagnetics. J. Comput. Phys. 123 (125), 104–123. https://doi.org/10.1006/jcph.1996.0082.

Kelbert, A., Meqbel, N.M., Egbert, G.D., Tandon, K., 2014. ModEM: a modular system for inversion of electromagnetic geophysical data. Comput. Geosci. 66, 40–53. https://doi.org/10.1016/j.cageo.2014.01.010.

Kordy, M., Wannamaker, P., Maris, V., Cherkaev, E., Hill, G.J., 2016. 3-dimensional magnetotelluric inversion including topography using deformed hexahedral edge finite elements and direct solvers parallelized on symmetric multiprocessor computers – Part II: Direct data-space inverse solution. Geophys. J. Int. 204 (1), 94–110. https://doi.org/10.1093/gji/ggv411.

Lindquist, N., Luszczek, P., Dongarra, J., 2022. Accelerating restarted GMRES with mixed precision arithmetic. IEEE Trans. Parallel Distr. Syst. 33 (4), 1027–1037. https://doi.org/10.1109/TPDS.2021.3090757.

Liu, Shangbin, Chen, C., Sun, H., 2022. Fast 3D transient electromagnetic forward modeling using BEDS-FDTD and GPU parallelization. Geophysics 87 (5), E359–E375. https://doi.org/10.1190/geo2021-0596.1.

Liu, Sheng, Jin, S., Chen, Q., 2022. Three-dimensional gravity inversion based on optimization processing from edge detection. Geodesy Geodynam. 13 (5), 503–524. https://doi.org/10.1016/j.geog.2022.03.005.

Mackie, R.L., Watts, M.D., 2012. Detectability of 3-D sulphide targets with AFMAG. In: SEG Technical Program Expanded Abstracts 2012. Society of Exploration Geophysicists, pp. 1–4. https://doi.org/10.1190/segam2012-1248.1.

Mackie, R.L., Smith, J.T., Madden, T.R., 1994. Three-dimensional electromagnetic modeling using finite difference equations: the magnetotelluric example. Radio Sci. 29 (4), 923–935. https://doi.org/10.1029/94RS00326.

Mackie, R.L., Rodi, W., Watts, M.D., 2001. 3-D magnetotelluric inversion for resource exploration. In: *SEG Technical Program Expanded Abstracts 2001*. Society of Exploration Geophysicists, pp. 1501–1504. https://doi.org/10.1190/1.1816392.

McCourt, M., Smith, B., Zhang, H., 2015. Sparse matrix-matrix products executed through coloring. SIAM J. Matrix Anal. Appl. 36 (1), 90–109. https://doi.org/10.1137/13093426X.

Mittal, S., Vetter, J.S., 2015. A survey of CPU-GPU heterogeneous computing techniques. ACM Comput. Surv. 47 (4), 69. https://doi.org/10.1145/2788396.

Mudigere, D., Hao, Y., Huang, J., Jia, Z., Tulloch, A., Sridharan, S., et al., 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In: Proceedings of the 49th Annual International Symposium on Computer Architecture. ACM Press, New York, NY, USA, pp. 993–1011. https://doi.org/10.1145/3470496.3533727.

Newman, G.A., 2014. A review of high-performance computational strategies for modeling and imaging of electromagnetic induction data. Surv. Geophys. 35 (1) https://doi.org/10.1007/s10712-013-9260-0, 85–10.

Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T. J., 2007. A survey of general-purpose computation on graphics hardware. Comput. Graph. Forum 26 (1), 80–113. https://doi.org/10.1111/j.1467-8659.2007.01012.x.

Pandey, M., Fernandez, M., Gentile, F., Isayev, O., Tropsha, A., Stern, A.C., Cherkasov, A., 2022. The transformational role of GPU computing and deep learning in drug discovery. Nat. Mach. Intell. 4 (3), 211–221. https://doi.org/10.1038/s42256-022-00463-x.

Patro, P.K., Egbert, G.D., 2008. Regional conductivity structure of Cascadia: preliminary results from 3D inversion of USArray transportable array magnetotelluric data. Geophys. Res. Lett. 35 (20), L20311 https://doi.org/10.1029/2008GL035326.

Rogers, B.M., Krishna, A., Bell, G.B., Vu, K., Jiang, X., Solihin, Y., 2009. Scaling the bandwidth wall. Comput. Architect. News 37 (3), 371–382. https://doi.org/10.1145/1555815.1555801.

Rothberg, E., Gupta, A., 1990. A Comparative Evaluation of Nodal and Supernodal Parallel Sparse Matrix Factorization: Detailed Simulation Results, pp. 4–7. Stanford, California.

Sadiku, M.N.O., 2000. In: Raton, Boka (Ed.), Numerical Techniques in Electromagnetics, Second Edi). CRC Press, London, pp. 136–142.

Schwarzbach, C., 2009. *Stability Of Finite Element Discretization of Maxwell's Equations for Geophysical Applications* (PhD). Technische Universität Bergakademie Freiberg.

Siripunvaraporn, W., Egbert, G.D., 2009. WSINV3DMT: vertical magnetic field transfer function inversion and parallel implementation. Phys. Earth Planet. In. 173 (3–4), 317–329. https://doi.org/10.1016/j.pepi.2009.01.013.

Smith, J.T., 1996. Conservative modeling of 3-D electromagnetic fields, part II: biconjugate gradient solution and an accelerator. Geophysics 61 (5), 1319–1324. https://doi.org/10.1190/1.1444055.

Varılsüha, D., 2020. 3D inversion of magnetotelluric data by using a hybrid forward-modeling approach and mesh decoupling. Geophysics 85 (5), E191–E205. https://doi.org/10.1190/geo2019-0202.1.

van der Vorst, H.A., 1993. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 13 (2), 631–644. Retrieved from. http://epubs.siam.org/doi/abs/10.1137/0913035.

Wait, J.R., 1982. Geo-Electromagnetism. Elsevier, New York, pp. 184–208. https://doi.org/10.1016/B978-0-12-730880-7.X5001-7.

Wang, M., Yang, T., Flechas, M.A., Harris, P., Hawks, B., Holzman, B., et al., 2021. GPU-accelerated machine learning inference as a service for computing in neutrino experiments. Front. Big Data 3, 604083. https://doi.org/10.3389/fdata.2020.604083.

Weaver, J.T., Agarwal, A.K., Pu, X.H., 1999. 3-D finite-difference modeling of the magnetic field in geoelectromagnetic induction. In: Oristaglio, M., Spies, B. (Eds.), Three-Dimensional Electromagnetics, first ed. Society of Exploration Geophysicists, pp. 426–443. https://doi.org/10.1190/1.9781560802154.ch27.

Weiland, T., 1985. On the unique numerical solution of Maxwellian eigenvalue problems in three dimensions. Part. Accel. 17, 227–242.

Weiss, C.J., 2013. Project APhiD: a Lorenz-gauged A- decomposition for parallelized computation of ultra-broadband electromagnetic induction in a fully heterogeneous Earth. Comput. Geosci. 58, 40–52. https://doi.org/10.1016/j.cageo.2013.05.002.

Xiao, Y., Liu, Y., 2015. GPU acceleration for the Gaussian elimination in magnetotelluric Occam inversion algorithm. In: Wong, W. (Ed.), Proceedings of the 4th International Conference on Computer Engineering and Networks. Springer, Cham, pp. 123–131. https://doi.org/10.1007/978-3-319-11104-9_15.

Yang, P., Gao, J., Wang, B., 2015. A graphics processing unit implementation of time-domain full-waveform inversion. Geophysics 80 (3), F31–F39. https://doi.org/10.1190/geo2014-0283.1.

Yang, D., Merchant, D., Haber, E., 2017. 3D electromagnetic inversion of logging while drilling data. In: SEG Technical Program Expanded Abstracts 2017. Society of Exploration Geophysicists, pp. 890–894. https://doi.org/10.1190/segam2017-17795351.1.

Zhdanov, M.S., Keller, G.V., 1994. The Geoelectrical Methods in Geophysical Exploration. Elsevier, Amsterdam; New York, pp. 248–252.