Development of 3D viscoelastic crustal deformation analysis solver with data-driven method on GPU

Sota Murakami[1], Kohei Fujita[1,2], Tsuyoshi Ichimura[1,2], Takane Hori[3], Muneo Hori[3], Maddegedara Lalith[1], and Naonori Ueda[4]

[1] Earthquake Research Institute and Department of Civil Engineering, The University of Tokyo, Tokyo, Japan
`{souta,fujita,ichimura,lalith}@eri.u-tokyo.ac.jp`
[2] Center for Computational Science, RIKEN, Kobe, Japan
[3] Japan Agency for Marine-Earth Science and Technology, Yokohama, Japan
`{horit,horimune}@jamstec.go.jp`
[4] Center for Advanced Intelligence Project, RIKEN, Tokyo, Japan
`naonori.ueda@riken.jp`

Abstract. In this paper, we developed a 3D viscoelastic analysis solver with a data-driven method on GPUs for fast computation of highly detailed 3D crustal structure models. Here, the initial solution is obtained with high accuracy using a data-driven predictor based on previous time-step results, which reduces the number of multi-grid solver iterations and thus reduces the computation cost. To realize memory saving and high performance on GPUs, the previous time step results are compressed by multiplying a random matrix, and multiple Green's functions are solved simultaneously to improve the memory-bound matrix-vector product kernel. The developed GPU-based solver attained an 8.6-fold speedup from the state-of-art multi-grid solver when measured on compute nodes of AI Bridging Cloud Infrastructure at National Institute of Advanced Industrial Science and Technology. The fast analysis method enabled calculating 372 viscoelastic Green's functions for a large-scale 3D crustal model of the Nankai Trough region with 4.2×10^9 degrees of freedom within 333 s per time step using 160 A100 GPUs, and such results were used to estimate coseismic slip distribution.

Keywords: unstructured finite-element method · data-driven predictor · OpenACC · viscoelastic analysis

1 Introduction

Improvement in the estimation of interplate conditions such as plate sticking and sliding is expected to play an important role in the advancement of source scenarios for large earthquakes. In particular, the estimation of interplate conditions considering viscoelastic deformation is useful for estimating an afterslip and predicting continuous crustal deformation after a large earthquake. In recent years, data required for the advancement of interplate state estimation have been accumulated due to the improvement of the seafloor crustal deformation observation directly above the seismogenic zone (e.g., [16]) and the acquisition of crustal structure data with approximately 1 km resolution by advancement in underground structure exploration. On the other hand, the theoretical solution assuming the crustal structure as a multilayered semi-infinite medium [8] is often used in obtaining the displacement responses at observation points to unit slips (Green's function), which are used in the inverse analysis of interplate conditions. Although the calculation of Green's functions based on a highly detailed three-dimensional (3D) crustal structure model and its use for estimating the interplate state is expected to improve the accuracy of interplate state estimation, this calculation leads to the huge analysis cost comprising 100-1000 cases of large-scale viscoelastic analysis.

Most of the computational cost in viscoelastic crustal deformation analysis is spent on solving the large-scale simultaneous equations obtained by discretizing the crustal structure model. Since a method scalable on a parallel computing environment is essential for conducting large-scale calculations, and since low-frequency components dominate in viscoelastic response, a multi-grid-based solver is considered effective. In fact, multi-grid based conjugate gradient solvers, which use geometric and algebraic multi-grid methods, have been developed and applied to crustal deformation analysis [6,10]. In addition, viscoelastic analysis using these multi-grid solvers has been accelerated using GPUs, enabling forward analysis of viscoelastic response on highly detailed 3D models. On the other hand, further reduction of computation cost is required to realize viscoelastic Green's function calculation, which corresponds to a computation cost of about 100-1000 cases of forward analysis.

In recent years, data-driven methods have been utilized to improve the performance of equation-based methods (e.g., [11]), and their effectiveness in viscoelastic crustal deformation analysis has also been demonstrated [7]. The initial solution to a large simultaneous equation is obtained with high accuracy using a data-driven predictor based on past time-step results, which reduces the number of multi-grid solver iterations and thus reduces the computation cost. Both the data-driven predictor and the multi-grid solver are designed to be scalable, and have been shown to perform well on CPU-based massively parallel computer Fugaku [4], and are expected to be effective on GPU-based systems. In this study, a multi-grid solver with a data-driven predictor for GPU computation environment is developed for fast computation of Green's functions of viscoelastic crustal deformation. Since the data-driven predictor, which learns and predicts solutions based on a large amount of data, hinders the performance of GPUs with relatively small memory capacity, methods enabling a reduction in memory footprint are combined with the data-driven predictor. While the multi-grid solver is also effective on GPUs due to its high scalability, its performance is limited by random access in the sparse matrix-vector computations; we introduce simultaneous computation of multiple Green's functions for a reduction in random access and a further performance improvement. Considering the development cost, we develop the solver using directive-based OpenACC [3].

As an application example, we calculated 372 viscoelastic Green's functions at 333 s per time step for a large-scale 3D crustal model of the Nankai Trough with 4.2×10^9 degrees of freedom using 160 A100 GPUs, and performed inverse estimation of coseismic slip distribution.

The following is the structure of this paper. In Section 2, the target viscoelastic crustal deformation analysis is described. Section 3 describes the target multi-grid solver with a data-driven predictor algorithm. Section 4 describes the development of the multi-grid solver with the data-driven predictor on GPUs. Section 5 describes the performance of the solver, and Section 6 describes an application example of the proposed analysis method to the Nankai-Trough earthquake. Section 7 summarizes this study.

2 Target problem

In this study, we model the Earth's crust as a linear viscoelastic body based on the Maxwell model and solve the equations

$$\sigma_{ij,j} + f_i = 0, \tag{1}$$

with

$$\dot{\sigma}_{ij} = \lambda \dot{\epsilon}_{kk}\delta_{ij} + 2\mu\dot{\epsilon}_{ij} - \frac{\mu}{\eta}(\sigma_{ij} - \frac{1}{3}\sigma_{kk}\delta_{ij}), \tag{2}$$

$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}). \tag{3}$$

Here, σ and f are the stress tensor and external force, while $\dot{()}$, δ, η, ϵ, and u are the first derivative in time, Kronecker delta, viscosity coefficient, strain tensor, and displacement, respectively. λ and μ are Lame's coefficients. In this study, the governing equations are discretized by the finite-element method, which analytically satisfies the traction-free boundary conditions. Herein, second-order tetrahedral elements are used for the accurate calculation of stress and strain for crust deformation problems with complex geometry and heterogeneous material properties. The time evolution of viscoelastic crustal deformation analysis is computed based on [10] (Algorithm 1). Here, the fault slip is evaluated based on the split-node technique [15]. In general, it is difficult to generate high-quality large-scale 3D finite-element models for complex crustal structure models. In this paper, we construct a 3D finite-element model with unstructured second-order tetrahedral elements by using an automated robust mesh generation method[10]. In Algorithm 1, almost all of the computation time is spent on solving simultaneous equations:

$$\mathbf{K}^v \delta\mathbf{u} = \mathbf{f}, \tag{4}$$

where the degrees of freedom (DOF) of the unknown vector $\delta\mathbf{u}$ becomes large (e.g., the DOF becomes 4.2×10^9 for the application problem shown in this study). Thus, the goal becomes solving Eq. (4) with large DOF in a short time on multi-GPU environments.

Algorithm 1 Algorithm for solving linear viscoelastic response of crust. Here, superscript $()^i$ is the variable in the i-th time step. dt is the time increment. $\mathbf{B}$ is the displacement-strain transformation matrix. $\mathbf{A}$ and $\mathbf{D}$ are matrices indicating material property. $\mathbf{K} = \Sigma_e \int_{\Omega_e} \mathbf{B}^T\mathbf{D}\mathbf{B}d\Omega$. $\mathbf{K}^v = \Sigma_e \int_{\Omega_e} \mathbf{B}^T\mathbf{D}^v\mathbf{B}d\Omega$. $\beta^n = \mathbf{D}^{-1}\mathbf{A}\sigma^n$. $\mathbf{D}^v = (\mathbf{D}^{-1} + \alpha\ dt\ \beta')^{-1}$, where β' is the Jacobian matrix of β and α is the controlling parameter. Ω is the viscoelastic body.

```
Compute $f$ by split − node technique
Solve $\mathbf{K}\mathbf{u}^1 = \mathbf{f}^1$
$\sigma^1 \Leftarrow \mathbf{D}\mathbf{B}\mathbf{u}^1$
$\delta\mathbf{u}^1 \Leftarrow 0$
$i \Leftarrow 2$
while $i \leq N_t$ do
    $f^i \Leftarrow \Sigma_e \int_{\Omega_e} \mathbf{B}^T(dt\mathbf{D}^v\beta^{i-1} - \sigma^{i-1})d\Omega + \mathbf{f}^1)$
    Solve $\mathbf{K}^v\delta\mathbf{u}^i = \mathbf{f}^i$ with initial solution $\delta\mathbf{u}^i_{init}$
    $\mathbf{u}^i \Leftarrow \mathbf{u}^{i-1} + \delta\mathbf{u}^i$
    $\sigma^i \Leftarrow \sigma^{i-1} + \mathbf{D}(\mathbf{B}\delta\mathbf{u}^i - dt\beta^i)$
    $i \Leftarrow i + 1$
end while
```

3 Base multi-grid solver with data-driven predictor

In this section, we outline the multi-grid solver with the data-driven predictor[7] proposed as a fast solver for Eq. (4), which will be used as a base of the GPU solver developed in this study. A solver algorithm with high single-node peak performance with low computational cost, together with good load-balancing and low communication cost, is required for fast computation of large-scale finite-element models in a massively parallel computing environment. In this solver, a scalable data-driven initial solution predictor is added to a multi-grid solver that fulfills such requirements, leading to a reduction in the number of iterations in the multi-grid solver and thus a reduction in the computation time. Below, we outline the data-driven predictor and the multi-grid based iterative solver.

3.1 Data-driven predictor

By using the results of past time steps to accurately predict the initial solution δu^i_{init}, the number of iterations and thus the computation time of the multi-grid solver for solving Eq. (4) can be reduced. The idea of Dynamic Mode Decomposition (DMD)[14] is applied to construct an initial solution predictor suitable for massively parallel computers. Here, computed results up to the $i-1$-th time step are learned to predict the initial solution at the i-th time step. In DMD, an operator that represents time evolution is estimated from time series data, and this operator is used to predict the solution of the next step based on the solution at the current step. Instead of predicting the solution of the entire target domain at once, the target domain is divided into small domains, and the solutions in each domain are predicted within each domain. This enables efficient prediction of the modes including the local time and space components in each domain by only a small number of modes. However, even if a small region is targeted for prediction, it includes trend due to non-stationary time evolution, which is difficult to predict. Therefore, the second-order Adams-Bashforth method is used to predict the trend as

$$\delta \mathbf{u}^i_{adam} \Leftarrow \mathbf{u}^{i-3} - 3\mathbf{u}^{i-1} + 2\mathbf{u}^{i-1}. \tag{5}$$

We apply DMD to $\mathbf{x}^i = \delta\mathbf{u}^i - \delta\mathbf{u}^i_{adam}$ excluding the trend component. This allows $\delta\mathbf{u}^i$ to be predicted with sufficient accuracy from a small number of modes. Specifically, we define a matrix as $\mathbf{X}^{i-1} = [\mathbf{x}^{i-1}, \cdots, \mathbf{x}^{i-s}]$ using the data of previous $s+1$ steps, the time evolution operator $\mathbf{C}$ which satisfies $\mathbf{X}^{i-1} = \mathbf{C}\mathbf{X}^{i-2}$ is estimated from this matrix by the modified Gram-Schmidt method, and the initial solution for the next step is estimated using operator $\mathbf{C}$ as

$$\delta\mathbf{u}^i_{init} \Leftarrow \delta\mathbf{u}^i_{adam} + \mathbf{C}(\delta\mathbf{u}^{i-1} - \delta\mathbf{u}^i_{adam}). \tag{6}$$

The domain in each MPI process is divided into small non-overlapping domains using METIS[2], and the displacement increments for the nodes in each domain are estimated from the time-series data of nodes in the same domain. The algorithm does not require communication between domains, making it scalable in a massively parallel computing environment.

3.2 Multi-grid solver with data-driven predictor

The prediction results from the data-driven predictor are used for the initial solution of an adaptive conjugate gradient solver with a three-level multi-grid preconditioner. Algorithm 2 shows an overview of the method. In the preconditioner of the adaptive conjugate gradient method, multi-grid models generated by stepwise coarsening of the target finite-element model with second-order tetrahedral elements are used to solve the target model approximately. First, a coarse grid consisting of first-order tetrahedral elements is obtained by removing the edges nodes in second-order tetrahedral elements based on the geometric multigrid method, and then a further coarsened model is obtained by the algebraic multi-grid method. Although various types of algebraic coarsening are proposed, uniform coarsening is used for maintaining load balance. Using these coarsened models, an approximate solution is obtained for preconditioning of the conjugate gradient method. Hereafter, we refer to the iteration of the original conjugate gradient loop as the outer loop and refer to the iteration of solving the preconditioning equations with another conjugate gradient solver as the inner loop. First, an approximate solution is obtained using the coarsest model (Algorithm 2 line 9; inner loop 2), and using the obtained solution as the initial solution, the approximate solution is updated using the tetrahedral linear element model (Algorithm 2, line 11; inner loop 1). Finally, the solution to the original mesh is obtained (Algorithm 2, line 13; inner loop 0). Inner loops reduce the cost per iteration compared to the original model by reducing the number of unknowns and the nonzero component of the sparse matrix $\mathbf{K}$. In addition, the coarsened

Algorithm 2 The iterative solver to obtain solution $\delta\mathbf{u}$. The input variables are $\mathbf{K}, \bar{\mathbf{K}}_i, \bar{\mathbf{P}}_i, \delta\mathbf{u}, \mathbf{f}, \epsilon, \bar{\epsilon}_i^{in}, N_i$ and N. Here, $\bar{\mathbf{K}}_i$ and $\bar{\mathbf{P}}_i$ represent global stiffness matrices and the mapping matrices between grids. $\bar{N}_i$ and $\bar{\epsilon}_i$ are the threshold values. The other variables are temporal. $(\bar{\ })$ represents FP32 variables, while the other variables are in FP64. All computation steps in this solver, except MPI synchronization and scalar coefficient computation, are performed in GPUs.

(a) Outer loop

```
 1: predict $\delta\mathbf{u}$ by the data-driven predictor
 2: $\mathbf{r} \Leftarrow \mathbf{K}\delta\mathbf{u}$
 3: $\mathbf{r} \Leftarrow \mathbf{f} - \mathbf{r}$
 4: $\beta \Leftarrow 0$
 5: $i \Leftarrow 1$
 6: while $||\mathbf{r}||^2/||\mathbf{f}||^2 > \epsilon$ do
 7:     $\bar{\mathbf{u}}_0 \Leftarrow \bar{\mathbf{M}}^{-1}\mathbf{r}$
 8:     $\bar{\mathbf{r}}_2 \Leftarrow \bar{\mathbf{P}}_2^T\bar{\mathbf{P}}_1^T\mathbf{r}$
 9:     $\bar{\mathbf{u}}_2 \Leftarrow \bar{\mathbf{P}}_2^T\bar{\mathbf{P}}_1^T\bar{\mathbf{u}}_0$
10:     solve $\bar{\mathbf{u}}_2 = \bar{\mathbf{K}}_2^{-1}\bar{\mathbf{r}}_2$ using (b) with $\bar{\epsilon}_2^{in}$ and
        $N_2$ (* inner loop 2 *)
11:     $\bar{\mathbf{u}}_1 \Leftarrow \bar{\mathbf{P}}_2\bar{\mathbf{u}}_2$
12:     solve $\bar{\mathbf{u}}_1 = \bar{\mathbf{K}}_1^{-1}\bar{\mathbf{r}}_1$ using (b) with $\bar{\epsilon}_1^{in}$ and
        $N_1$ (* inner loop 1 *)
13:     $\bar{\mathbf{u}} \Leftarrow \bar{\mathbf{P}}_1\bar{\mathbf{u}}_1$
14:     solve $\bar{\mathbf{u}} = \bar{\mathbf{K}}_0^{-1}\bar{\mathbf{r}}_0$ using (b) with $\bar{\epsilon}_0^{in}$ and $N_0$
        (* inner loop 0 *)
15:     $\mathbf{z} \Leftarrow \bar{\mathbf{u}}_0$
16:     if $i > 1$ then
17:         $\gamma \Leftarrow (\mathbf{z}, \mathbf{q})$
18:         $\beta \Leftarrow \gamma/\rho$
19:     end if
20:     $\mathbf{p} \Leftarrow \mathbf{z} + \beta\mathbf{p}$
21:     $\mathbf{q} \Leftarrow \mathbf{K}\mathbf{p}_e$
22:     $\rho \Leftarrow (\mathbf{z}, \mathbf{r})$
23:     $\gamma \Leftarrow (\mathbf{p}, \mathbf{q})$
24:     $\alpha \Leftarrow \rho/\gamma$
25:     $\mathbf{r} \Leftarrow \mathbf{r} - \alpha\mathbf{q}$
26:     $\delta\mathbf{u} \Leftarrow \delta\mathbf{u} + \alpha\mathbf{p}$
27:     $i \Leftarrow i + 1$
28: end while
```

(b) Inner loop

```
 1: $\bar{\mathbf{e}} \Leftarrow \bar{\mathbf{K}}\bar{\mathbf{u}}$
 2: $\bar{\mathbf{e}} \Leftarrow \bar{\mathbf{r}} - \bar{\mathbf{e}}$
 3: $\bar{\beta} \Leftarrow 0$
 4: $i \Leftarrow 1$
 5: while $||\bar{\mathbf{e}}_1||^2/||\bar{\mathbf{r}}||^2 > \bar{\epsilon}$ and
    $N > i$ do
 6:     $\bar{\mathbf{z}} \Leftarrow \bar{\mathbf{M}}^{-1}\bar{\mathbf{e}}$
 7:     if $i \geq 1$ then
 8:         $\bar{\beta} \Leftarrow \bar{\rho}_a/\bar{\rho}_b$
 9:     end if
10:     $\bar{\mathbf{p}} \Leftarrow \bar{\mathbf{z}} + \bar{\beta}\bar{\mathbf{p}}$
11:     $\bar{\mathbf{q}} \Leftarrow \bar{\mathbf{K}}\bar{\mathbf{p}}$
12:     $\bar{\gamma} \Leftarrow (\bar{\mathbf{p}}, \bar{\mathbf{q}})$
13:     $\alpha \Leftarrow \bar{\rho}_a/\bar{\gamma}$
14:     $\bar{\rho}_b \Leftarrow \bar{\rho}_a$faccou
15:     $\bar{\mathbf{e}} \Leftarrow \bar{\mathbf{e}} - \bar{\alpha}\bar{\mathbf{q}}$
16:     $\bar{\mathbf{u}} \Leftarrow \bar{\mathbf{u}} + \bar{\alpha}\bar{\mathbf{p}}$
17:     $i \Leftarrow i + 1$
18: end while
```

model allows long-range errors to be solved with fewer iterations. In each inner loop solver, a 3×3 block-Jacobi preconditioned conjugate gradient solver (Algorithm 2,b) with good load-balance and robustness is used. While FP64 is used in the outer loop to guarantee the computational accuracy of the final solution, FP32 is used in inner loops, where only approximate solutions are required. This halves the memory footprint, data transfer size, and communication size in the inner loops, which account for most of the computation time, and is expected to reduce time-to-solution.

4 GPU-based multi-grid solver with data-driven predictor

The multi-grid solver with data-driven predictor, which is designed to be efficient and scalable on massively parallel environments, is also expected to perform well on GPU-based environments. However, GPUs have relatively low memory capacity and memory bandwidth in comparison with its floating point performance, when compared to A64FX CPU-based Fugaku; thus, the data-driven predictor that requires large amounts of memory and matrix-vector products that require large amounts of memory accesses become bottlenecks in GPU performance. Therefore, we developed a multi-grid solver with the data-driven predictor for GPUs based on the previous CPU-based solver while improving the algorithm by reducing the amount of memory usage, memory accesses, and random data accesses.

Considering program development cost and portability, we use OpenACC to port CPU code to the GPU. OpenACC, which enables computation on the GPU by inserting compiler directives into CPU programs, allows porting pre-developed CPU applications to the GPU environment incrementally with relatively little effort. Although native programming models such as CUDA enable detailed tuning of the code to maximize performance on GPUs, it has been shown that by designing algorithms suitable for GPUs, the computation time of an OpenACC implementation is comparable to that of a CUDA implementation (for example, see [20] as an example of crustal deformation analysis using a multi-grid solver).

4.1 Data-driven predictor enhanced by memory footprint reduction method

In the method of [7], given a data set $\mathbf{X}, \mathbf{Y}$of sizes $m \times s$ (the number of degrees of freedom in the domain $\times$ time steps), where $\mathbf{X}$ is the input and $\mathbf{Y}$ is the corresponding output, the response $\mathbf{y}$ to another input $\mathbf{x}$ is computed as

$$\mathbf{y} = \mathbf{Y}\mathbf{U}\mathbf{P}^T\mathbf{x}. \tag{7}$$

Here, $\mathbf{P} = \mathbf{X}\mathbf{U}$, where $\mathbf{P}$ is a matrix with orthogonal columns and $\mathbf{U}$ is an upper triangular matrix. This orthogonalization $\mathbf{P} = \mathbf{X}\mathbf{U}$ is computed by the modified

Gram-Schmidt method, but it is not suitable for GPUs with small memory capacity because it requires keeping matrices $\mathbf{X}, \mathbf{Y}$ and another temporary matrix on memory during orthogonalization. In addition, since many times of the inner product is required sequentially for vectors as long as the number of degrees of freedom in the corresponding domain, a large memory access cost is involved. Therefore, in this paper, a random matrix $\mathbf{Q}$ of size $n \times m$ $(m \ll n)$, is used to transform the input data set $\mathbf{X}$ into $\mathbf{X}' \Leftarrow \mathbf{QX}$ and the input value $\mathbf{x}$ into $\mathbf{x}' \Leftarrow \mathbf{Qx}$ (e.g., a $25,745 \times 16$ matrix $\mathbf{X}$ is replaced with a 96×16 matrix $\mathbf{X}'$ in the performance measurement problem), which reduces the computational cost and memory usage for modified Gram-Schmidt orthogonalization. Although predictions based on the transformed data set are an approximation of the original algorithm's predictions, it is known that by taking m sufficiently larger than the number of time steps s used for the prediction, the singular values of $\mathbf{QX}$ and $\mathbf{X}$ coincide with high probability[9]. Therefore, it is possible to estimate $\mathbf{y}$ with almost no reduction in accuracy. In this study, $\mathbf{y}$ is computed as $\mathbf{a} = \mathbf{UP}^T\mathbf{x}'$ at first, and then as $\mathbf{y} = \mathbf{Ya}$. While additional computation for transforming $\mathbf{x}' \Leftarrow \mathbf{Qx}$ is required, its cost is negligible compared to the Gram-Schmidt method on the original problem, and the memory requirement of storing random matrix $\mathbf{Q}$ is also negligible as a common random matrix $\mathbf{Q}$ can be reused for all the small domains in which the data-driven predictor is applied.

4.2 Multi-grid solver enhanced by multi-vector computation

In the multi-grid solver, the sparse matrix-vector product (SpMV) kernel is the most computationally expensive kernel of each inner loop (Algorithm 2b). In general, the Generalized SpMV (GSpMV) kernel, which computes sparse-matrix dense-matrix products, achieves higher throughput than the SpMV kernels as it corresponds to computing multiple SpMVs by reading the target matrix once, which reduces the amount of memory access. This also leads to a reduction in random memory accesses by allocating the same components of multiple vectors consecutively in the memory address space. This leads to high throughput on GPUs that can access continuous data efficiently. Since the sparse matrix (e.g., $\bar{\mathbf{K}}$ in Algorithm 2b line 11) is constant at any source input in viscoelastic analysis, we calculate four sets of Green's functions simultaneously, thereby replacing the SpMV with the GSpMV. The maximum values for the relative errors in the 4 residual vectors are used for judging the convergence of each loop.

For the outer loop and inner loop 0, the Element-by-Element (EBE) method [17] is used to compute the GSpMV. In the parallel computation of matrix-vector products based on the EBE method, it is necessary to avoid data inconsistency when adding the local matrix-vector product results for each element to the global vector. While coloring of elements can be used to avoid data recurrence on multi-core CPUs, recent NVIDIA GPUs equip hardware-accelerated atomics and have high throughput atomic operations capability. Utilizing this atomic add functionality makes more efficient data access possible compared to the coloring algorithm. In inner loop 1 and inner loop 2, sparse matrices are stored in memory by Block Compressed Row Storage (BCRS) with block size 3 to compute GSpMV.

5 Performance Measurement

5.1 Performance Measurement Settings

Since the performance of the data-driven predictor is highly dependent on the problem characteristics, we evaluate solver performance on the example application problem in Section 6. The finite-element model comprises 1.0×10^9 tetrahedral elements with 4.2×10^9 DOF. Setting the time increment as $dt = 86400$ s, we measure the performance of crustal deformation between time step number $21 \leq N_t \leq 30$, where the data-driven predictor can be applicable, as the actual calculation of Green's functions is computed for several to 100 years (100 to 5000 time steps). We solve all problems with relative error tolerance $\epsilon = 10^{-8}$. The tolerances and maximum iterations in the inner loops are set to $(\epsilon_0, \epsilon_1, \epsilon_2) = (0.5, 0.25, 0.15)$ and $(N_0, N_1, N_2) = (30, 80, 300)$, respectively. In the data-driven predictor, the entire domain is divided into 163,840 subdomains, and data of the previous $s = 16$ time steps are used for estimation. The transformation is calculated using a random matrix with $m = 96$.

To demonstrate the effectiveness of the developed method, we compare performance with a 3×3 block-Jacobi preconditioned solver (PCGE) and a multi-grid based adaptive conjugate gradient solver (multi-grid solver), both using second-order Adams-Bashforth method for predicting the initial solution [5]. Here, PCGE corresponds to skipping lines 7-13 in Algorithm 2a, and the multi-grid solver corresponds to switching the data-driven predictor in the proposed solver with the Adams-Bashforth method. We also compare the performance of the proposed solver with the multi-grid solver with data-driven predictor on CPU.

Performance was measured on GPU-based supercomputer AI Bridging Cloud Infrastructure(ABCI)[1], which is operated by the National Institute of Advanced Industrial Science and Technology. Each compute node (A) of ABCI has eight NVIDIA Tesla A100 GPUs and two Intel Xeon Platinum 8360Y CPUs (36 cores), and is interconnected with a full bisection bandwidth network (see Table 1). The FP64 peak performance of the GPU is 14.0× (memory bandwidth is 30.4×) of that of the CPU. 16 nodes (128 GPUs) with 1 MPI process per GPU (128 total MPI processes) were used for GPU measurements, and the same number of nodes and processes were used with 9 OpenMP threads per MPI process for CPU measurements.

5.2 GPU kernel performance

We measure the performance of the computation kernels which account for most of the execution time of the entire application (Table 2).

[5] Although sophisticated GPU-based methods specialized for viscoelastic crustal deformation analysis and specific GPU architecture is proposed [19], we compare with generally available solvers stated above for readability.

Table 1. Configuration of ABCI Compute Node (A)

		Hardware peak per node
CPU	Intel Xeon Platinum 8360Y	5.529 TFLOPS
	(54 MB Cache, 2.4 GHz, 36 Cores, 72 Threads)×2	
memory	512 GiB DDR4 3200MHz RDIMM	408 GB/s
GPU	NVIDIA A100 for NVLink	77.6 TFLOPS
	40 GiB HBM2 ×8	12.4 TB/s
Interconnect	InfiniBand HDR (200 Gbps) ×4	100 GB/s

As the Gram-Schmidt kernel is memory bandwidth bound, direct porting to GPU led to $4020/248 = 16.2$-fold speedup from the CPU. attained by directly porting it to (78.9% of memory bandwidth, 1.47% of FP64 peak performance). Furthermore, the reduction in computation by the random matrix transformation led to a further reduction in the time of the Gram-Schmidt kernel. This is due to the reduction of GPU device memory data transfer size from 302 GB to 2.36 GB by use of the proposed method replacing a $25,745 \times 16$ matrix $\mathbf{X}$ with a 96×16 matrix $\mathbf{X}'$. Although this method required computing the random matrix-vector product $\mathbf{Qx}$, it can be performed in 5.38 ms; leading to the overall speedup of the data-driven predictor by 18.9-fold from the direct porting case. Furthermore, the memory size required for the data-driven predictor was 16.3 GB per GPU for the developed method, which is significantly smaller than the 62.9 GB required for the direct porting method.

Next, we measure the performance of SpMV and GSpMV kernels. While the FP32 peak performance of EBE-based SpMV of inner loop 0 was improved from 10.5% on the CPU to 16.3% on the GPU, due to the large number of registers on GPUs, the use of GSpMV led to 44.3% of FP32 peak on GPU, leading to further improvement in computational performance. While the BCRS-based SpMV in inner loops 1 and 2 are memory-bandwidth bound kernels, conversion of the kernel to GSpMV kernels with 4 vectors reduced the amount of memory access per vector, (1.19 GB to 0.253 GB and 420 MB to 105 MB for inner loop 1 and 2, respectively) resulting in 2.46- and 2.89-fold speedup, respectively, compared to the GPU-based SpMV implementations.

As is seen, the introduction of suitable algorithms for GPUs led to high efficiency on each kernel.

5.3 Solver Performance

We see the effectiveness of the data-driven predictor for a reduction in elapsed time. By use of the data-driven predictor, the initial error ϵ of the second-order Adams-Bashforth method (2.11×10^{-3}) was improved to 2.46×10^{-5}, indicating that prediction can be performed with high accuracy. As a result, the total number of iterations of the multi-grid solver was reduced from 5237 iterations to 1098 iterations. In particular, the number of iterations in inner loop 2 was

Table 2. Performance of each kernel. Elapsed time is normalized per vector.

	Elapsed time (FLOPS efficiency, Memory Throughput Efficiency)		
computation component	CPU	GPU (direct porting)	GPU (proposed)
BCRS $\bar{K}_2\bar{u}_2$*	13.2 ms (1.85%, 57.2%)	0.396 ms (2.30%, 65.9%)	0.137 ms (6.60%, 60.4%)
BCRS $\bar{K}_1\bar{u}_1$*	35.4 ms (1.96%, 57.5%)	0.782 ms (2.52%, 76.2%)	0.318 ms (6.19%, 70.8%)
2nd order EBE $\bar{K}_0\bar{u}_0$*	145 ms (10.5%, 27.6%)	4.90 ms (16.3%, 64.3%)	1.81 ms (44.3%, 69.0%)
2nd order EBE Ku**	184 ms (8.42%, 39.0%)	8.66 ms (18.5%, 53.2%)	4.63 ms (34.6%, 51.3%)
Gram-Schmidt**	4020 ms (1.04%, 28.5%)	248 ms (1.47%, 78.9%)	2.3 ms (10.9%, 73.2%)
random compression**	-	-	5.38 ms (16.3%, 82.9%)

*ratio to FP32 peak. **ratio to FP64 peak.

significantly reduced from 4473 to 936, suggesting that the data-driven predictor has high prediction performance for the low-frequency components. In addition, introducing GSpMV significantly reduces the computation time for the cost dominant matrix-vector products, resulting in 2.01-, 2.12-, and 2.90-fold speedup per iteration for inner loops 0, 1, and 2, respectively. As a result, the developed solver attained an 8.6-fold speedup from a widely used state-of-the-art multi-grid solver (Fig. 1). The multi-grid solver performs well due to its ability to efficiently solve low-frequency errors with the use of fast inner loops (the multi-grid solver's inner loops were 1.59, 9.15, and 15.8-fold faster than the PCGE iterations for the inner loop 0, 1, and 2, respectively), leading to a 191-fold speedup from the standard PCGE solver requiring 10056 iterations and 170 s computation time. Since scalability has been demonstrated for the original CPU-based solver with data-driven predictor, it is expected that the proposed GPU-based solver will also be scalable. The speedup using GPU was 72.5 times when compared with the CPU-based implementation of SCALA22 (64.2 s), which is higher than peak performance and memory bandwidth ratio between CPU and GPU of 14.0- and 30.4-fold, respectively, indicating that the development of algorithms suitable for GPU led to large performance improvements. The introduction of the data-driven predictor enhanced by memory footprint reduction and GSpMV reducing computational cost is expected to be equally effective in CPU implementations.

6 Application example

To demonstrate the effectiveness of the developed solver, we conducted an inversion analysis on a highly detailed crustal structure model to estimate the coseismic slip for the Nankai Trough earthquake. In this study, only elastic/viscoelastic deformation due to coseismic slip is considered, and crustal deformation due to afterslip and fault locking is not considered. Green's function g_i, which aggregates the displacements at each time and observation point for the unit fault x_i, is calculated by viscoelastic crustal deformation analysis. The observation model using these Green's functions is expressed as

$$\mathbf{d} = \mathbf{G}\mathbf{a} + \mathbf{e}, \tag{8}$$

where $\mathbf{d}$ is the observed data (the observed amount of crustal deformation), $\mathbf{G} = [g_1, \cdots, g_n]$, $\mathbf{a}$ is a model parameter (the amount of slip in the unit fault

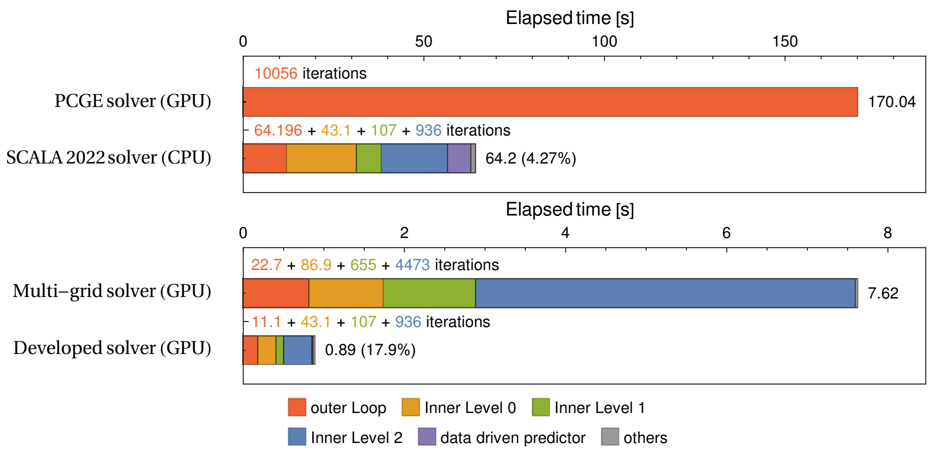

Fig. 1. Elapsed time and required iterations per time step for each solver

x_i), and $\mathbf{e}$ is the error following a normal distribution with mean $\mathbf{0}$ and variance-covariance matrix $\mathbf{\Sigma}$. Here, the model parameter $\mathbf{a}$ is determined by minimizing the objective function,

$$\Phi(\mathbf{a}) = (\mathbf{d} - \mathbf{Ga})^T \Sigma (\mathbf{d} - \mathbf{Ga}) + \lambda \mathbf{a}^T \mathbf{La} + \mu |\mathbf{a}|_1, \tag{9}$$

where $a^T La$ is a term used to constrain the smoothness of the slip distribution [18]. Since the extent of slip cannot be predicted in advance, the basic function of the slip distribution is set wider than the range where slip actually occurs, and the L1 regularization term $|\mathbf{a}|_1$ is used to estimate a sparse slip distribution. The hyperparameters λ and μ are determined by k-fold cross-validation [5].

For the crustal structure data, we use the model based on [12,13]. Based on crustal structure data, the 3D finite-element model of the Japanese island is generated with a target area of 2496 km $\times$ 2496 km $\times$ 1100 km centered at 135°E, 33.5°N. The viscosity of the continental and oceanic mantle is set to 2.0×10^{18} Pa s. Figure 2 shows the finite-element model generated with the smallest element size $ds = 500$ m. As in the performance measurement problem, $dt = 86400$ s and $N_t = 30$ are used. We introduce unit faults set up in grid form in Hori et al (only unit faults that are in the Eastern half of the FE model are used). The number of unit faults is 186, and since we consider the slip distribution responses of two components on the fault plane, we calculate $186 \times 2 = 372$ Green's functions.

We set a hypothetical reference coseismic slip distribution shown in Fig. 3a). The direction of the reference seismic slip is assumed to be uniform in the direction of azimuth 125 degrees. Surface displacement is assumed to be observed by the Global Navigation Satellite System (GNSS), GNSS-Acoustic system, and ocean bottom pressure sensors (Fig. 3). The observation noise is not considered, and the displacement obtained from viscoelastic analysis using the reference coseismic slip as input is used as observation data.

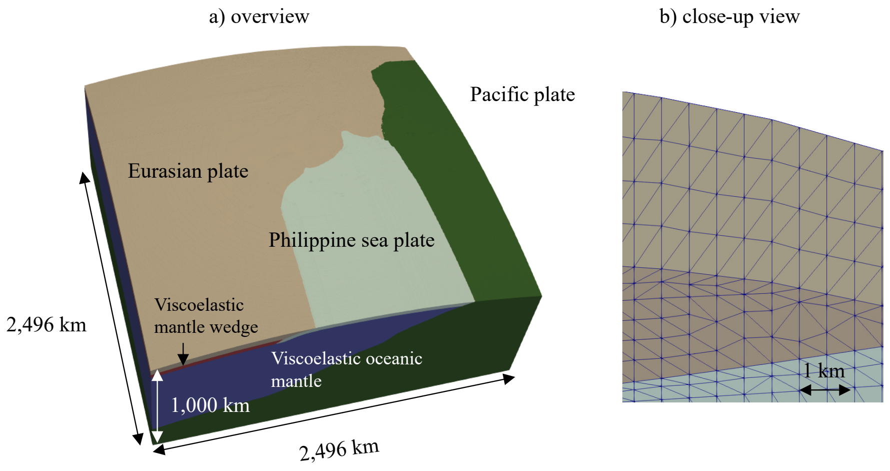

Fig. 2. Generated finite-element model used for the application example. a) Overview and b) close-up view.

In the proposed method, four Green's functions are calculated simultaneously in one set of viscoelastic analyses; thus, 372 Green's functions were calculated in 96 sets of viscoelastic analyses. The overall computation time was 33800 s on 160 GPUs. The computation time for the $21 \leq N_t \leq 30$ steps measured in the performance measurement was 3330 s (8.96 s per step/function), which was almost the same time in the performance measurement. Thus, we can see that the developed method was robustly effective for the many Green's function inputs.

The estimated coseismic slip distribution is shown in Fig. 3b). The estimated moment magnitude is 8.13, which is almost the same as that of the reference slip (8.11), indicating that the magnitude of the earthquake is almost accurately captured.

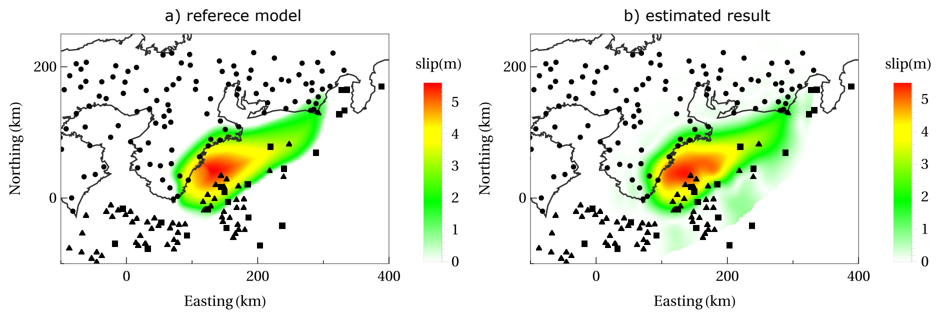

Fig. 3. Coseismic slip distribution in a) reference model and b) estimated results. Black points show observation points.

7 Conclusions

In this study, we developed a multi-grid solver with the data-driven predictor on GPUs for fast computation of the viscoelastic response of a highly detailed 3D crustal structure model for inverse analysis. While the original algorithm resulted in large memory footprint for storing time-history data, suitable algorithms were made to reduce GPU memory usage and elapsed time, and Green's functions were solved simultaneously for improving the performance of memory-bound matrix-vector product kernels. As a result, the developed GPU solver attained an 8.6-fold speedup from a state-of-art multi-grid solver on the ABCI compute environment. As an application example, we calculated 372 viscoelastic Green's functions of a large-scale 3D crustal model with 4.2×10^9 degrees of freedom using 160 A100 GPUs. Calculation of viscoelastic Green's functions using highly detailed 3D crustal structure models enabled by this study is expected to contribute to the improvement of slip estimation considering the 3D crustal structure.

Acknowledgements Computational resource of AI Bridging Cloud Infrastructure (ABCI) provided by National Institute of Advanced Industrial Science and Technology (AIST) was used. This work was supported by MEXT as "Program for Promoting Researches on the Supercomputer Fugaku" (Large-scale numerical simulation of earthquake generation, wave propagation and soil amplification, JPMXP1020200203). This work was supported by JSPS KAKENHI Grant Numbers 18H05239, 22K12057, 22K18823. This work was supported by MEXT, under its Earthquake and Volcano Hazards Observation and Research Program. This work was supported by JST SPRING, Grant Number JPMJSP2108.

References

1. Computing Resources of AI bridging Cloud Infrastructure. `http://abci.ai/en/about_abci/computing_resource.html`
2. Metis 5.1.0. `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`, accessed 2022-02-19
3. OpenACC. `http://www.openacc.org/`, accessed 2022-02-19
4. Supercomputer fugaku, Riken center for computational science. `https://www.r-ccs.riken.jp/en/postk/project`
5. Blum, A., Kalai, A., Langford, J.: Beating the hold-out: Bounds for k-fold and progressive cross-validation. In: Proceedings of the twelfth annual conference on Computational learning theory. pp. 203–208 (1999)
6. Fujita, K., Ichimura, T., Koyama, K., Inoue, H., Hori, M., Maddegedara, L.: Fast and Scalable Low-Order Implicit Unstructured Finite-Element Solver for Earth's Crust Deformation Problem. Proceedings of the Platform for Advanced Scientific Computing Conference pp. 1–10 (2017)
7. Fujita, K., Murakami, S., Ichimura, T., Hori, T., Hori, M., Lalith, M., Ueda, N.: Scalable finite-element viscoelastic crustal deformation analysis accelerated with data-driven method. In: 2022 IEEE/ACM Workshop on Latest Advances in

Scalable Algorithms for Large-Scale Heterogeneous Systems (ScalAH). pp. 18–25 (2022). https://doi.org/10.1109/ScalAH56622.2022.00008

8. Fukahata, Y., Matsu'ura, M.: Quasi-static internal deformation due to a dislocation source in a multilayered elastic/viscoelastic half-space and an equivalence theorem. Geophysical Journal International **166**(1), 418–434 (2006)
9. Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM review **53**(2), 217–288 (2011)
10. Ichimura, T., Agata, R., Hori, T., Hirahara, K., Hashimoto, C., Hori, M., Fukahata, Y.: An elastic/viscoelastic finite element analysis method for crustal deformation using a 3-D island-scale high-fidelity model. Geophys. J. Int. **206**(1), 114–129 (2016)
11. Ichimura, T., Fujita, K., Koyama, K., Kusakabe, R., Kikuchi, Y., Hori, T., Hori, M., Maddegedara, L., Ohi, N., Nishiki, T., et al.: 152k-computer-node parallel scalable implicit solver for dynamic nonlinear earthquake simulation. In: International Conference on High Performance Computing in Asia-Pacific Region. pp. 18–29 (2022)
12. Koketsu, K., Miyake, H., Afnimar, Tanaka, Y.: A proposal for a standard procedure of modeling 3-d velocity structures and its application to the tokyo metropolitan area, japan. Tectonophysics **472**(1), 290–300 (2009). https://doi.org/10.1016/j.tecto.2008.05.037, https://www.sciencedirect.com/science/article/pii/S0040195108002539, deep seismic profiling of the continents and their margins
13. Koketsu, K., Miyake, H., Suzuki, H.: Japan integrated velocity structure model version 1. In: Proceedings of the 15th World Conference on Earthquake Engineering. No. 1773, Lisbon (2012)
14. Kutz, J.N., Brunton, S.L., Brunton, B.W., Proctor, J.L.: Dynamic mode decomposition: data-driven modeling of complex systems. SIAM (2016)
15. Melosh, H.J., Raefsky, A.: A simple and efficient method for introducing faults into finite element computations. Bull. Seismol. Soc. Am. **71**(5), 1391–1400 (1981)
16. Tadokoro, K., Kinugasa, N., Kato, T., Terada, Y.: Experiment of acoustic ranging from gnss buoy for continuous seafloor crustal deformation measurement. In: AGU Fall Meeting Abstracts. vol. 2018, pp. T41F–0361 (2018)
17. Winget, J., Hughes, T.: Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies. Comput. Methods Appl. Mech. Eng. p. 711–815 (1985)
18. Yabuki, T., Matsu'Ura, M.: Geodetic data inversion using a bayesian information criterion for spatial distribution of fault slip. Geophysical Journal International **109**(2), 363–375 (1992)
19. Yamaguchi, T., Fujita, K., Ichimura, T., Glerum, A., van Dinther, Y., Hori, T., Schenk, O., Hori, M., Wijerathne, L.: Viscoelastic crustal deformation computation method with reduced random memory accesses for gpu-based computers. In: Computational Science–ICCS 2018: 18th International Conference, Wuxi, China, June 11-13, 2018, Proceedings, Part II 18. pp. 31–43. Springer (2018)
20. Yamaguchi, T., Fujita, K., Ichimura, T., Naruse, A., Lalith, M., Hori, M.: GPU implementation of a sophisticated implicit low-order finite element solver with FP21-32-64 computation using OpenACC pp. 3–24 (2020)