Geophysical Journal International

Geophys. J. Int. (2017) **210,** 787–800
Advance Access publication 2017 May 10
GJI Seismology

doi: 10.1093/gji/ggx203

Fast crustal deformation computing method for multiple computations accelerated by a graphics processing unit cluster

Takuma Yamaguchi,[1] Tsuyoshi Ichimura,[1] Yuji Yagi,[2] Ryoichiro Agata,[3] Takane Hori[3] and Muneo Hori[1]

[1] *Earthquake Research Institute, The University of Tokyo,* 1-1-1 *Yayoi, Bunkyo-ku, Tokyo* 1130032, *Japan. E-mail: yamaguchi@eri.u-tokyo.ac.jp*
[2] *Graduate School of Life and Environmental Science, University of Tsukuba,* 1-1-1 *Ten-noudai, Tsukuba* 3058572, *Japan*
[3] *Japan Agency for Marine-Earth Science and Technology,* 3173-25 *Showa-machi, Kanazawa-ku, Yokohama* 2360001, *Japan*

Accepted 2017 May 9. Received 2017 May 1; in original form 2016 December 12

SUMMARY

As high-resolution observational data become more common, the demand for numerical simulations of crustal deformation using 3-D high-fidelity modelling is increasing. To increase the efficiency of performing numerical simulations with high computation costs, we developed a fast solver using heterogeneous computing, with graphics processing units (GPUs) and central processing units, and then used the solver in crustal deformation computations. The solver was based on an iterative solver and was devised so that a large proportion of the computation was calculated more quickly using GPUs. To confirm the utility of the proposed solver, we demonstrated a numerical simulation of the coseismic slip distribution estimation, which requires 360 000 crustal deformation computations with 82 196 106 degrees of freedom.

Key words: Numerical solutions; Inverse theory; Numerical approximations and analysis; Computational seismology; Kinematics of crustal and mantle deformation.

1 INTRODUCTION

Numerical simulations of crustal deformation have been studied intensively with the aim of understanding the state of the crustal structures in relation to earthquake generation processes. Most studies have used analytical solutions, with the assumption that the crustal structure is a semi-infinite space (e.g. Okada 1985). However, several recent studies have used the 3-D finite element (FE) method at low resolutions, because simplifying the 3-D heterogeneity of crustal structures may significantly impact the results of some cases, as noted by Masterlark (2003) and Hughes *et al.* (2010).

The availability of high-resolution crustal deformation observational data has increased with the improvement of observation technologies. For example, in the area around Japan, the GNSS Earth Observation Network System (GEONET; Geospatial Information Authority of Japan 2010) continuously observes crustal deformation on land at a spatial resolution of approximately 20 km. In addition, several types of observation systems have been established for seafloor observations, including the Seafloor Observation Network for Earthquakes and tsunamis along the Japan Trench (S-Net); the geodetic seafloor monitoring system of the Japan Coast Guard used to observe coseismic displacements of the 2011 Tohoku earthquake at five benchmarks on the seabed (Sato *et al.* 2011); and the post-seismic deformation observations following the 2011 Tohoku earthquake (Watanabe *et al.* 2014). Integrating these data can capture crustal deformation across Japan, both on land and on the seafloor. Moreover, high-resolution geometric data of crustal structure are available. In Japan, digital elevation data of the 3-D crustal structure are available for the region (e.g. the Crustal Activity Modelling Program (CAMP) model of plate geometry for the Japanese Islands; Hashimoto *et al.* 2012). The velocity structure of northeast and southwest Japan at a 1 km resolution for the most detailed areas are available [Japan seismic hazard information station (J-SHIS), provided by National Research Institute for Earth Science and Disaster Resilience], and the ground-surface geometry at a 250 m resolution in an area of 1000 × 1000 km are now available (JTOPO30, provided by the Marine Information Research Centre). Moreover, observational and high-resolution crustal structure data (hereinafter, referred to as high-fidelity crustal structure data) are now available for many other regions, worldwide.

As a result, the demand is growing for methods that can compute elastic crustal deformation using high-resolution 3-D numerical modelling and that consider the surface geometry and heterogeneity of crustal structures. When targeting Japan, the domain of the analysis is in the order of $10^{3-4} \times 10^{3-4} \times 10^{2-3}$ km. If a numerical model based on high-fidelity crustal structure data with sufficient fine discretization to guarantee convergence of the numerical solution was used, the model would have 10^{8-10} degrees of freedom (DOF). To handle such a massive computation cost within a realistic time frame, there is great interest in developing fast numerical computation methods for large-scale crustal deformation computations. In these computations, most of computation time is spent solving a simultaneous linear equation, which is the result of constitutive rules and discretization. Therefore, creating a faster solver for simultaneous linear equations would benefit fast numerical simulations. Ichimura *et al.* (2016) proposed a

simulation method accelerated by an iterative solver based on variable pre-conditioning and multiple-precision arithmetic using Open Multi-Processing (OpenMP)/Message Passing Interface (MPI) hybrid parallel computation. Using this method, a crustal deformation computation with 10^{10} DOF was calculated in 10^2 s with 8192 K computer nodes. However, this method was designed for calculations by a supercomputer, such as K computer (Miyazaki *et al.* 2012), to calculate elastic and viscoelastic crustal deformations in the Japanese Islands.

Multiple crustal deformation computations enable optimization, data assimilation inverse analysis, sensitivity analysis and Monte Carlo (MC) simulation. Although these applications are also of great interest, in this study, we focused on estimating the effects of uncertainties included in the model as an example application of our proposed method. In crustal deformation computations, it is desirable to consider uncertainties, including material properties, geometries, and inputs. Quantitative approaches for handling uncertainties have become popular. For example, Yagi and Fukahata (2011) performed crustal deformation computations with simplified crustal structures, included error to obtained Green's functions and estimated slip distribution. Meanwhile, Fukuda & Johnson (2008) estimated the posterior distribution of fault parameters using Markov chain Monte Carlo methods. Combining computations using numerical models based on high-fidelity data with approaches using multiple computations is expected to elucidate mechanisms of earthquake generation and crustal deformation. However, the computation cost of such simulations increases depending on the number of repetitive computations required. Both of the aforementioned approaches require 10^{5-6} straightforward analyses. Using a supercomputer is not always realistic, and the method by Ichimura *et al.* (2016) is not easily applied to simulations with more than 10^5 repetitive calculations which have 10^{8-10} DOF. In this study, we only examined elastic crustal deformation. Compared to viscoelastic deformation, displacements change more locally in elastic crustal deformation. Therefore, we assumed that a model with 10^8 DOF would be reasonable for the analysis. Small-scale computation environments, such as computer clusters, can be used in problems of this scale; therefore, there is a need for a fast computation method that can be conducted in these environments.

Graphics processing unit (GPU) clusters, which are computer clusters that contain GPUs, have recently become common in scientific computing. GPU clusters are advantageous due to their lower cost and increased computation performance per power consumed compared with computer clusters without GPUs, and it is thought that GPU clusters are broadly applicable to numerical simulations with parallel computation. However, GPU calculation performance varies greatly depending on the computation method. Therefore, GPU computing must be performed within the limits of the hardware to ensure good performance. Here, we proposed a computation method of elastic crustal deformation using a heterogeneous solver with a GPU cluster. We developed the solver so that a significant proportion of the computation was calculated quickly in the GPUs. Using a smaller computer comprising GPUs, we reduced the computation time and performed many computations within a reasonable timeframe. This method can be used to compute crustal deformation with quantitative uncertainty evaluations.

The remainder of this paper is organized as follows. Section 2.1 describes the fundamental method of FE formulation. Section 2.2 describes the fast solver, with central processing units (CPUs) and GPUs, in detail and Section 2.3 describes each calculation methods appropriate for GPUs. Section 3.1 provides a numerical verification of the proposed method. In Section 3.2, we first show that the proposed method increases the computation speed with a comparison of the computation time and power consumption with previous methods. Next, we show a stochastic estimation of coseismic slip distribution including 360 000 crustal deformation computations with 10^8 DOF based on the 2011 Tohoku earthquake to confirm the advantages of the proposed method.

2 METHODOLOGY

2.1 Summary of the FE formulation

We used the FE method for the simulation, because it is suitable for accurately handling geometric shapes. Coseismic crustal deformation is typically modelled as elastic deformation caused by dislocation of the fault surface. The target equation is rewritten after discretization with the FE method, while taking the boundary conditions into consideration:

$$\mathbf{Ku} = \mathbf{f}, \tag{1}$$

where $\mathbf{K}$, $\mathbf{u}$ and $\mathbf{f}$ are the global stiffness matrix, displacement vector, and force vector, respectively. Infinite elements are generated on the side and the bottom of the domain to describe the infinite condition. The global stiffness matrix is generally sparse because basis functions are only supported locally. We used the split-node technique (Melosh & Raefsky 1981) in our simulation to introduce fault dislocation to the FE model, because it does not require the modification of $\mathbf{K}$. The force vector $\mathbf{f}$ is calculated as follows:

$$\mathbf{f} = -(\mathbf{K}^{\text{upper}}\Delta\mathbf{u}_f - \mathbf{K}^{\text{lower}}\Delta\mathbf{u}_f), \tag{2}$$

where $\Delta\mathbf{u}_f$ is half of the dislocation on the fault surface, and $\mathbf{K}^{\text{upper}}$ and $\mathbf{K}^{\text{lower}}$ are components of the stiffness matrix constructed from elements over and under the fault surface. For the FE mesh construction, we used a method that enabled automated and robust construction directly from digital elevation data of the crustal structure without creating a computer-aided design model. The geometric resolution of the FE model can be the same as that of the input digital elevation data, with a slight approximation of geometry based on a user-independent prescribed threshold. The mesh is generated within each grid of a background cell in the target domain. The generated mesh consists of quadratic tetrahedral elements and triangular-prism infinite elements (Zienkiewicz *et al.* 1983, Ichimura *et al.* 2014).

2.2 Structure of parallel finite element solver using GPU

In the simulation, a large proportion of the computation time is spent solving $\mathbf{Ku} = \mathbf{f}$. The crustal deformation computations result in the problem of having a large DOF (10^{8-10}); therefore, it is important to apply a faster solver. As solvers that use direct methods are too inefficient with regard to their memory usage, iterative solvers are required to conserve memory usage. We used an iterative solver based on the conjugate gradient (CG) method, which is one of the most commonly used iterative solvers. Appendix A shows an algorithm of a general CG solver. In this algorithm, all calculations are computed in double-precision format and many iterations are required.

To introduce CPU-GPU heterogeneous computing into this solver, we had to improve the original CG solver so that computations suitable for GPUs predominated. GPUs are accelerators with many cores that rapidly compute simple calculations via parallel computation using many threads. To support high performance in

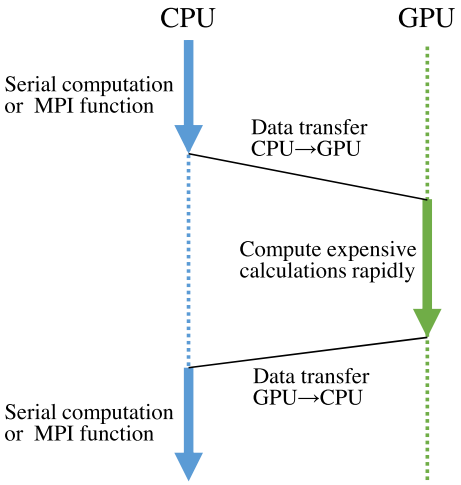

Figure 1. Rough scheme in CPU/GPU heterogeneous computing.

GPUs, computations must meet several requirements of the hardware. First, GPUs cannot perform computations alone, and require instructions to perform calculation or data transfers from CPUs. In CPU-GPU heterogeneous computing, CPUs control GPUs. Therefore CPUs are often called hosts, while GPUs are called devices. Moreover, GPUs must receive their data for computations from CPUs and send data necessary to CPUs for their computations. These processes are described as shown in Fig. 1. The bandwidth between CPUs and GPUs is generally much lower than the calculation performance of GPUs. Peripheral Component Interconnect (PCI) Express is typically used to transfer data between CPUs and GPUs. While improvements have been made in its performance, its bandwidth becomes saturated in massive computations. This latency is one of the main bottlenecks in GPU computing, and the amount of data transferred between CPUs and GPUs must be reduced to decrease computation times. In addition, some GPUs employ simultaneous instruction, multiple threads computation, where multiple threads perform the same procedures simultaneously to control many threads. For example, in NVIDIA GPUs, 32 cores compute simultaneously without explicit instructions. During memory accesses, GPUs are designed so that they exhibit high performance when multi-threads refer to consecutive memories. Consequently, random accesses to the device memory markedly degrade GPU performance. In addition, some GPUs do not perform double-precision calculations well. For example, the GPUs used in this study had one-third the performance for double-precision calculations compared to single-precision calculations. In recent years, the double-precision-to-single-precision computation performance ratio has improved. For example, in the NVIDIA Tesla P100 GPU, double-precision computation has achieved a half performance in single precision. However, it is desirable to reduce double-precision computations to increase the performance of accelerators that lack this capacity. Finally, device memory in GPUs ranges from several to tens of gigabytes. To handle large-scale problems, multiple GPUs are needed. Moreover, to remove data conflicts among GPUs, MPI parallel computation must be introduced, and GPUs must be bound to each MPI process. To reduce computation time, we aimed to reconstruct the algorithms in the iterative solver considering these constraints.

Algorithms 1 and 2 were proposed to solve $\mathbf{Ku} = \mathbf{f}$ quickly. Algorithm 1 is the CG algorithm for solving $\mathbf{Ku} = \mathbf{f}$, the final target equation. In the CG solver, we employed a method that uses

Algorithm 1 The iterative solver is carried out in each computation node to obtain a converged solution of $\mathbf{Ku} = \mathbf{f}$ using an initial solution, $\mathbf{u}$, with a threshold of $\|\mathbf{r}\|^2/\|\mathbf{f}\|^2 \leq \epsilon$. The input variables are: $\mathbf{K}$, $\overline{\mathbf{K}}$, $\overline{\mathbf{K}}_c$, $\overline{\mathbf{P}}$, $\mathbf{u}$, $\mathbf{f}$, ϵ, $\overline{\epsilon}_c^{\text{in}}$, $N_c^{\max}$, $\overline{\epsilon}^{\text{in}}$ and $N^{\max}$. The other variables in this solver are temporal. ($\bar{}$) represents the single-precision variables, while the others represent the double-precision variables. $\overline{\mathbf{P}}$ is a mapping matrix from the coarse model to the target model, which is defined by interpolating the components in each element of the coarse model.

Host (CPU)

$\mathbf{r} \Leftarrow \sum_i \mathbf{K}_e^i \mathbf{u}_e^i$ for some initial guess solution $\mathbf{u}$
synchronize $\mathbf{r}$ by point-to-point communication
$\mathbf{r} \Leftarrow \mathbf{f} - \mathbf{r}$
$\beta \Leftarrow 0$
$i \Leftarrow 1$
while $\|\mathbf{r}\|^2/\|\mathbf{f}\|^2 \leq \epsilon$ **do**
  $\overline{\mathbf{u}} \Leftarrow \overline{\mathbf{M}}^{-1}\mathbf{r}$
  $\overline{\mathbf{r}}_c \Leftarrow \overline{\mathbf{P}}^T\mathbf{r}$
  $\overline{\mathbf{u}}_c \Leftarrow \overline{\mathbf{P}}^T\overline{\mathbf{u}}$
  $\overline{\mathbf{K}}_c\overline{\mathbf{u}}_c = \overline{\mathbf{r}}_c$, solved by Algorithm 2 with $\overline{\epsilon}_c^{\text{in}}$ and $N_c^{\max}$
  $\overline{\mathbf{u}} \Leftarrow \overline{\mathbf{P}}\overline{\mathbf{u}}_c$
  $\overline{\mathbf{K}}\overline{\mathbf{u}} = \overline{\mathbf{r}}$, solved by Algorithm 2 with $\overline{\epsilon}^{in}$ and $N^{\max}$
  $\mathbf{z} \Leftarrow \overline{\mathbf{u}}$
  if $i > 1$ **then**
    $\gamma \Leftarrow (\mathbf{z}, \mathbf{q})$
    synchronize γ by collective communication
    $\beta \Leftarrow \gamma/\rho$
  end if
  $\mathbf{p} \Leftarrow \mathbf{z} + \beta\mathbf{p}$
  $\mathbf{q} \Leftarrow \sum_i \mathbf{K}_e^i \mathbf{p}_e^i$
  synchronize $\mathbf{q}$ by point-to-point communication
  $\rho \Leftarrow (\mathbf{z}, \mathbf{r})$
  synchronize ρ by collective communication
  $\gamma \Leftarrow (\mathbf{p}, \mathbf{q})$
  synchronize γ by collective communication
  $\alpha \Leftarrow \rho/\gamma$
  $\mathbf{q} \Leftarrow -\alpha\mathbf{q}$
  $\mathbf{r} \Leftarrow \mathbf{r} + \mathbf{q}$
  $\mathbf{u} \Leftarrow \mathbf{u} + \alpha\mathbf{p}$
  $i \Leftarrow i + 1$
end while

adaptive pre-conditioning Gene & Qiang (2000), which uses the CG method to solve the pre-conditioning of $\mathbf{Kz} = \mathbf{r}$ with a larger threshold value. Because this pre-conditioning CG loop is inside the original CG loop, it is herein referred to as the inner loop. By applying adaptive pre-conditioning, we did not have to store another pre-conditioning matrix, $\mathbf{M}$, thereby decreasing memory consumption. Because pre-conditioning does not require high accuracy, we used mixed-precision arithmetic consisting of double- and single-precision calculations, and introduced single-precision calculation into the pre-conditioning. In addition, the iteration of the inner loop is much larger than that of the outer loop, and the computation cost of the inner loop accounted for largest proportion of the solver. Introducing GPUs to the single-precision pre-conditioning enabled us to obtain high performance. Regarding double-precision calculations in the solver, the computational efficiency of GPUs per memory consumption is low, so double-precision calculations were computed using CPUs. For the CPU computations, we used

Algorithm 2 The iterative solver is carried out in each computation node to obtain the solution of $\overline{\mathbf{K}}\overline{\mathbf{u}} = \overline{\mathbf{r}}$ using an initial solution of $\overline{\mathbf{u}}$ with a threshold of $\|\overline{\mathbf{e}}\|^2/\|\overline{\mathbf{r}}\|^2 \leq \overline{\epsilon}$ or $N^{\max} < i$. This algorithm is carried out in a combination of CPUs and GPUs. The CPUs launch kernels in the GPUs and assign calculations that are computationally expensive to GPUs. The input variables are: $\overline{\mathbf{K}}$, $\overline{\mathbf{M}}$, $\overline{\mathbf{u}}$, $\overline{\mathbf{r}}$, $\overline{\epsilon}$ and $N^{\max}$. The other inputs in this solver are temporal. $\overline{\mathbf{K}}$ is stored in the block HYB format. The ELL and COO components are stored in the device memory. ($\bar{}$) represents the single-precision variables. We used a 3×3 block Jacobi matrix of $\overline{\mathbf{K}}$ for the pre-conditioning matrix, $\overline{\mathbf{M}}$, stored in the device memory.

	Host (CPU)		**Device** (GPU)
1:	send $\overline{\mathbf{u}}$, $\overline{\mathbf{r}}$ to Device	1:	receive $\overline{\mathbf{u}}$, $\overline{\mathbf{r}}$ from Host
2:		2:	$\overline{\mathbf{e}} \Leftarrow \overline{\mathbf{K}}\overline{\mathbf{u}}$
3:	receive $\overline{\mathbf{e}}$ from Device	3:	send $\overline{\mathbf{e}}$ to Host
4:	synchronize $\overline{\mathbf{e}}$ by point-to-point communication	4:	
5:	send $\overline{\mathbf{e}}$ to Device	5:	receive $\overline{\mathbf{e}}$ from Host
6:		6:	$\overline{\mathbf{e}} \Leftarrow \overline{\mathbf{r}} - \overline{\mathbf{e}}$
7:	$\overline{\beta} \Leftarrow 0$	7:	
8:	$i \Leftarrow 1$	8:	
9:	**while** $\|\overline{\mathbf{e}}\|^2/\|\overline{\mathbf{r}}\|^2 > \overline{\epsilon}$ and $N^{\max} > i$ **do**	9:	**while** $\|\overline{\mathbf{e}}\|^2/\|\overline{\mathbf{r}}\|^2 > \overline{\epsilon}$ and $N^{max} > i$ **do**
10:		10:	$\overline{\mathbf{z}} \Leftarrow \overline{\mathbf{M}}^{-1}\overline{\mathbf{e}}$
11:		11:	$\overline{\rho}_a \Leftarrow (\overline{\mathbf{z}}, \overline{\mathbf{e}})$
12:	receive $\overline{\rho}_a$ from Device	12:	send $\overline{\rho}_a$ to Host
13:	synchronize $\overline{\rho}_a$ by collective communication	13:	
14:	**if** $i > 1$ **then**	14:	
15:	$\overline{\beta} \Leftarrow \overline{\rho}_a/\overline{\rho}_b$	15:	
16:	**end if**	16:	
17:	send $\overline{\beta}$ to Device	17:	receive $\overline{\beta}$ from Host
18:		18:	$\overline{\mathbf{p}} \Leftarrow \overline{\mathbf{z}} + \overline{\beta}\overline{\mathbf{p}}$
19:		19:	$\overline{\mathbf{q}} \Leftarrow \overline{\mathbf{K}}\overline{\mathbf{p}}$
20:	receive $\overline{\mathbf{q}}$ from Device	20:	send $\overline{\mathbf{q}}$ to Host
21:	synchronize $\overline{\mathbf{q}}$ by point-by-point communication	21:	
22:	send $\overline{\mathbf{q}}$ to Device	22:	receive $\overline{\mathbf{q}}$ from Host
23:		23:	$\overline{\gamma} \Leftarrow (\overline{\mathbf{p}}, \overline{\mathbf{q}})$
24:	receive $\overline{\gamma}$ from Device	24:	send $\overline{\gamma}$ to Host
25:	synchronize $\overline{\gamma}$ by collective communication	25:	
26:	$\overline{\alpha} \Leftarrow \overline{\rho}_a/\overline{\gamma}$	26:	
27:	$\overline{\rho}_b \Leftarrow \overline{\rho}_a$	27:	
28:	send $\overline{\alpha}$ to Device	28:	receive $\overline{\alpha}$ from Host
29:		29:	$\overline{\mathbf{e}} \Leftarrow \overline{\mathbf{e}} - \overline{\alpha}\,\overline{\mathbf{q}}$
30:		30:	$\overline{\mathbf{u}} \Leftarrow \overline{\mathbf{u}} + \overline{\alpha}\,\overline{\mathbf{p}}$
31:	$i \Leftarrow i + 1$	31:	
32:	**end while**	32:	**end while**
33:	receive $\overline{\mathbf{u}}$ from Device	33:	send $\overline{\mathbf{u}}$ to Host

OpenMP thread paralleling in each MPI process. By introducing OpenMP/MPI hybrid paralleling, all available cores in CPUs can be used. The element-by-element (EBE) method is performed in CPUs to reduce computation cost. The EBE method computes the products of the global stiffness matrix and vectors without storing the matrix on the computer memory. The matrix-vector product with $\mathbf{K}$ in the CG algorithm is calculated as

$$\mathbf{Ku} = \sum_i \mathbf{Kt}_e^i \mathbf{u}_e^i + \sum_i \mathbf{Ki}_e^i \mathbf{u}_e^i, \tag{3}$$

where $\mathbf{Kt}_e^i$ and $\mathbf{Ki}_e^i$ are the element stiffness matrices for the ith unstructured tetrahedral and infinite elements. This approach significantly reduces the amount of memory necessary. We avoided applying the EBE method to the GPUs because it requires random accesses to device memories to generate element stiffness matrices and GPUs cannot demonstrate fast calculation performance when including random accesses. Using this framework, the convergence of solving $\mathbf{Kz} = \mathbf{r}$ is time-consuming, and many repetitive calculations are required. To reduce the computation time, we introduced methods to improve convergence of the inner loop. To reduce the computation time in the inner loop, we focused on MPI communication. We assumed that all communications in the computing environment are performed by the CPUs. Therefore, when data in the GPUs are communicated with other MPI processes, it must be transferred from the GPUs to the CPUs before MPI communication, and then transferred back to the GPUs after the necessary communication by the CPUs. The optimal algorithm requires that the communication between the host memory, controlled in the CPUs, and the device memory, controlled in the GPUs, is handled efficiently in MPI parallel computing and that the amount and frequency of data transferred among MPI processes is reduced. When communicating among processes, a method whereby each process sends and receives data that borders on neighbouring processes is generally used. This method of one-by-one communication reduces the amount of data transferred. The data transfer time depends on how the domain is split. In our method, we introduced METIS partitioning (Karypis & Kumar 1998) to optimize load balancing among processes. However, the bandwidth between the CPUs and GPUs remained saturated due to the scale of our target problem.Therefore, we introduced a multi-grid approach. Constructing a coarser FE

model in terms of the spatial resolution in the same target domain, we solve $\overline{\mathbf{K}}_c\overline{\mathbf{u}}_c = \overline{\mathbf{r}}_c$ using Algorithm 2. $(\ )_c$ and $\bar{(\ )}$ indicate variables defined in the coarse FE model and single-precision variables, respectively.

Using the coarse model, we obtained a solution that approximated the long-wavelength component of the original model solution. The resulting solution is used as the initial estimation in Algorithm 2 to solve $\overline{\mathbf{K}}\overline{\mathbf{z}} = \overline{\mathbf{r}}$. Then, the iteration for solving $\overline{\mathbf{K}}\overline{\mathbf{z}} = \overline{\mathbf{r}}$ could be reduced. The coarse model has a lower resolution than the fine model, so fewer nodes are necessary for the data transfer, and the amount of data transferred between the CPUs and GPUs is decreased. As a result, the bandwidth of PCI Express does not become saturated. Moreover, the amount of data transferred is equalized among processes due to the division of the domain using METIS. There is little difference in the computation time among processes, even if data transfer is instructed many times. The coarse FE model has less computation cost due to its smaller DOF, so it significantly reduces the computation cost for solving $\mathbf{Kz} = \mathbf{r}$. In this method, the coarse model is generated by converting the quadratic elements in the fine model into linear elements. Hence, mapping between the two models is calculated in the same node for each process. Data common to both models, such as material properties and node-connectivity, are unified and stored. Thus memory usage is reduced and large-scale problems can be computed in a GPU cluster. In these respects, the geometric multi-grid method is suitable to this algorithm using multiple GPUs.

As explained above, the solver consists of the outer loop with double-precision variables, the inner fine loop with single-precision variables from the original model, and the inner coarse loop with single-precision variables from the coarser model. Using GPUs to calculate these inner loops enables a large proportion of the computation to be calculated rapidly, significantly decreasing the overall computation time. In addition, it can exhibit good parallel efficiency by reducing MPI communication, even if many MPI processes are used. These algorithms are extensions of the computation method by Ichimura *et al.* (2016) that enables the solver to be carried out in a smaller computation environment using GPUs.

2.3 Calculation methods appropriate for GPUs

In the previous subsection, we discussed the structure of the solver. Next, we discuss calculation methods appropriate for GPUs. The global stiffness matrices of the coarse and fine FE models are stored in the GPU memory. These global stiffness matrices are sparse. Sparse matrix operations exhibit irregular access patterns in the device memory and limited purely by their bandwidth. In addition, the computation cost of sparse matrix vector products accounts for a large proportion of the total application time. Therefore, various types of data structures to store sparse matrices have been developed to improve the performance of GPU calculations. In this study, we applied a blocked hybrid (HYB) format to store matrices in GPUs. In the HYB format, elements in matrices are divided into two parts, an ELL format and a COO format, as noted by Bell & Garland (2008). A large proportion of non-zero elements are stored in the ELL format, and the rest are stored in the COO format. Fig. 2 illustrates the HYB format of an example matrix. The ELL format is suited to matrices obtained from well-structured grids, and the COO format offers consistent performance to all sparse matrices. The HYB format, which is a combination of the ELL and COO format, efficiently handles varying numbers of non-zero elements per row. The 3-D FE model in this computation splits the target domain based on background cells, reducing the dispersion of node connectivity. With this improvement, the HYB format becomes effectively applicable. The stiffness matrix contains non-zero elements in 3×3 blocks for each node. By blocking the HYB format into 3×3 blocks, we reduced the amount of data required to refer to the global stiffness matrix. Read-only memory was assigned to the vector multiplied by the matrix to increase the speed of memory access to the vector.

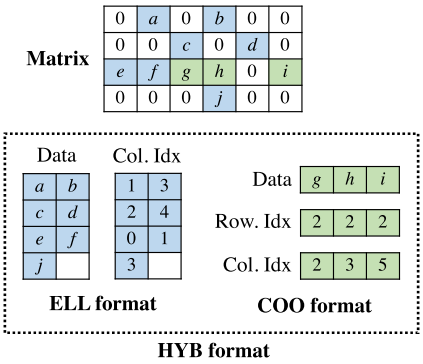

Figure 2. Example of the HYB format.

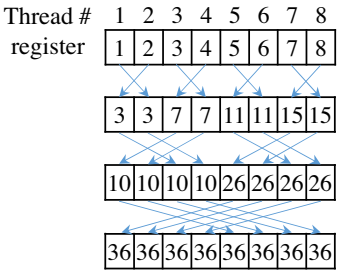

Figure 3. Example of inner product using warp shuffle functions. By referring value in other threads and adding up, we enable to obtain the result of inner vector products. This case uses eight threads, however, actual GPUs group 32 threads into one warp.

In addition, the solver has inner vector products. Paralleling in an inner vector product requires some synchronization among threads. GPUs run more than 10^3 threads, and redundant synchronization decreases performance. In this study, we use a mechanism whereby threads in the same warp are calculated with automatic synchronization. Several newer GPU models are equipped with the function, using a warp synchronization system called warp shuffle (Ukidave *et al.* 2015). By referring to and adding up values of registers in other threads, the total value in one warp is obtained without additional synchronization as shown in Fig. 3. We calculate the inner vector products by repeating the warp shuffle function.

Introducing geometric multi-grid method minimized the data transferred between the CPUs and GPUs, which is bottleneck of the computation, decreasing the amount of data transferred among other MPI processes. However, the date used for MPI communication reside at intervals. Transferring requested values individually leads to random accesses to device memories and causes performance deterioration of the solver. Therefore, we launched new kernels in GPUs that packed data necessary for MPI communication into a new array. Forming the new array and reordering the data in serial numbers result in coalesced accesses to device memories and faster transfers. The communicated array is returned to GPUs and summed using the atomic operation of the GPUs. Atomic operation is used for avoiding data race among different threads automatically and often degrades performance; however, in this case the decrease in performance is limited because conflicts among threads are

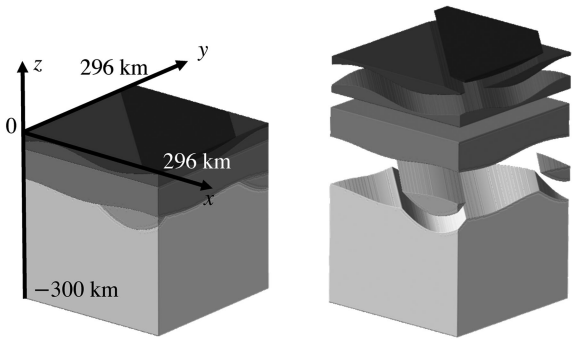

Figure 4. 3-D FE model of the numerical verification. The model is consisted of six layers which has the same material properties and their colours in greyscale represent each layer.

reduced due to halo communication. In addition, the transferred array is pinned so that data can be transferred faster.

3 NUMERICAL EXPERIMENT

For the numerical analysis, we used a GPU cluster with eight nodes. Each compute node of the GPU cluster has two CPUs (Intel Xeon E5-2695 v2) with 24 cores and eight GPUs (NVIDIA Tesla K40). We assigned one MPI process per GPU and performed OpenMP parallel computation using three cores per MPI process. The application was implemented using CUDA Fortran. InfiniBand FDR and PCI Express 3.0 × 16 were used as the interconnects. We decomposed the FE model into sub-domains for MPI processes using METIS. The FE model was decomposed into eight sub-domains and each sub-domain was assigned to one MPI process. One compute node of the GPU cluster performs one crustal deformation computation using eight MPI processes. To carry out the proposed solver, parameters for judging the convergence of the solution were set as follows: $\epsilon = 10^{-16}$, $\bar{\epsilon}^{\text{in}} = 1.6 \times 10^{-3}$, $\bar{\epsilon}_c^{\text{in}} = 1.6 \times 10^{-3}$, $N^{\max} = 200$, and $N_c^{\max} = 2000$. The threshold values of the inner loops were determined based on the resulting reduction in the overall computation time.

3.1 Numerical verification

We performed a numerical verification of the proposed method by computing the surface displacement against a fault slip in half-space. Young's modulus of the half-space was set to 72.3 GPa and Poisson's ratio was set to 0.243. A 3-D FE model was generated in the range of $0 \leq x \leq 296$ km, $0 \leq y \leq 296$ km, and $-300 \leq z \leq 0$ km. Infinite elements are placed on the boundary of target domain so that we approximate half-space condition. To verify the influence of mesh inhomogeneity, we generated six virtual layers and set the same material properties for all layers. Their discretization size in the space was 1000 m. The DOF, number of tetrahedral elements, infinite elements of the model were 156 519 072, 38 773 187, and 589 940, respectively. An overview of the model is shown in Fig. 4. The fault slip was located in the domain 133 km $\leq x \leq$ 163 km, 133 km $\leq y \leq$ 163 km, $z = -10$ km. The fault input had the x and y components. The input dislocations in the domain of 143 km $\leq x \leq$ 153 km and 143 km $\leq y \leq$ 153 km were 10 m and 2 m in the x and y components, respectively. The inputs in the surrounding area continuously decreased from the centre area toward the edge of the fault slip. We evaluated the solution obtained in the proposed method via a comparison with an analytical solution (Okada 1985).

The displacement on the surface of the domain ($0 \leq x \leq 296$ km, $0 \leq y \leq 296$ km, $z = 0$ km) and the error in comparison with an analytical solution are shown in Figs 5 and 6. The error for the analytical solution in the central area exhibited irregular results. This geometrical irregularity was assumed to be due to the use of quadratic tetrahedral elements. There was a slight difference in the FE solution between values on vertexes and values on midpoints. Nevertheless, this effect was much smaller than the error closer to the edges of the model. When we targeted only the area completely away from the side of the domain, we ensured sufficient accuracy of the obtained solution.

3.2 Application example

To demonstrate the usefulness of the proposed method, we generated a 3-D high-fidelity FE crustal model of northeastern Japan and computed the crustal deformation of a fault slip. The target domain of the numerical analysis is shown in Fig. 7. The domain sizes were 784, 976 and 400 km in the x (east–west), y (north–south) and z (up–down) directions, respectively. Ichimura *et al.* (2013) used the elevation data of Koketsu *et al.* (2008) to construct an FE model for almost the same region. The top surface of the Pacific Plate, where fault rupture is most likely to occur, does not reach the trench axis in this velocity structure data set. The edge of the rupture surface is usually assumed to be located at the trench axis when a shallow fault slip occurs. Therefore, we generated new elevation data by interpolating the trench axis and the top surface of the Pacific

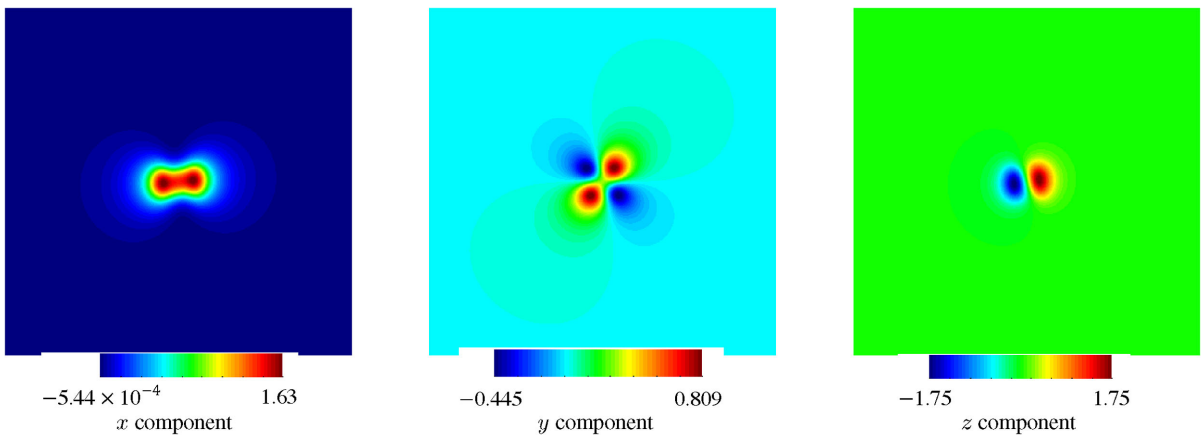

Figure 5. Distribution of the displacement response (m) in the numerical verification.

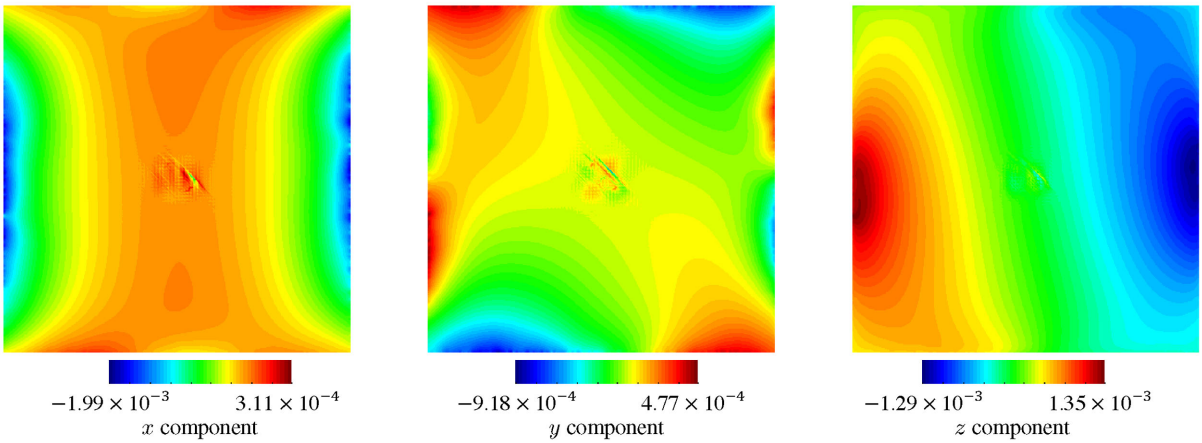

Figure 6. Error distribution (m) of the FE solution based on the solution by Okada (1985).

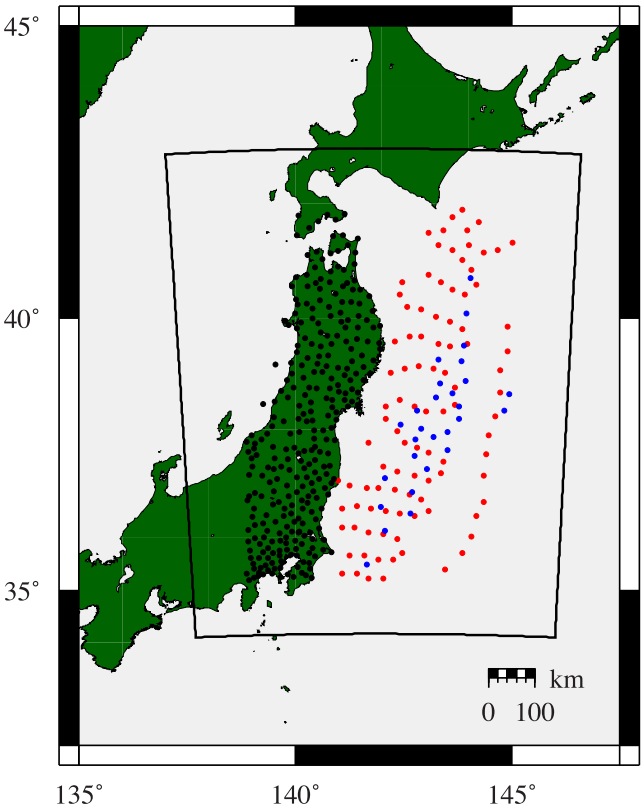

Figure 7. Target area of the FE models (black line) and virtual observation locations (black points: GEONET; blue points: GPS-A; red points: S-NET).

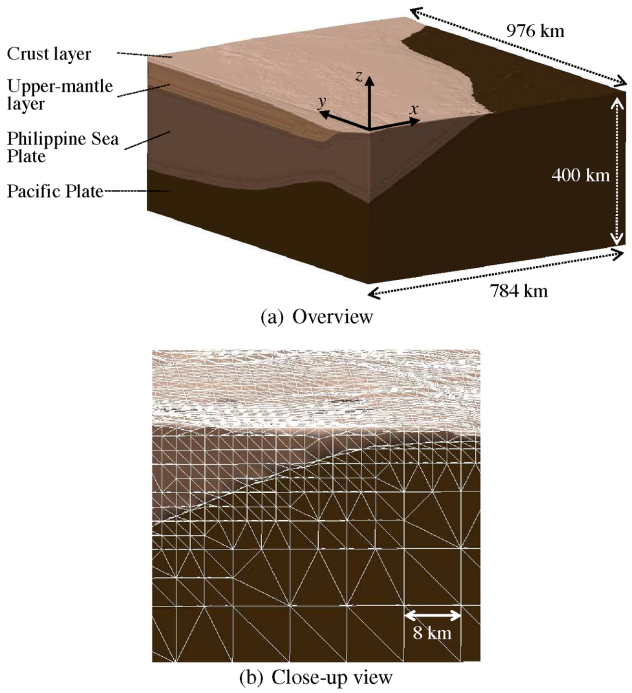

Figure 8. The FE model used for the application.

Plate, allowing the fault surface to reach the trench axis in the FE model constructed with this data. The model was composed of four layers, and its discretization size in space was 2000 m. Under these conditions, the DOF, number of quadratic tetrahedral elements, and number of triangular-prism infinite elements of the FE model were 82 196 106, 19 921 530 and 233 344, respectively. Fig. 8 shows a 3-D FE crustal model generated from the crustal geometry. Table 1 summarizes the original material properties of the model based on Sato *et al.* (2007). On the assumed fault surface in the FE model, we use a B-spline as the basic function of unit fault slip. The response of the surface is calculated with the proposed method toward the input fault slip.

Table 1. Original material properties of the four layers forming the model. V_p^{ori}: P-wave velocity (m s^{-1}), V_s^{ori} : S-wave velocity (m s^{-1}), ρ^{ori}: density (kg m^{-3}), E^{ori}: Young's modulus (Gpa), ν^{ori}: Poisson's ratio.

	V_p^{ori}	V_s^{ori}	ρ^{ori}	E^{ori}	ν^{ori}
Crust layer	5664	3300	2670	72.3	0.243
Upper-mantle layer	8270	4535	3320	176	0.285
Philippine Sea Plate	6686	3818	2600	95.4	0.258
Pacific Plate	6686	3818	2600	95.4	0.258

3.2.1 *Computation efficiency*

We estimated the computational efficiency of the proposed method via a comparison with the computation times of the previous method. For this computation, we considered only the computation time in the solver for one unit fault slip. First, we applied a standard CG solver and used a 3 × 3 block Jacobi matrix of $\overline{\mathbf{K}}$ for the pre-conditioning matrix, $\overline{\mathbf{M}}$. We compared the computation

Table 2. Computation time of the packing arrays for MPI communication and data transfer.

	Non-packing	Packing
Packing time (ms)	0	0.053
Data transfer time (ms)	30.623	0.181

times of the CPUs with those of the GPUs. The total computation time for the CPUs was 1306.100 s, while that of the GPUs was 251.828 s. In total, 2504 iteration loops were required. Ultimately, the GPUs accelerated the conventional CG solver only 5.2 times.

Next, we carried out the solver using adaptive pre-conditioning, and included the inner fine loops calculated in the GPUs. The resulting computation time was 152.271 s, which confirmed that the introduction of adaptive pre-conditioning in the single-precision computation decreased the computation time. However, the outer loop had six reiterations and the inner loop had 3008 reiterations, suggesting that there was room to improve convergence in the inner loop. Finally, we used the proposed solver consisting of the outer loop, inner fine loop, and inner coarse loop, which required 19.943 s to solve the same problem. The proposed solver had iteration of six outer loop, 148 inner coarse loop, and 1498 inner coarse loop iterations. Computing more reiterations in the inner loops, which had lower computation costs than the outer loop, drastically increased the computation speed of the proposed solver. Moreover, introducing the proposed method using GPUs resulted in a computation time 65 times faster than that of the conventional CG solver.

As described above, we were able to reduce the computation time by both reconstructing the algorithm of the solver and introducing GPUs into the solver. In this section, we discuss the computational efficiency of GPUs. We carried out the proposed solver without GPUs and measured each computation time. To solve $\overline{\mathbf{K}}\overline{\mathbf{u}} = \overline{\mathbf{r}}$, we calculated sparse matrix vector products and inner vector products. When calculating sparse matrix vector products in CPUs, the stiffness matrix is stored the Blocked Compressed Row Stored (BCRS) format, one of the most commonly used formats. In the GPUs, the matrix vector products and inner vector products were computed in 39.93 and 0.66 ms. Meanwhile, in the CPUs, these computations required 478.76 and 14.24 ms, respectively. The matrix vector products were calculated 12 times faster and the inner vector products were calculated 22 times faster by introducing GPUs when solving $\overline{\mathbf{K}}\overline{\mathbf{u}} = \overline{\mathbf{r}}$. Similarly, when solving $\overline{\mathbf{K}}_c\overline{\mathbf{u}}_c = \overline{\mathbf{r}}_c$, the matrix vector products were calculated 27 times faster and the inner vector products were calculated 11 times faster by introducing GPUs. To evaluate performance in our data structure, we measured the computation time in cases we used cuSPARSE (NVIDIA 2014). For sparse matrix vector products in cuSPARSE, only BCRS format can handle blocking of nonzero components. The computation time for the matrix $\overline{\mathbf{K}}_c$ was 5.51 ms in using cuSPARSE and 2.20 ms in using our codes. We confirmed that our data structure for the global stiffness matrix was so appropriate that our codes calculated the matrix vector product in 2.5 times shorter time than the library, which has been well-tuned for general sparse matrices. Regarding data transfer for MPI communication, we transferred all elements of the array for comparison. The transfer and packing times are shown in Table 2. When the array was packed by the GPUs and transferred in a minimum size, it required an additional 53 μs to pack each time, but the data transfer time was reduced by 1/169. In the inner fine loop, 1.05 MB of data was transferred per process, and the bandwidth is 11.9 GB s^{-1}. Because this is almost the same in terms of the whole vector, we considered the memory bandwidth to be saturated in the MPI communication of the inner fine loop. Conversely, in the inner coarse loop, 262 KB of data was transferred, and the bandwidth is 10.7 GB s^{-1}. The introduction of the inner coarse loop enabled the transfer of data between the CPUs and GPUs without saturation of bandwidth.

Table 3. Computation times and power consumption of the proposed solver.

	CPUs	CPU+GPUs
Pre-conditioning time (s)	158.673	14.553
Other time (s)	5.357	5.390
Total time (s)	164.030	19.943
Power consumption (W)	440	1516

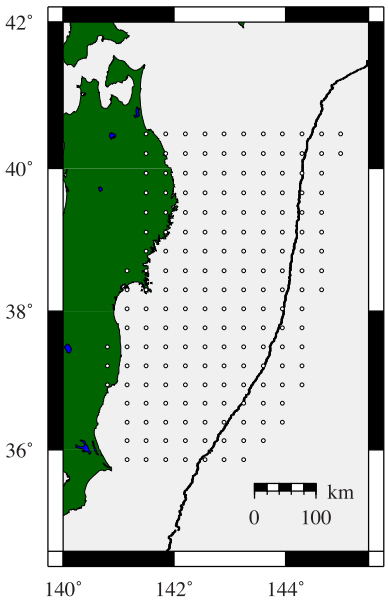

Figure 9. Locations of the grid points used as inputs for the basis functions.

Table 3 shows a comparison of the total computation times for the whole solver. In the solver, the adaptive pre-conditioning computation time was originally 158.673 s, accounting for 96.7 per cent of the total computation time. By introducing GPUs, we reduced the pre-conditioning computation time by 11 times, which resulted in a total computation time eight times lower for solving $\mathbf{Ku} = \mathbf{f}$. In addition, we measured the power consumed while carrying out the solvers for each computation environment (Table 3). By introducing GPUs into the proposed solver, power consumption increased 3.4 times, while the total computation speed increased 8.2 times. This suggests that the heterogeneous computing solver that we developed is more efficient in terms of computation performance per power consumption.

3.2.2 *Multiple computation example*

In general, modelling errors originating from Green's function have been a major problem in seismic source inversion (e.g. Yagi & Fukahata 2011; Minson *et al.* 2013; Duputel *et al.* 2014; Hallo & Gallovi 2016). The proposed solver enables the evaluation of the uncertainty of estimated earthquake source models with realistic modelling errors, while considering randomly perturbed 3-D structure models. In addition, the proposed solver has a short computation time. In this section, we demonstrate an estimation of the coseismic slip distribution of the 2011 Tohoku earthquake by conducting multiple crustal deformation computations. We used an

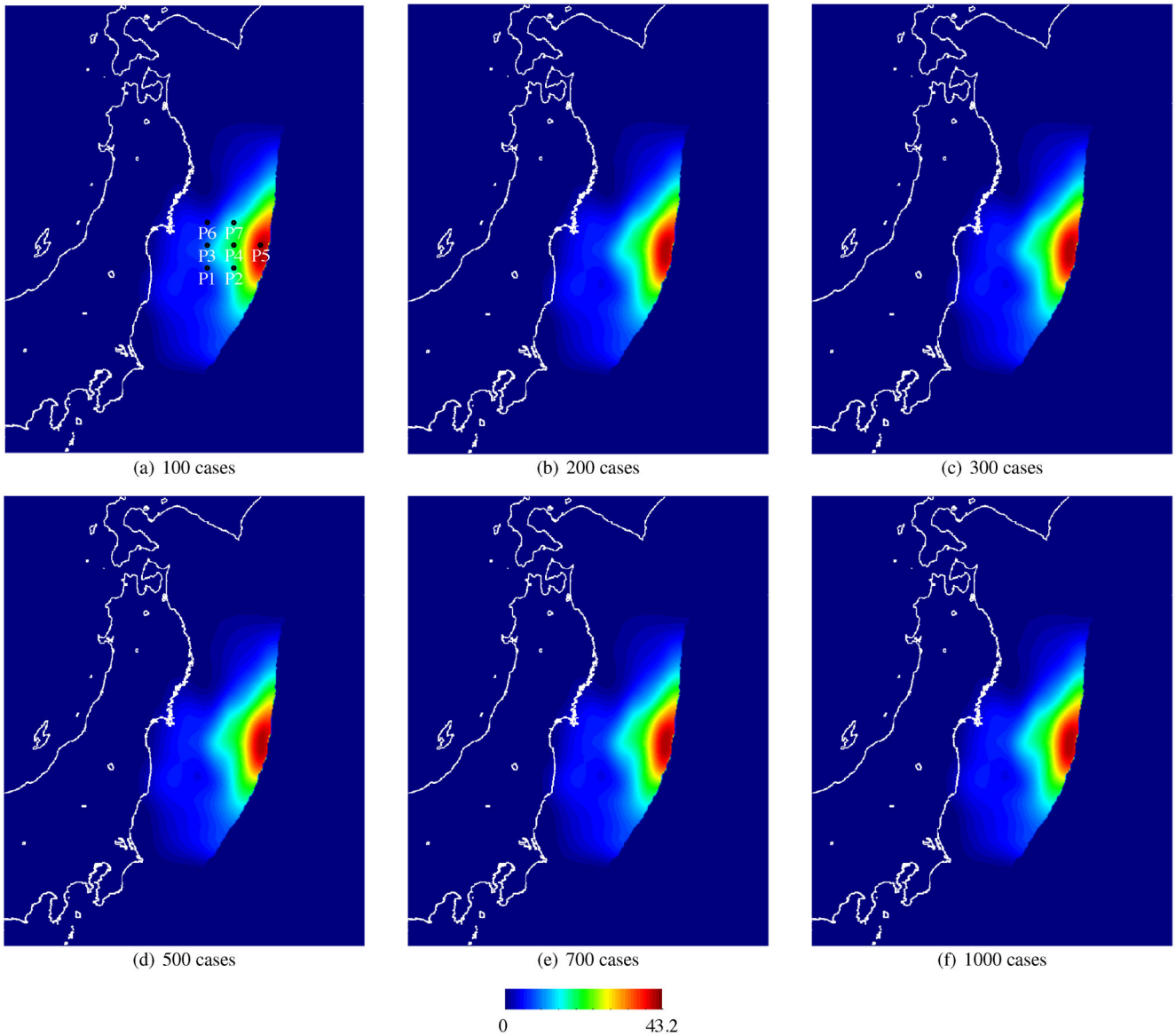

Figure 10. Average coseismic slip distribution (m). The white lines delineate the coastline.

inverse analysis method for the coseismic distribution estimation, which is described in detail in Appendix B. When performing this simulation, we used the same model as that used in the previous section. We applied bicubic B-spline-shaped unit fault slips to the FE model. The grid points used to input Green's functions were located at 30 km intervals within an area of 310 km $\leq x \leq$ 670 km and 200 km $\leq y \leq$ 710 km. This interval was in accordance with the subfault size of the coseismic slip distribution estimation of the 2011 Tohoku earthquake by Koketsu *et al.* (2011). When a small fault intersected the trench axis, its dislocations were input only on the layer boundaries. In total, there were 180 small faults (Fig. 9). Because we needed to compute Green's functions in the strike and the dip direction on the fault plane, the total number of Green's functions to compute was $180 \times 2 = 360$ per inverse analysis. We performed the inverse analysis using synthetic observational data extracted from GEONET, GPS-A at Tohoku University, and S-NET. We used the x, y and z components of displacement from GEONET, the x and y components from GPS-A, and the z component from S-NET. These settings for Green's functions and observational data were the same as those in the inversion analysis by Agata *et al.* (2016).

In this simulation, 1000 inversion analyses were performed and the MC approach was applied to the crustal deformation computation. We targeted at uncertainties in material properties of the model. Whereas this MC simulation may be converged faster, we analysed using typical and simple settings because the setting for uncertainties had much margin for discussion. Table 1 summarizes the original material properties of the model based on Sato *et al.* (2007), which were used as the standards to generate new sets of material properties with uncertainties. Regarding uncertainties in the underground structure model, we referred to velocity perturbation in seismic wave tomographic analyses. *P*-wave velocity has been postulated to vary within a range of 10 per cent of the original value based on a tomographic analysis by Matsubara & Obara (2011). Similarly, *S*-wave velocity has been postulated to vary within a range of 20 per cent of the original value based on the results of

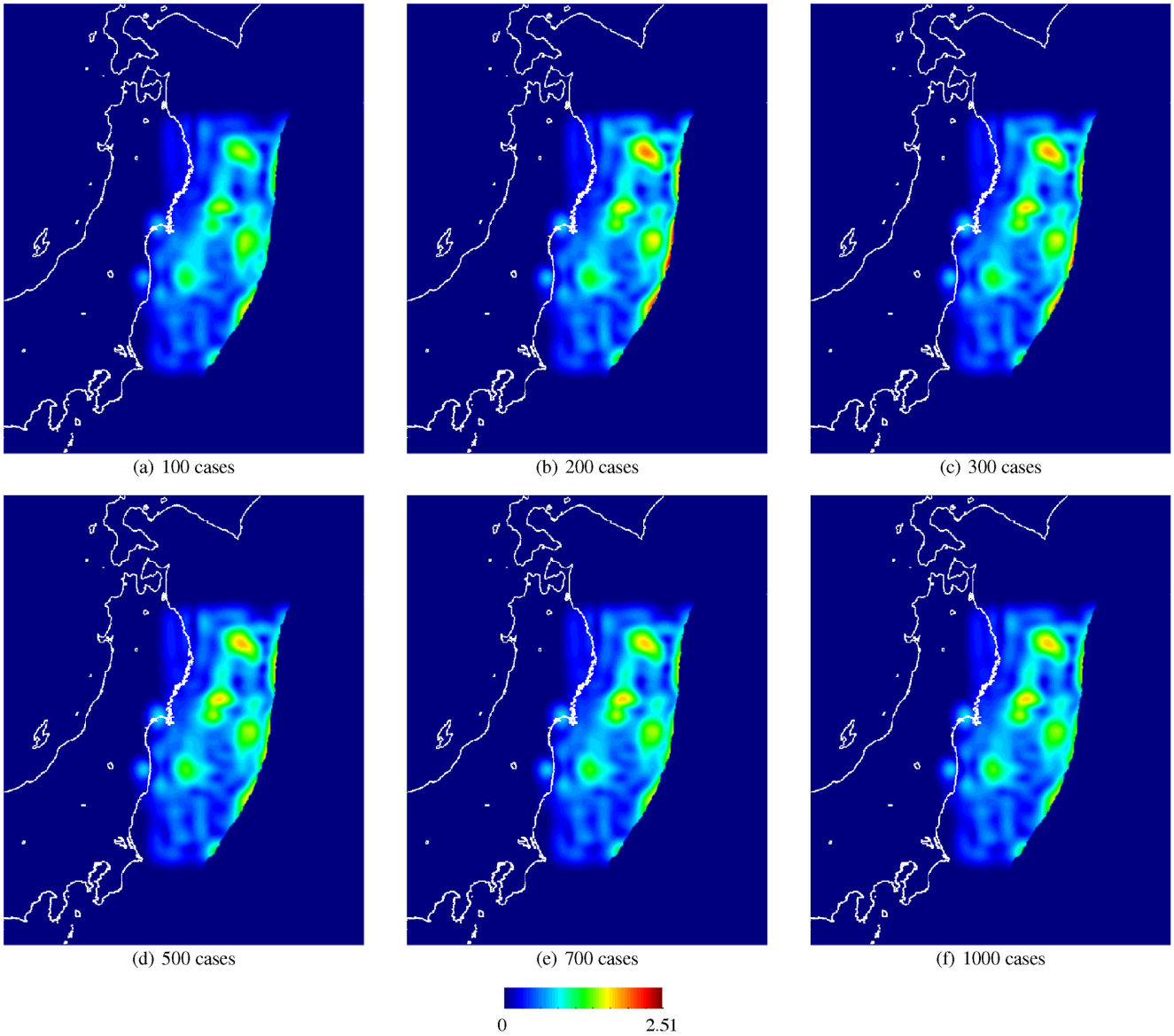

Figure 11. Standard deviation of the coseismic slip distribution (m).

Nishida *et al.* (2008). Density is usually derived from *P*-wave velocity; however, Brocher (2005) suggested that dispersion dependant on models was within 15 per cent of the original value. We used these assumptions to generate new material properties. We generated random seeds from a Gaussian distribution for each inversion analysis with varying material properties. The mean and standard deviation of the Gaussian distribution were 0 and $\frac{1}{3}$, respectively. We defined new material properties using random seeds as follows:

(1) $V_p(i) = V_p^{\mathrm{ori}}(i) \times (1 + 0.1r(i))$,
(2) $V_s(i) = V_s^{\mathrm{ori}}(i) \times (1 + 0.2r(i))$, and
(3) $\rho(i) = \rho^{\mathrm{ori}}(i) \times (1 + 0.15r(i))$,

where $V_p(i)$, $V_s(i)$, $\rho(i)$ and $r(i)$ are *P*-wave velocity, *S*-wave velocity, density and a random seed generated from the Gaussian distribution for the ith layer ($1 \leq i \leq 4$), respectively. In these settings, material properties in each layer are defined homogeneously; however, stiffness along the plate interfaces is inhomogeneous to some extent because the upper side of the fault is located on both Crust layer and Upper-mantle layer. We calculated a stochastic fault slip in the earthquake using 1000 Green's functions units derived from different material properties. In total, this simulation comprised 360 000 crustal deformation computations. In this application example, eight compute nodes calculated crustal deformation computations independently.

Fig. 10 shows the average coseismic slip distribution of 1000 cases, while Fig. 11 shows the standard deviation of the coseismic slip distribution of 1000 cases. The maximum displacement of the average distribution reached 41.18 m, while the standard deviation ranged from 0.00 m to 1.75 m. The standard deviation on each point may be related to the distribution of observation points. The location of observation points and standard deviation are described in Fig. 12. Areas where the standard deviation was relatively high were closely related to areas with few observation points.

The distributions of average and standard deviation continued to be almost flat through 1000 cases and appeared to have converged in the early stages. To verify the convergence of the MC method, we

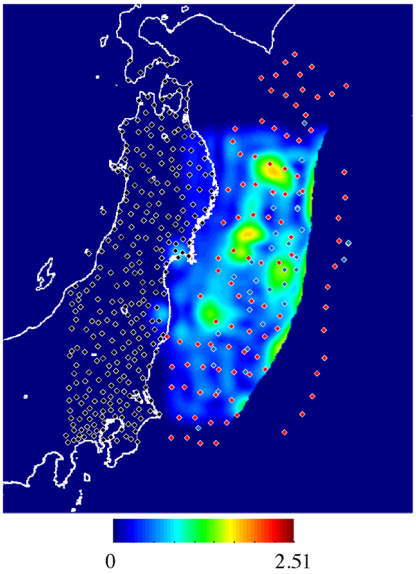

Figure 12. Relationship between standard deviation distribution in 1000 cases and observation points (black points: GEONET; blue points: GPS-A; red points: S-NET)

selected several points and confirmed the change of values. Fig. 10(a) shows the locations of the extracted points, while Fig. 13 shows the means and standard deviation of each point. Both the means and standard deviations almost unchanged after 600 cases, and the MC simulation obtained sufficient convergence by extracting 1000 material properties. We examined the effects of the differences in material properties and crustal structures. First, we compared the Green's function of one unit slip. We calculated the displacement on the surface using the original material properties, as well as the properties that resulted in the largest difference from the original slip distribution among the 1000 cases. For comparison, we generated an FE model with homogeneous material properties and a flat ground surface, and calculated for the same unit slip. Each Green's function is represented in Fig. 14. Case 1 and 2 used the model, which had a complex surface and material properties based on Sato *et al.* (2007) On the contrary, the model in case 3 has a flat surface and homogeneous material properties. Compared to cases 1 and 2, some vectors in the right side of the fault in case 3 pointed in opposite directions. This is because as for the term $\mathbf{K}^{\text{lower}}$ in split-node technique decreased and stiffness in the upper layer relatively increased. We confirmed that the uncertainty of the material properties and complexity of the crustal structures affected the Green's functions. Next, we compared coseismic slip distributions. The results derived from the original material properties and those that resulted in the largest difference in slip displacement among 1000 cases were compared.

The slip distribution calculated using the original material properties is shown in Fig. 15(a), and the difference in slip distribution between the original case and the extracted case is shown in Fig. 15(b). The difference was 5–6 m due to uncertainties in the material properties in areas without GPS-A stations (Fig. 7). This difference was negligible near the maximum coseismic slip area. However, the large difference in Fig. 15(b) occurred in the area where the slip is less than 10 m in Fig. 15(a). This difference cannot be negligible when discussing the coseismic slip distribution and related stress change distribution. In the source area of the M9 Tohoku earthquake, M7-8 earthquakes occurred repeatedly.

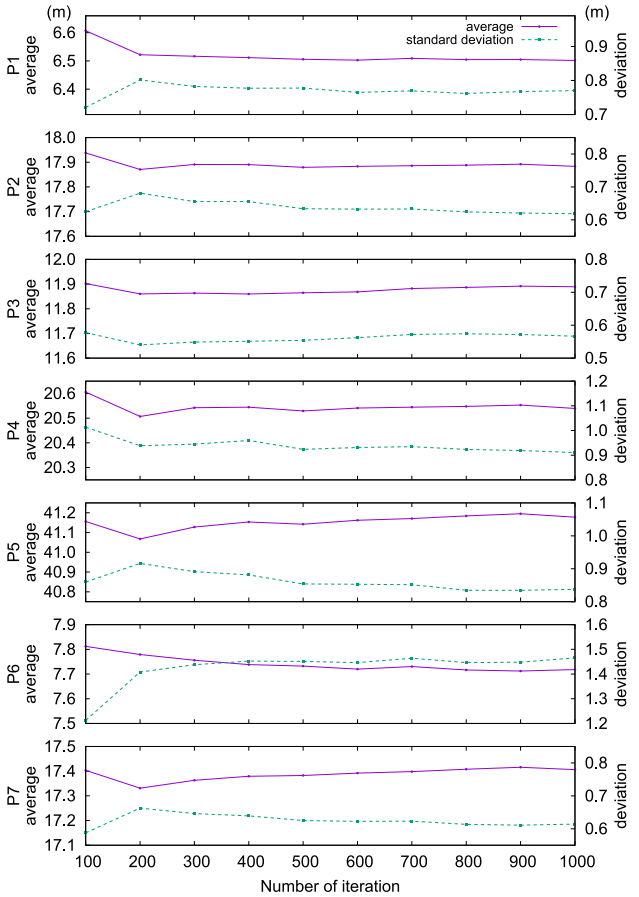

Figure 13. Point-wise convergence of the slip distribution using the MC method.

Therefore, there has been concern over the slip amount at the M7-8 source areas during the M9 earthquake. In particular, it has been questioned if the slip amount was larger than that during other M7-8 earthquake (e.g. Nakata *et al.* 2016). Based on our results, it is necessary to evaluate the model errors due to uncertainties (e.g. material properties) to discuss such a small slip amount. Furthermore, the results suggest that a high-density distribution of stations with both S-net and GPS-A stations is required to evaluate slip distribution and the amount of slip for earthquakes in the Tohoku area.

Our proposed method, which enables the rapid computation of many Green's functions for 3-D heterogeneous structures will contribute to solving these problems. Using our proposed method, we completed a calculation of the crustal deformation computation 360 000 times in 17 d.

4 CONCLUDING REMARKS

We developed an elastic crustal deformation computation method using a CPU-GPU heterogeneous computing solver. 3-D FE method at high resolution originally requires much computation cost than other methods based on analytical solution. On the other hand, this approach enables to compute for a crustal structure that has complex geometry by setting out many nodes around the source and our method enabled to reduce computation cost by using multiple GPUs. To test the proposed method, we constructed an FE model with 82 196 106 DOF for northeastern Japan and computed the elastic crustal deformation computation 360 000 times in 17 d using

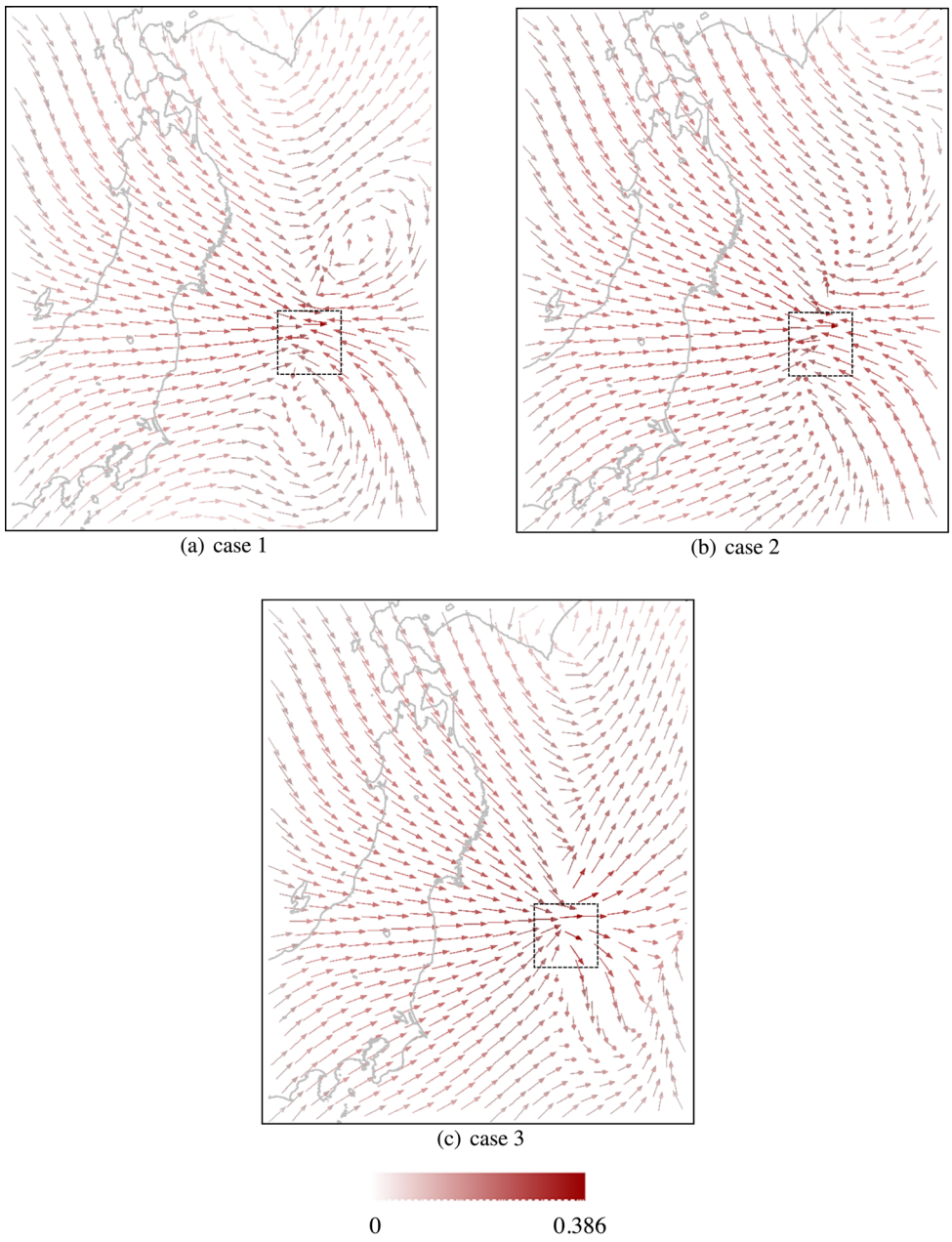

Figure 14. Results of one Green's function derived from (a) the original material properties, (b) one case of the 1000 cases and (c) an FE model with a flat surface and homogeneous material properties. Each arrow represents the direction and degree (m) of slip and grey lines show the coastline. The colour scale is logarithmic. The area inside the dashed box is the location of the input fault slip.

a GPU cluster comprising 16 CPUs and 64 GPUs. The proposed method enabled the crustal deformation computation using high-fidelity crustal structure data 65 times faster than the conventional method. Using this method, a stochastic estimation of coseismic slip distribution, with uncertainties in material properties, was computed within a realistic time frame. In addition, our proposed method can be useful to other simulations including stress calculation such as post-seismic deformation and earthquake cycles by reconfiguring the resolution of FE model. For instance, the importance of elastic stress response functions has been proposed in the works by Barbot & Fialko (2010) and Barbot *et al.* (2017), which include viscoelasticity and poroelasticity response computation. Their methods can be expanded to a crustal structure that has complex geometry with heterogeneous elastic property, by introducing our proposed method. Also, the framework of the solver in this paper is used in the viscoelastic crustal deformation computation by Ichimura *et al.* (2016). Thus we are expected to reduce computation cost of this calculation similarly by replacing the original solver with our proposed solver. These advances will contribute to our understanding of the physics of earthquake generation, estimation of crustal structures and prediction of possible earthquakes. In future studies, we will decrease the computation time for FE mesh construction and consider uncertainties in geometry. Our proposed method is supposed to be more useful for such a complex setting.

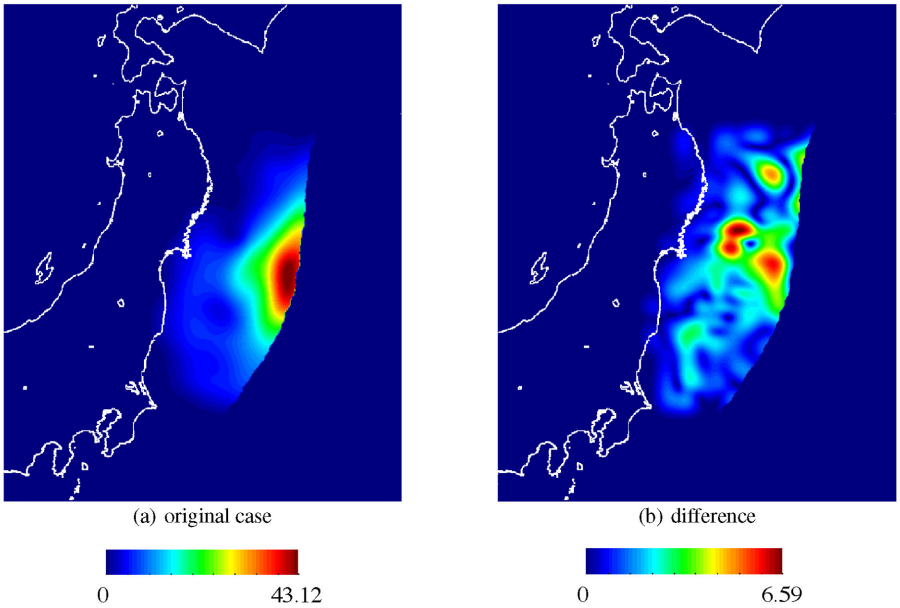

Figure 15. Comparison of two cases with different material properties. (a) The slip distribution using original material properties (m). (b) The largest difference in 1000 cases from the original slip distribution (m).

ACKNOWLEDGEMENTS

We gratefully acknowledge Prof K. Hirahara and the anonymous reviewer whose reviews helped us improve the manuscript. We also thank Dr Y. Okada for providing the code used in our numerical verification. This work was supported by Post K computer project (priority issue 3: Development of Integrated Simulation Systems for Hazard and Disaster Induced by Earthquake and Tsunami), Japan Society for the Promotion of Science (KAKENHI Grant Numbers 26249066) and FOCUS Establishing Supercomputing Center of Excellence.

REFERENCES

Agata, R., Ichimura, T., Hirahara, K., Hyodo, M., Hori, T. & Hori, M., 2016. Robust and portable capacity computing method for many finite element analyses of a high-fidelity crustal structure model aimed for coseismic slip estimation, *Comput. Geosci.,* **94,** 121–130.

Barbot, S. & Fialko, Y., 2010. A unified continuum representation of post-seismic relaxation mechanisms: semi-analytic models of afterslip, poroelastic rebound and viscoelastic flow, *Geophys. J. Int.,* **182**(3), 1124–1140.

Barbot, S., Moore, J.D. & Lambert, V., 2017. Displacement and stress associated with distributed anelastic deformation in a half-space, *Bull. seism. Soc. Am.,* **107**(2), 821–855.

Bell, N. & Garland, M., 2008. Efficient sparse matrix-vector multiplication on CUDA, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, **2**(5).

Brocher, T.M., 2005. Empirical relations between elastic wavespeeds and density in the Earth's crust, *Bull. seism. Soc. Am.,* **95,** 2081–2092.

Duputel, Z., Agram, P.S., Simons, M., Minson, S.E. & Beck, J.L., 2014. Accounting for prediction uncertainty when inferring subsurface fault slip, *Geophys. J. Int.,* **197**(1), 464–482.

Fukuda, J. & Johonson, K.M., 2008. A fully Bayesian inversion for spatial distribution of fault slip with objective smoothing, Bulletin of the Seismological Society of America, **98**(3), 1128–1146.

Gene, H.G. & Qiang, Y., 2000. Inexact preconditioned conjugate gradient method with inner-outer iteration, *J. Sci. Comput.,* **21,** 1305–1320.

Geospatial Information Authority of, Japan, 2010. 'GNSS earth observation network system'. Available at: http://terras.gsi.go.jp/geo_info/geonet_top.html, last accessed May 2017.

Hallo, M. & Gallovic, F., 2016. Fast and cheap approximation of Green function uncertainty for waveform-based earthquake source inversions, *Geophys. J. Int.,* **207,** 1012–1029.

Hansen, P.C., 1992. Analysis of discrete ill-posed problems by means of the L-curve, *SIAM Rev.,* **34**(4), 561–580.

Hashimoto, C., Noda, A. & Matsu'ura, M., 2012. The Mw 9.0 northeast Japan earthquake: total rupture of a basement asperity, *Geophys. J. Int.,* **189**(1), 1–5.

Hughes, K., Masterlark, T. & Moonry, W., 2010. Poroelastic stress-triggering of the 2005 M8.7 Nias earthquake by the 2004 M9.2 Sumatra-Andaman earthquake, *Earth planet. Sci. Lett.,* **293,** 289–299.

Ichimura, T., Agata, R., Hori, T., Hirahara, K. & Hori, M., 2013. Fast numerical simulation of crustal deformation using a three-dimensional high-fidelity model, *Geophysical Journal International,* **195**(3), 1730–1744.

Ichimura, T., Agata, R., Hori, T., Hirahara, K., Hashimoto, C., Hori, M. & Fukahata, Y., 2016. 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.

Japan Seismic Hazard Information Station, 2005. 'National Research Institute for Earth Science and Disaster Resilience'. Available at: http://www.j-shis.bosai.go.jp/download, last accessed May 2017.

Karypis, G. & Kumar, V., 1998. A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.,* **20**(1), 359–392.

Koketsu, K., Miyake, H., Fujiwara, H. & Hashimoto, T., 2008. Progress towards a Japan Integrated Velocity Structure Model and Long-Period Ground Motion Hazard Map, in *14th World Conference on Earthquake Engineering,* Beijing, China, pp. 1–7.

Koketsu, K. *et al.*, 2011. A unified source model for the 2011 Tohoku earthquake, *Earth planet. Sci. Lett.,* **310**(3–4), 480–487.

Masterlark, T., 2003. Finite element model predictions of static deformation from dislocation sources in a subduction zone: Sensitivities to homogeneous, isotropic, Poisson-solid, and half-space assumptions, *J. geophys. Res.,* **108**(B11), 2540, doi:10.1029/2002JB002296.

Matsubara, M. & Obara, K., 2011. The 2011 Off the Pacific Coast of Tohoku earthquake related to a strong velocity gradient with the Pacific plate, *Earth Planets Space,* **63,** 663–667.

Melosh, H. & Raefsky, A., 1981. A simple and efficient method for introducing faults into finite element computations, *Bull. seism. Soc. Am.,* **71**(5), 1391–1400.

Minson, S.E., Simons, M. & Beck, J.L., 2013. Bayesian inversion for finite fault earthquake source models I—theory and algorithm, *Geophys. J. Int.,* **194**(3), 1701–1726.

Miyazaki, H., Kusano, Y., Shinjou, N., Shoji, F., Yokokawa, M. & Watanabe, T., 2012. Overview of the K computer system, *Fujitsu Sci. Tech. J.,* **48**(3), 302–309.

Nakata, R., Hori, T., Hyodo, M. & Ariyoshi, K., 2016. Possible scenarios for occurrence of M 7 interplate earthquakes prior to and following the 2011 Tohoku-Oki earthquake based on numerical simulation, *Sci. Rep.,* **6,** 25704.

Nishida, K., Kawakatsu, H. & Obara, K., 2008. Three-dimensional crustal *S* wave velocity structure in Japan using microseismic data recorded by Hi-net tiltmeters, *J. geophys. Res.,* **113**(B10).

NVIDIA, 2014. CUSPARSE library. NVIDIA Corporation, Santa Clara, California.

Okada, Y., 1985. Surface deformation due to shear and tensile faults in a half-space, *Bull. seism. Soc. Am.,* **75**(4), 1135–1154.

Ryoo, S., Rodrigues, C.I., Baghsorkhi, S.S., Stone, S.S., Kirk, D.B. & Hwu, W.M.W., 2008. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,* ACM, pp. 73–82.

Sato, M., Minagawa, N., Hyodo, M., Baba, T., Hori, T. & Kaneda, Y., 2007. Effect of elastic inhomogeneity on the surface displacements in the northeastern Japan: based on three-dimensional numerical modeling, *Earth Planets Space,* **59,** 1083–1093.

Sato, M., Ishikawa, T., Ujihara, N., Yoshida, S., Fujita, M., Mochizuki, M. & Asada, A., 2011. Displacement above the hypocenter of the 2011 Tohoku-Oki earthquake, *Science,* **332**(6036), 1395.

Sedaghati, N., Mu, T., Pouchet, L.N., Parthasarathy, S. & Sadayappan, P., 2015. Automatic selection of sparse matrix representation on GPUs, in *Proceedings of the 29th ACM on International Conference on Supercomputing,* ACM, pp. 99–108.

Ukidave, Y., Paravecino, F.N., Yu, L., Kalra, C., Momeni, A., Chen, Z. & Kaeli, D., 2015. Nupar: a benchmark suite for modern GPU architectures, in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering,* ACM, pp. 253–264.

Watanabe, S., Sato, M., Fujita, M., Ishikawa, T., Yokota, Y., Ujihara, N. & Asada, A., 2014. Evidence of viscoelastic deformation following the 2011 Tohoku-oki earthquake revealed from seafloor geodetic observation, *Geophys. Res. Lett*., **41,** 5789–5796.

Yagi, Y. & Fukahata, Y., 2011. Introduction of uncertainty of Green's function into waveform inversion for seismic source processes, *Geophys. J. Int.,* **186**(2), 711–720.

Zienkiewicz, O.C., EMson, C. & Bettess, P., 1983. A novel boundary infinite element, *Int. J. Numer. Methods Eng.,* **19,** 393–404.

Zienkiewicz, O.C., Taylor, R.L. & Zhu, J.Z., 2005. *The Finite Element Method: Its Basis and Fundamentals,* 6th edn, Butterworth-Heinemann.

APPENDIX A: CONJUGATE GRADIENT METHOD

The algorithm for the conjugate gradient (CG) method is shown in Algorithm 3. To reach the solution of the linear system, matrix-vector multiplication, inner products and single-precision A·X plus Y (SAXPY) are calculated many times. A pre-conditioning matrix, **M**, is usually used to improve the convergence of the solver.

Algorithm 3 The iterative solver performed in each computation node to obtain a converged solution, $\mathbf{Ku} = \mathbf{f}$, using an initial solution, $\mathbf{u}$, with a threshold of $\|\mathbf{r}\|^2/\|\mathbf{f}\|^2 \leq \epsilon$. The input variables are $\mathbf{K}$, $\mathbf{u}$, $\mathbf{f}$, ϵ. The other variables in this solver are temporal. $\mathbf{M}$ is the pre-conditioning matrix.

Host (CPU)
1: $\mathbf{r} \Leftarrow \mathbf{Ku}$ for some initial guess solution $\mathbf{u}$
2: $\mathbf{r} \Leftarrow \mathbf{f} - \mathbf{r}$
3: $\beta \Leftarrow 0$
4: $i \Leftarrow 1$
5: **while** $\|\mathbf{r}\|^2/\|\mathbf{f}\|^2 \leq \epsilon$ **do**
6: $\quad\mathbf{z} \Leftarrow \mathbf{M}^{-1}\mathbf{r}$
7: $\quad$**if** $i > 1$ **then**
8: $\quad\quad\gamma \Leftarrow (\mathbf{z}, \mathbf{q})$
9: $\quad\quad\beta \Leftarrow \gamma/\rho$
10: $\quad$**end if**
11: $\quad\mathbf{p} \Leftarrow \mathbf{z} + \beta\mathbf{p}$
12: $\quad\mathbf{q} \Leftarrow \mathbf{Kp}$
13: $\quad\rho \Leftarrow (\mathbf{z}, \mathbf{r})$
14: $\quad\gamma \Leftarrow (\mathbf{p}, \mathbf{q})$
15: $\quad\alpha \Leftarrow \rho/\gamma$
16: $\quad\mathbf{q} \Leftarrow -\alpha\mathbf{q}$
17: $\quad\mathbf{r} \Leftarrow \mathbf{r} + \mathbf{q}$
18: $\quad\mathbf{u} \Leftarrow \mathbf{u} + \alpha\mathbf{p}$
19: $\quad i \Leftarrow i + 1$
20: **end while**

APPENDIX B: INVERSE ANALYSIS FORMULATION

Fault slip distribution is modelled from crustal deformation data corresponding to m observation points on the ground surface. The fault slip distribution is composed of a linear combination of a finite number of basic functions. Unit fault slips are introduced in two directions on $n/2$ small faults, where n is an even number distributed over the fault surface for the basic functions in the formulation. Because crustal deformation is modelled as a linear elastic deformation, the fault slip estimation is formulated using a linear equation:

$$\mathbf{Gx} = \mathbf{d}, \tag{B1}$$

where $\mathbf{G}$ is an $m \times n$ matrix of Green's function, $\mathbf{x}$ is an n-dimensional vector of unknown displacement for each subfault, and $\mathbf{d}$ is an m-dimensional vector of the observed displacements. The components in the $\mathbf{G}$ matrix, $[g_{1i}, g_{2i}, \cdots, g_{mi}]^T$, are calculated by inputting the unit fault slip as the source fault in the FE model. Thus, an FE analysis with a 10^7-order DOF model must be computed n times to construct $\mathbf{G}$. In this study, n was in the order of $10^2 - 10^3$. The inversion defined by eq. (B1) is usually ill-posed. We regularized it using a smoothness constraint on $\mathbf{x}$ with a discrete Laplacian operator. This constraint can be represented in matrix notation $\mathbf{Lx} = \mathbf{0}$ where $\mathbf{L}$ is an $n \times n$ matrix. The target equation is rewritten as

$$\begin{pmatrix} \mathbf{G} \\ \alpha\mathbf{L} \end{pmatrix} \mathbf{x} = \begin{pmatrix} \mathbf{d} \\ \mathbf{0} \end{pmatrix}, \tag{B2}$$

where α is the weighting factor of the smoothness constraint, and is determined using the L-curve method Hansen (1992).