Viscoelastic Crustal Deformation Computation Method with Reduced Random Memory Accesses for GPU-based Computers

Takuma Yamaguchi[1], Kohei Fujita[1,2], Tsuyoshi Ichimura[1,2], Anne Glerum[3], Ylona van Dinther[4], Takane Hori[5], Olaf Schenk[6], Muneo Hori[1,2], and Lalith Wijerathne[1,2]

[1] Earthquake Research Institute and Department of Civil Engineering, The University of Tokyo, Bunkyo, Tokyo, Japan
{yamaguchi, fujita, ichimura, hori, lalith}@eri.u-tokyo.ac.jp
[2] Advanced Institute for Computational Science, RIKEN, Kobe, Japan
[3] Helmholtz-Centre Potsdam, GFZ German Research Centre for Geosciences, Potsdam, Germany
acglerum@gfz-potsdam.de
[4] Institute of Geophysics, ETH Zurich, Zurich, Switzerland
ylona.vandinther@erdw.ethz.ch
[5] Research and Development Center for Earthquake and Tsunami, Japan Agency for Marine-Earth Science and Technology, Yokosuka, Japan
horit@jamstec.go.jp
[6] Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
olaf.schenk@usi.ch

Abstract. The computation of crustal deformation following a given fault slip is important for understanding earthquake generation processes and reduction of damage. In crustal deformation analysis, reflecting the complex geometry and material heterogeneity of the crust is important, and use of large-scale unstructured finite-element method is suitable. However, since the computation area is large, its computation cost has been a bottleneck. In this study, we develop a fast unstructured finite-element solver for GPU-based large-scale computers. By computing several times steps together, we reduce random access, together with the use of predictors suitable for viscoelastic analysis to reduce the total computational cost. The developed solver enabled 2.79 times speedup from the conventional solver. We show an application example of the developed method through a viscoelastic deformation analysis of the Eastern Mediterranean crust and mantle following a hypothetical M 9 earthquake in Greece by using a 2,403,562,056 degree-of-freedom finite-element model.

Keywords: CUDA, Finite Element Analysis, Conjugate Gradient Method

1 Introduction

One of the targets of solid earth science is the prediction of the place, magnitude, and time of earthquakes. One approach to this target is to estimate earthquake

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

occurrence probability by comparing the current plate conditions with plate conditions when past earthquakes have occurred [9]. In this process, inverse analysis is required to estimate the current inter-plate displacement distribution using the crustal deformation data observed at the surface. In order to realize this inverse analysis, forward analysis methods computing elastic and viscoelastic crustal deformation for a given inter-plate slip distribution are under development.

In previous crustal deformation analyses, simplified models such as horizontally stratified layers were used [8]. However, recent studies point out that the simplification of crustal geometry has significant effects on the response [11]. Recently, 3D crust property data as well as crustal deformation data measured at observation stations are being accumulated. Thus, 3D crustal deformation analyses reflecting these data in full resolution are being anticipated.

The 3D finite-element method is capable of modeling 3D geometry and material heterogeneity of the crust. However, modeling the available 1 km resolution crust property data fully into 3D finite-element crustal deformation analysis leads to large computational problems with more than 10^9 degrees-of-freedom. Thus, acceleration of this analysis using high-performance computers is required. Targeting the elastic crustal deformation analysis problem, we have been developing unstructured finite-element solvers suitable for GPU-based high-performance computers by developing algorithms considering the underlying hardware [7]. When compared with elastic analysis, viscoelastic analysis requires solving many time steps and thus its computational cost becomes even larger; therefore we target further acceleration of this solver in this paper.

Due to its high floating point performance, GPUs generally have relatively low memory bandwidth. Furthermore, data transfer performance is further decreased when memory access is not coalesced. Finite-element analysis mainly consists of memory bandwidth bound kernels, and the most computationally expensive sparse matrix-vector product kernel has many random memory accesses. Thus, it is not straight forward to utilize the high arithmetic capability of GPUs in finite-element solvers. Reduction of data transfer and random access is important to improve computational efficiency. In this study, we accelerate the previous GPU solver by introducing algorithms that reduce data transfer by reduction of solver iterations, and reduce random access of the major computational kernels. Here we use a multi-time step method together with a predictor to obtain the initial solution of the iterative solver. We improve the convergency of the iterative solver by adapting the predictor to the characteristic of solutions for the viscoelastic problem. In addition, by using several vectors for computation, we can reduce random memory access in the major sparse matrix-vector kernel and improve performance.

Section 2 explains the developed method. Section 3 shows the performance of the developed method on Piz Daint [4], which is a P100 GPU based supercomputer system. Section 4 shows an application example using the developed method. Section 5 summarizes the paper and gives future prospects.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

2 Methodology

We target elastic and viscoelastic crustal deformation to a given fault slip. Following [8], the governing equation is

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

with

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

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

where σ_{ij} and f_i are the stress tensor and outer force. $\dot{(\ )}$, $(\ )_{,i}$, δ_{ij}, η, ϵ_{ij}, and u_i are the first derivative in time, spatial derivative in the i-th direction, Kronecker delta, viscosity coefficient, strain tensor, and displacement, respectively. λ and μ are Lame's constants. Discretization of this equation by the finite-element method leads to solving a large system of linear equations. For a solver, (i) good convergency and (ii) small computational cost in each kernel are basically required to reduce the time-to-solution. The proposed method considering these requirements is based on viscoelastic analysis by [10], which can be described as follows (Algorithm 1 and 2).

An adaptive preconditioned conjugate gradient solver with Element-by-Element method [13], multi-grid method, and mixed-precision arithmetic is used in Algorithm 2. Most of the computational cost is in the inner loop of Algorithm 2. It can be computed in single precision, and we can reduce computational cost and data transfer size; thereby we can expect it to be suitable for GPU systems. In addition, we introduce the multi-grid method and use a coarse model to estimate the initial solution for the preconditioning part. This procedure reduces the whole computation cost in the preconditioner as the coarse model has less degrees-of-freedom compared to the target model. Below, we call line 7 of Algorithm 2(a) as the inner coarse loop and line 9 of Algorithm 2(a) as the inner fine loop. First-order tetrahedral elements are used in the inner coarse loop and second-order tetrahedral elements are used in the inner fine loop, respectively. The most computational costly kernel is the Element-by-Element kernel which computes sparse matrix-vector products. The Element-by-Element kernel computes the product of the element stiffness matrix and vectors element wise, and adds the results for all elements to compute a global matrix vector product. As element matrices are computed on the fly, the data transfer size from memory can be reduced significantly. This leads to circumventing the memory bandwidth bottleneck, and thus is suitable for recent architectures including GPUs, which have low memory bandwidth compared with its arithmetic capability. In summary, our base solver [1] computes much part of computation in single precision, reduces the amount of data transfer and computation, and avoids memory bound computation in sparse matrix-vector multiplication. They are desirable conditions for GPU computation to exhibit higher performance. On the other hand,

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

Compute $\mathbf{f}^1$ by split-node technique
Solve $\mathbf{K}\mathbf{u}^1 = \mathbf{f}^1$
$\{\boldsymbol{\sigma}^j\}_{j=1}^4 \Leftarrow \mathbf{D}\mathbf{B}\mathbf{u}^1$
$\{\delta\mathbf{u}_j\}_{j=1}^4 \Leftarrow \mathbf{0}$
$i \Leftarrow 2$
while $i \leq N_t$ **do**
  if $6 \leq i \leq 8$ **then**
    Compute initial guess solution by 2nd-order Adams-Bashforth method $\delta\mathbf{u}^{i+3} \Leftarrow \mathbf{u}^i - 3\mathbf{u}^{i+1} + 2\mathbf{u}^{i+2}$
  end
  if $i \geq 9$ **then**
    Compute initial guess solution by linear predictor $\delta\mathbf{u}^{i+3} \Leftarrow (-17\delta\mathbf{u}^{i-7} - 10\delta\mathbf{u}^{i-6} - 3\delta\mathbf{u}^{i-5} + 4\delta\mathbf{u}^{i-4} + 11\delta\mathbf{u}^{i-3} + 18\delta\mathbf{u}^{i-2} + 25\delta\mathbf{u}^{i-1})/28$
  end
  while $\|\mathbf{K}^v\delta\mathbf{u}^i - \mathbf{f}^i\| > \epsilon$ **do**
    $\{\mathbf{f}^j\}_{j=i}^{i+3} \Leftarrow \sum_k \int_{\Omega_e^k} \mathbf{B}^T(dt\mathbf{D}^v\{\boldsymbol{\beta}^j\}_{j=i}^{i+3} - \{\boldsymbol{\sigma}^j\}_{j=i}^{i+3})d\Omega_e + \mathbf{f}^0$
    Solve $\mathbf{K}^v\{\delta\mathbf{u}^j\}_{j=i}^{i+3} = \{\mathbf{f}^j\}_{j=i}^{i+3}$ using **Algorithm 2**
    $\{\boldsymbol{\sigma}^j\}_{j=i+1}^{i+3} \Leftarrow \{\boldsymbol{\sigma}^j\}_{j=i}^{i+2} + \mathbf{D}^v(\mathbf{B}\{\delta\mathbf{u}^j\}_{j=i}^{i+2} - dt\{\boldsymbol{\beta}^j\}_{j=i}^{i+2})$
  end
  $\mathbf{u}^i \Leftarrow \mathbf{u}^{i-1} + \delta\mathbf{u}^i$
  $\boldsymbol{\sigma}^{i+4} \Leftarrow \boldsymbol{\sigma}^{i+3} + \mathbf{D}^v(\mathbf{B}\delta\mathbf{u}^{i+3} - dt\boldsymbol{\beta}^{i+3})$
  $i \Leftarrow i + 1$
end

Algorithm 1: Coseismic/postseismic crustal deformation computation against given fault displacement. $(\ )^n$ is the variables in the nth timestep. dt is time increment and $\boldsymbol{\beta}^n = \mathbf{D}^{-1}\mathbf{A}\boldsymbol{\sigma}^n$, where $\boldsymbol{\sigma}^n = (\sigma_{11}^n, \sigma_{22}^n, \sigma_{33}^n, \sigma_{12}^n, \sigma_{23}^n, \sigma_{13}^n)^{\mathrm{T}}$. $\mathbf{B}$ is the displacement-strain transformation matrix and $\mathbf{D}$ and $\mathbf{A}$ are 6×6 matrices indicating material properties. $\mathbf{D}^v = (\mathbf{D}^{-1} + \alpha dt\boldsymbol{\beta}')$, where α is a controlling parameter and $\boldsymbol{\beta}'$ is the Jacobian matrix of $\boldsymbol{\beta}$.

the key kernel in the solver, Element-by-Element kernel, requires many random data accesses when adding up element wise results. This data access becomes the bottleneck in the solver. In this paper, we aim to improve the performance of the Element-by-Element kernel. We add two techniques described in following subsections, into our baseline solver.

2.1 Parallel computation of multiple time steps

In the developed method, we solve four time steps in the analysis in parallel. [6] describes its approach to obtain the accurate predictor using multiple time steps for linear wave propagation simulation. This paper extends the algorithm to viscoelastic analyses. As the stress of the step before needs to be obtained before solving the next step, only one time step can be solved exactly. In Algorithm 1, we focus on solving the equation on i-th timestep. Here we compute until the error

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

(a) Outer loop

1 $\mathbf{r} \Leftarrow \sum \mathbf{K}_e \mathbf{u}_e$
2 $\mathbf{r} \Leftarrow \mathbf{f} - \mathbf{r}$
3 $\beta \Leftarrow 0$
4 $\overline{\mathbf{u}} \Leftarrow \overline{\mathbf{M}}^{-1}\mathbf{r}$
5 $\overline{\mathbf{r}}_c \Leftarrow \overline{\mathbf{P}}^T\mathbf{r}$
6 $\overline{\mathbf{u}}_c \Leftarrow \overline{\mathbf{P}}^T\overline{\mathbf{u}}$
7 Solve $\overline{\mathbf{u}}_c = \overline{\mathbf{K}}_c^{-1}\overline{\mathbf{r}}_c$ in **(b)** with $\overline{\epsilon}_c^{in}$ and N_c
8 $\overline{\mathbf{u}} \Leftarrow \overline{\mathbf{P}}\overline{\mathbf{u}}_c$
9 Solve $\overline{\mathbf{u}} = \overline{\mathbf{K}}^{-1}\overline{\mathbf{r}}$ in **(b)** with $\overline{\epsilon}^{in}$ and N
10 $\mathbf{u} \Leftarrow \overline{\mathbf{u}}$
11 $\mathbf{p} \Leftarrow \mathbf{z} + \beta\mathbf{p}$
12 $\mathbf{q} \Leftarrow \sum \mathbf{K}_e \mathbf{p}_e$
13 $\rho \Leftarrow (\mathbf{z}, \mathbf{r})$
14 $\gamma \Leftarrow (\mathbf{p}, \mathbf{q})$
15 $\alpha \Leftarrow \rho/\gamma$
16 $\mathbf{r} \Leftarrow \mathbf{r} - \alpha\mathbf{q}$
17 $\mathbf{u} \Leftarrow \mathbf{u} + \alpha\mathbf{p}$

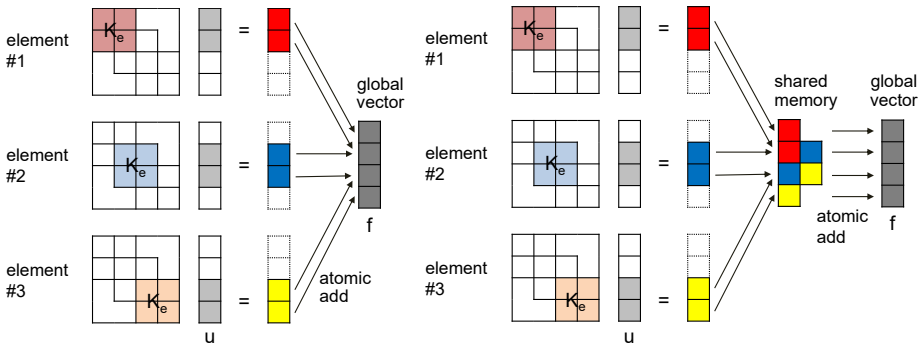
(b) Inner loop

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

Algorithm 2: The iterative solver to obtain a solution $\mathbf{u}$. $(\ )_c$ are variables in first-order tetrahedral model, while others are in second-order tetrahedral model. $(\ ^-)$ represents single-precision variables, while the others are double-precision variables. The input variables are $\mathbf{K}, \overline{\mathbf{K}}, \overline{\mathbf{K}}_c, \overline{\mathbf{P}}, \mathbf{u}, \mathbf{f}, \overline{\epsilon}_c^{in}, N_c, \overline{\epsilon}^{in}$, and N. The other variables are temporal. $\overline{\mathbf{P}}$ is a mapping matrix from the coarse model to the target model. This algorithm computes four vectors at the same time, so coefficients have the size of four and vectors have the size of $4 \times$ DOF. All computation steps in this solver, except MPI synchronization and coefficient computation, are performed in GPUs.

of the i-th time step (displacement) becomes smaller than prescribed threshold ϵ as described in lines 13 to 17 of Algorithm 1. The next three time steps, $i+1$, $i+2$, and $i+3$-th time steps, are solved using the solutions of the steps before to estimate the solution. The estimated solution of the step before is used to update the stress state and outer force vector, which is corresponding to lines 18 and 19 in Algorithm 1. By using this method, we can obtain estimated solutions for improving the convergency of the solver. In this method, four vectors for i, $i+1$, $i+2$, and $i+3$-th time steps can be computed simultaneously. In the Element-by-Element kernel, the matrix is read only once for four vectors; thus we can improve the computation efficiency. In addition, four values corresponding to the four time steps will be consecutive in memory address space. Therefore we can reduce random memory accesses and computation time compared to conducting the Element-by-Element kernel of one vector for four times. That is, the arithmetic count per iteration increases by approximately four times, but

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3



(a) Original scheme. Element-wise results are added to the global vector directly using atomic operations.

(b) Proposed scheme. Element-wise results are firstly summarized using shared memory, and then nodal values are added to the global vector.

Fig. 1: Rough scheme for reduction in Element-by-Element kernel to compute $\mathbf{f} \Leftarrow \sum \mathbf{K}_e \mathbf{u}_e$.

the decrease in the number of iterations and the improvement of computational efficiency of the Element-by-Element kernel are expected to reduce the time-to-solution.

In order to improve convergency, it is important to estimate the initial solution of the fourth time step accurately. We can use a typical predictor such as the Adams-Bashforth method, however we developed more accurate predictor considering that solutions for viscoelastic analysis smoothly change in each time step, as described in lines 7 to 12 in Algorithm 1. For predicting the 9th step and on, we use a linear predictor. In this linear predictor, a linear regression based on the accurately computed 7 time steps are used to predict the future time step. As regressions based on higher order polynomials or exponential base functions may lead to jumps in the prediction, we will not use them in this study.

2.2 Reduction of atomic access

The algorithm introduced in previous subsection is assumed to circumvent the bottleneck of the performance of Element-by-Element kernel. On the other hand, implementation in the previous study [7] requires to add up element wise results directly to the global vector using atomic function, as shown in Fig. 1a. Considering that each node can be shared by multiple elements, performance may decrease due to the race condition; thereby we need to modify its algorithm to improve the efficiency of the Element-by-Element kernel. We use a buffering method to reduce the number of accesses to the global vector. Regarding NVIDIA GPU, we can utilize a shared memory, in which values can be referred among threads in the same `Block`. The computation procedure is as below and also described in Fig. 1b.

1. Group elements in to blocks, and store element wise results into a shared memory

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

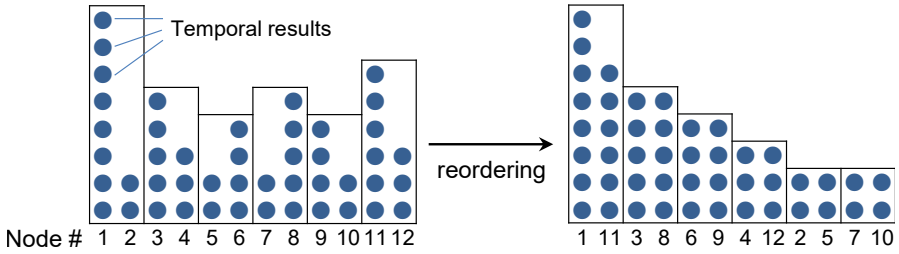


Fig. 2: Reordering of reduction table. Temporal results are aligned in corresponding node number. In this figure, we assume there are two threads per warp and 12 nodes in the thread block for simplicity. Load balance in warp is improved by reordering.

2. Add up nodal values in shared memory using a precomputed table
3. Add up nodal values to global vector

We can expect the performance improvement as the number of atomic operations to the global vector can be reduced and summation of temporal results is mainly performed in preliminary reduction in a shared memory, which has wider bandwidth. In this scheme, the setting of block size is assumed to have some impact on its performance. By allocating more elements in a `Block`, we can improve the number of reduction of nodal values in shared memory. However, the total number of threads is constrained by the shared memory size. In addition, we need to synchronize threads in a `Block` when switching from element wise matrix-vector multiplication to data addition part, using large number of threads in a `Block` leads to an increase in synchronization cost. Under these circumstances, we allocate 128 threads (32 elements $\times$ four time steps) per `Block`.

In GPU computation, SIMT composing of 32 threads is used [12]. When the number of computation differs between the 32 threads, it is expected to lead to decrease in performance. In reduction phase, we need to assign threads per node. However, since the number of connected elements differs significantly between nodes, we can expect large load imbalance among the 32 threads. Thus we sort the nodes according to the number of elements to be added up as described in Fig. 2. This leads to good load balance among the 32 threads, leading to higher computational efficiency.

This method on shared memory requires implementation by CUDA. We also use CUDA for inner product computation to improve the memory access pattern and thus improve efficiency. On the other hand, other computations such as vector addition and subtraction are very simple computation; thus each thread uses almost the same number of registers whether we use CUDA or OpenACC. Also it is not necessary to use functions specialized for NVIDIA GPUs such as shared memory or warp function. For these reasons, the computations result in memory bandwidth bound and there is little difference between implementation by CUDA and by OpenACC. Thus we use CUDA for these performance sensitive kernels, and use OpenACC for the other parts. The CUDA part is called via a wrapper function.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

Table 1: Configuration of Element-by-Element kernels for performance comparison

Case	# of vectors	Reduction using shared memory	Reordering of nodes in reduction
A	1	x	-
B	4	x	-
C	4	o	x
D	4	o	o

3 Performance Measurement

We measure performance of the developed method on hybrid nodes of Piz Daint[1].

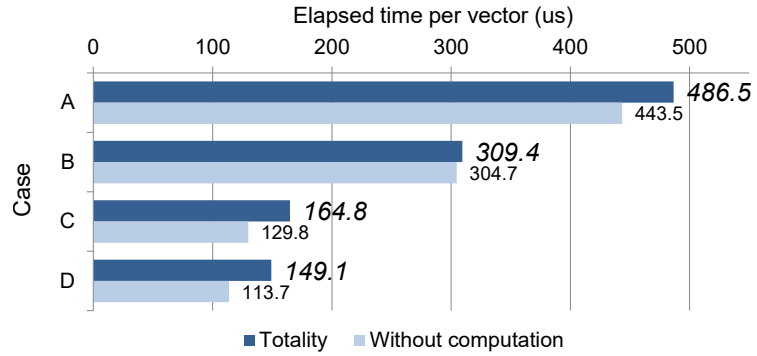
3.1 Performance measurement of the Element-by-Element kernel

We use one P100 GPU on Piz Daint to measure performance of the Element-by-Element kernels. The target finite-element problem consists of 959,128 tetrahedral elements, with 4,004,319 degrees-of-freedom in second-order tetrahedral mesh and 522,639 degrees-of-freedom in first-order tetrahedral mesh. Here we compare four versions of the kernels summarized in Table 1. Case A corresponds to the conventional Element-by-Element kernel, and Case D corresponds to the proposed kernel.

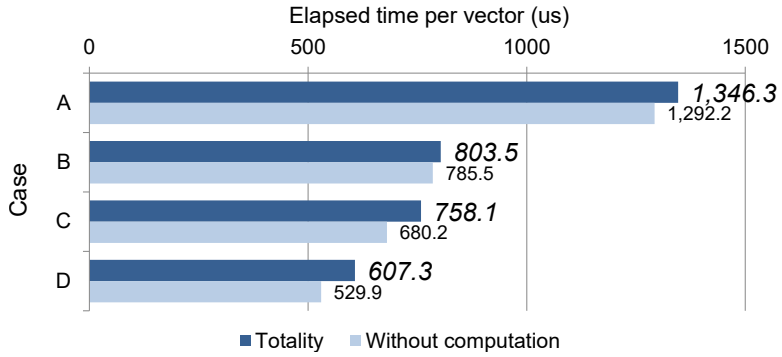
Figure 3 shows the normalized elapsed time per vector of the kernels in inner fine and coarse loops. We can see that the use of four vectors, reduction, and reordering significantly improves performance. In order to assess the time spent for data access, we also indicate the time measured for the Element-by-Element kernel without computing the element wise matrix-vector products. We can see that the data access is dominant in the Element-by-Element kernel on P100 GPUs, and that the elapsed time of the kernel has decreased with the decrease in memory access by reduction. When compared to the performance in second-order tetrahedral mesh, the performance in first-order tetrahedral mesh was further improved by reduction using shared memory. This effect can be confirmed by the number of call for atomic add to the global vector: In second-order tetrahedral mesh, atomic addition is performed 115,095,360 times in Case B and 43,189,848 times in Case D; thereby the number of calls is reduced by about 37%. For the first-order tetrahedral mesh, atomic addition is performed 46,038,144 times in Case B and 10,786,920 times in Case D; thus the number of calls is reduced by about 23%. In total, we can see that the computational performance of the developed kernel (Case D) has improved by 3.3 times in first-order tetrahedral mesh and 2.2 times in second-order tetrahedral mesh when comparing with the conventional kernel (Case A).

[1] Piz Daint comprises of 1,431 × multicore compute node (Two Intel Xeon E5-2695 v4) and 5,320 × hybrid compute node (Intel Xeon E5-2690 v3 + NVIDIA Tesla P100) connected by Cray Aries routing and communications ASIC, and Dragonfly network topology.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3



(a) First-order tetrahedral mesh



(b) Second-order tetrahedral mesh

Fig. 3: Elapsed time per Element-by-Element kernel call. Elapsed times are divided by four when using four vectors.

3.2 Comparison of solver performance

We compare the developed solver with the previous viscoelastic solver in [10] using GPUs in Piz Daint. This solver is originally designed for CPU-based supercomputers and we port this to GPU computation environment and for performance measurement. The solver uses CRS-based matrix-vector products, however, we modify this to Element-by-Element method, because it would be more clear to confirm the effects of our proposed method. The same tolerances of solvers is used for both methods, $\epsilon = 10^{-8}$ is used for the outer loop, $(\bar{\epsilon}_c^{in}, N_c) = (0.1, 300)$ is used for the inner coarse loop, and $(\bar{\epsilon}^{in}, N) = (0.2, 30)$ is used for the inner fine loop. These tolerance numbers are selected to minimize the elapsed time for both solvers. We use time step increment $dt = 2592000$ s with $N_t = 300$ time steps, and measure performance of the viscoelastic computation part (time step 2 to 300).

A model with 41,725,739 degrees-of-freedom and 30,720,000 second-order tetrahedral elements is computed using 32 Piz Daint nodes. Figure 4 shows the number of iterations and elapsed time of the solvers. By using the multistep

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

predictor, the number of iterations of the most computationally costly inner coarse loop has decreased by 2.3 times. In addition, Element-by-Element kernel performance is improved as measured in the previous subsection. These two modifications to the solver have decreased the total elapsed time by 2.79 times.

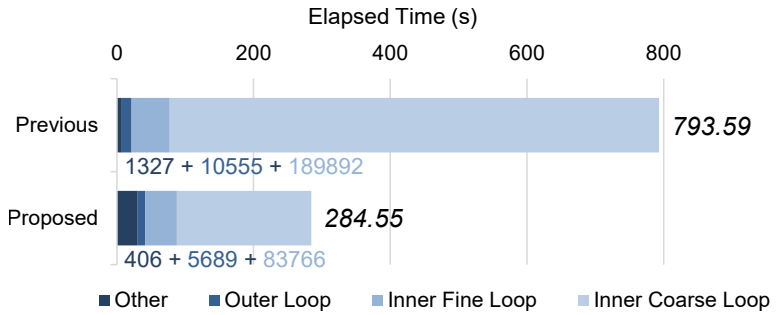


Fig. 4: Performance comparison of the entire solver. The numbers of iteration for outer loop, inner fine loop, and inner coarse loop are described below each bar.

4 Application Example

We apply the developed solver to a viscoelastic deformation problem following a hypothetical earthquake on the Hellenic arc subduction interface, which affects deformation measured in Greece and across the Eastern Mediterranean. We selected this Hellenic region, because recent analysis of time-scale bridging numerical models suggests that the large amount of sediments subducting could mean that a larger than anticipated M 9 earthquake might be able to occur in this highly populated region [3]. To model the complete viscoelastic response of the system we simulate a large depth range, including the Earth's crust, lithosphere and complete mantle down to the core boundary. The target domain is of size 3,686 km $\times$ 3,686 km $\times$ 2,857 km. Geometry data of layered structure is given in spatial resolution of 1 km [2].

To fully reflect the geometry data into the analysis model, we set resolution of finite-element model to 0.9 km (second-order tetrahedral element size is 1.8 km). As this becomes a large scale problem, we use a parallel mesh generator capable of robust meshing of large complex shaped multiple material problems [5, 6]. This leads to a finite-element model of 589,422,093 second-order tetrahedral elements, 801,187,352 nodes, and 2,403,562,056 degrees-of-freedom shown in Figure 5a-d. We can see that the layered structure geometry is reflected into the model. We input a hypothetical fault slip in the direction of the subduction, that is, slip with (dx, dy, dz) = (25, 25, -10) m, at the subduction interface separating the continental crust of Africa and Europe in the center of the model with diameter of 250 km. Following this hypothetical M 9 earthquake we compute the elastic coseismic surface deformation and postseismic viscoelastic deformation due to viscoelastic relaxation of the crust, lithosphere and mantle.. Following [10], a split

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

node method is used to input the fault dislocation, and time step increment dt is set to 30 days (2,592,000 s). The analysis of 2,000 time steps took 4587 s using 512 P100 GPUs on Piz Daint.

Figure 5e,f shows the surface deformation snapshots. We can see that elastic coseismic response as well as the viscoelastic response is computed reflecting the 3D geometry and heterogeneity of crust. We can expect more realistic response distribution by inputting fault slip distributions following current solid earth science knowledge.

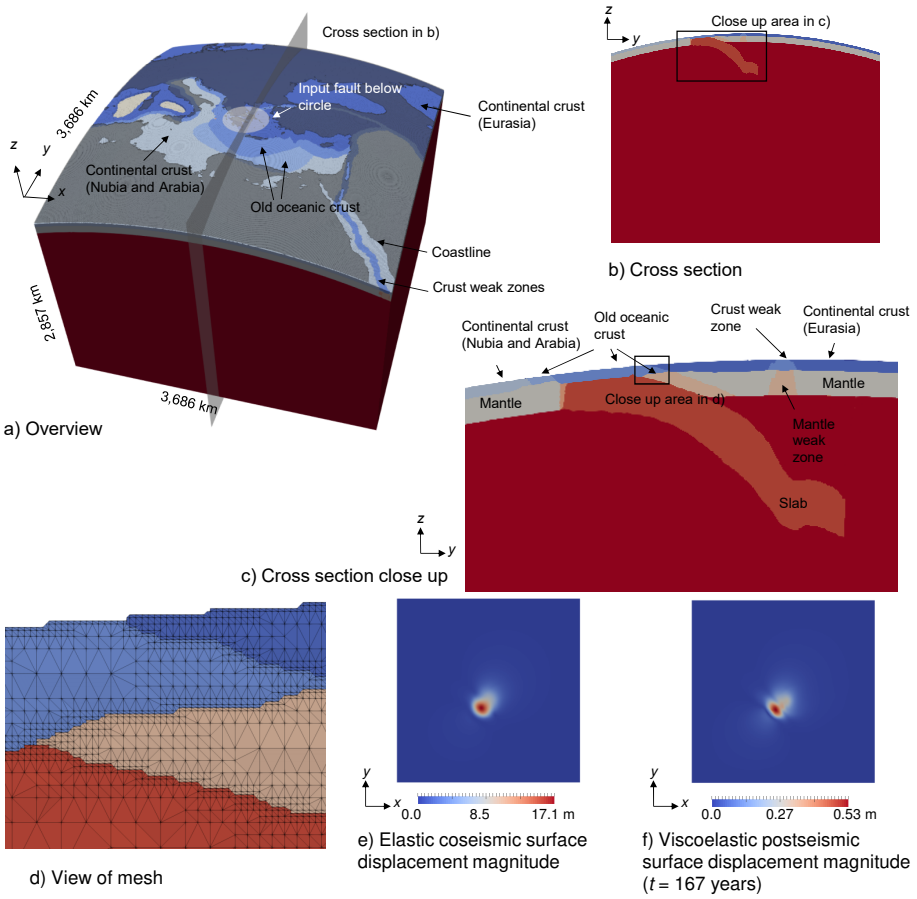


Fig. 5: Finite-element mesh for application problem. The 10 layered crust is modeled using 0.9 km resolution mesh. Elastic coseismic and viscoelastic postseismic displacements. a) Overview of finite-element mesh with position of input fault and position of cross section. b) Cross section of finite element mesh. c) Close up area in the cross section. d) Close up view of mesh. e) Elastic coseismic response and f) viscoelastic postseismic response.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

5 Conclusion

We developed a fast unstructured finite-element solver for viscoelastic crust deformation analysis targeting GPU-based computers. The target problem becomes very computationally costly since it requires solving a problem with more than 10^9 degrees-of-freedom. In this analysis, the random data access in Element-by-Element method in matrix-vector products was the bottleneck. To eliminate this bottleneck, we proposed two methods: one is a reduction method to use shared memory of GPUs, and the other one is a multi-step predictor and linear predictor to improve the convergency of the solver. Performance measurement on Piz Daint showed 2.79 times speedup from the previous solver. By the acceleration of viscoelastic analysis by the developed solver, we expect applications to inverse analysis of crust properties or many case analysis.

References

1. Ryoichiro Agata, Tsuyoshi Ichimura, Kazuro Hirahara, Mamoru Hyodo, Takane Hori, and Muneo Hori. Robust and portable capacity computing method for many finite element analyses of a high-fidelity crustal structure model aimed for coseismic slip estimation. *Computers & Geosciences*, 94:121–130, 2016.
2. Peter Bird. An updated digital model of plate boundaries. *Geochemistry, Geophysics, Geosystems*, 4(3):n/a–n/a, 2003. 1027.
3. S. Brizzi, Iris van Zelst, Ylona van Dinther, Francesca Funiciello, and Fabio Corbi. How long-term dynamics of sediment subduction controls short-term dynamics of seismicity. In *American Geophysical Union*, 2017.
4. Piz Daint. `https://www.cscs.ch/computers/piz-daint/`.
5. Kohei Fujita, Keisuke Katsushima, Tsuyoshi Ichimura, Muneo Hori, and Lalith Maddegedara. Octree-based multiple-material parallel unstructured mesh generation method for seismic response analysis of soil-structure systems. *Procedia Computer Science*, 80:1624 – 1634, 2016. International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.
6. Kohei Fujita, Keisuke Katsushima, Tsuyoshi Ichimura, Masashi Horikoshi, Kengo Nakajima, Muneo Hori, and Lalith Maddegedara. Wave propagation simulation of complex multi-material problems with fast low-order unstructured finite-element meshing and analysis. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, pages 24–35, New York, NY, USA, 2018. ACM.
7. Kohei Fujita, Takuma Yamaguchi, Tsuyoshi Ichimura, Muneo Hori, and Lalith Maddegedara. Acceleration of element-by-element kernel in unstructured implicit low-order finite-element earthquake simulation using openacc on pascal gpus. In *Proceedings of the Third International Workshop on Accelerator Programming Using Directives*, pages 1–12. IEEE Press, 2016.
8. Yukitoshi Fukahata and Mitsuhiro Matsu'ura. 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. Takane Hori, Mamoru Hyodo, Shin'ichi Miyazaki, and Yoshiyuki Kaneda. Numerical forecasting of the time interval between successive m8 earthquakes along the nankai trough, southwest japan, using ocean bottom cable network data. *Marine Geophysical Research*, 35(3):285–294, 2014.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3

10. Tsuyoshi Ichimura, Ryoichiro Agata, Takane Hori, Kazuro Hirahara, Chihiro Hashimoto, Muneo Hori, and Yukitoshi Fukahata. An elastic/viscoelastic finite element analysis method for crustal deformation using a 3-d island-scale high-fidelity model. *Geophysical Journal International*, 206(1):114–129, 2016.
11. Timothy Masterlark. Finite element model predictions of static deformation from dislocation sources in a subduction zone: sensitivities to homogeneous, isotropic, poisson-solid, and half-space assumptions. *Journal of Geophysical Research: Solid Earth*, 108(B11), 2003.
12. John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.
13. James M Winget and Thomas JR Hughes. Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies. *Computer Methods in Applied Mechanics and Engineering*, 52(1-3):711–815, 1985.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_3