Parallel Performance of Tectonic Loading Process Model at Transcurrent Plate Boundaries

Kengo Nakajima[(1)], Chihiro Hashimotoand[(2)] and

Mitsuhiro Matsu'ura[(2)]

(1) Research Organization for Information Science and Technology (RIST), Tokyo, Japan (e-mail: nakajima@tokyo.rist.or.jp). (2) Japan Marine Science and Technology Center, Yokosuka, Japan (e-mail: hashi@jamstec.go.jp) (3) The University of Tokyo, Tokyo, Japan (e-mail: hashi@solid.eps.s.u-tokyo.ac.jp, matsuura@eps.s.u-tokyo.ac.jp).

Abstract

Code for simulation of tectonic loading process at transcurrent plate boundaries was *parallelized* by MPI. Matrix assembly and inversion processes have been largely optimized for parallel computing, and 2-way parallelization was developed for local operations for parameter points and data/integral points. Computations using this method on a Hitachi SR2201 with up to 128 processors demonstrated good scalability.

1. Introduction

Boundary integral methods or boundary element methods are widely used for the simulation of tectonic loading process at plate boundaries. In these types of methods, the assembly and inversion of large-scale dense matrices are required, and represent the most expensive processes in the entire computational process in terms of both computation time and memory usage. Therefore, the full utilization of parallel computing is critical to achieving higher resolution and to treating larger regions.

In this study, existing code[1] for the tectonic loading process simulation at transcurrent plate boundaries has been *parallelized* on a Hitachi SR2201 with up to 128 processors. In the following sections we outline the original code, describe the parallelization of the code and present results and conclusions. We adopted message passing style parallel programming model using MPI[2] mainly due to the high efficiency, portability and robustness.

2. Physical and Numerical Modeling

Details of the physical modeling of the original code are found in[1]. Only a brief outline of the physical modeling process is given here. In the lithosphere-asthenosphere system with plate boundaries, the physical process of stress accumulation at the plate boundaries

is essentially governed by a coupled nonlinear system of fault slip and shear stress equations with a constitutive relationship. In the method in[1], these coupled equations are linearized by the Levenberg-Marquardt least-squares method[3] . Large-scale dense coefficient matrices are provided, and linearized equation systems are solved directly by Gaussian elimination[4]. Such methods usually require N^2 order of memory storage and N^3 order of operations for N unknowns. Therefore, parallel computing is critical in terms of both CPU time and memory requirement.

3. Parallelization

3.1 Profiling

Before starting parallelization, the original code was analyzed using the profiling tool *pixie* on a UNIX system. According to the profiling results, 96.5% of the entire process is devoted to the following two processes :

Procedure	**seconds**	**%**
Matrix Assembly	375.4	77.2
Gaussian Elimination	93.7	19.3

This measurement was executed on a COMPAQ Alpha 21164 (500MHz) single CPU workstation with Digital UNIX. The original code was written in FORTRAN90 and compiled by the Digital FORTRAN compiler. In the parallelization process, the main effort was devoted to these 2 parts; we focus on the parallelization of matrix assembly in this paper.

3.2 Matrix Assembly

The original FORTRAN source code for matrix assembly is shown in Fig. 1 (i). There are 3 loops in the code; the inner 2 loops (both for *1~m*a) compute the coefficient matrix *A(i,j)* and the right-hand side vector *B(j*). Unknowns *X* are spline coefficients and the array size is *m*a. Each coefficient is defined on a *parameter point* [1]. The outermost loop (for *1~na*da) is for the effect of *data points* or *integral point*s. The number of data/integral points is *ndat*a, and *nada* is usually several times larger than *m*a. According to the FORTRAN source code in Fig. 2 (i), the matrix formation process is perfectly independent for each parameter point and data/integral point. This means that this operation can be performed in very localized manner suitable for parallel computing. Therefore, the parallelization of this process is rather straightforward.

3.3 Two-Way Parallelization

Fig. 1 (ii) shows the *parallelized* source code for the equivalent part written in FORTRAN and MPI. The original code has been parallelized with only very small changes apart from the addition of subsidiary parameters/arrays and MPI messages.

Each processor stores a coefficient matrix in *distributed* manner as *A (ma,maP*), where *maP=ma* / P; *P = Processor numbe*r. Each parameter point is renumbered in a cyclic manner in order to avoid load imbalance during LU factorization in the parallel processing [4][5]. The inner 2 loops are almost identical to the original code. As mentioned above, the

outermost loop corresponds to the operation on each data/integral point and the number of data points *ndata* is larger than that of the parameter point *m*a. Therefore, operation and storage for data/operation points should also be localized in order to save memory and computation time. In Fig. 2 (ii), the outer most loop is from 1 to *ndataP* (= *ndata*/P) and the operation is as follows:

(1) The effect of each data/integral point is calculated locally
(2) On the ***call mpi_allgather*** (underlined), the value of *sig2ima*t, *dymat* and *dydamat* calculated at each integral point are delivered to each processor
(3) Parallel processes are synchronized at this MPI message and proceed to the inner 2 loops for the local matrix

Before and after the MPI message, an entirely different *space* is referred for parallel processing. This type of parallization method is very different from typical finite-element type local operations such as that in [6]. In this study, we call this type of processing *2-way parallelizatio*n. If there are *N* kinds of different parameters, *N-way parallelization* is required.

```
do i= 1, ndata
   call FUNCS
   sig2i= 1.d0/(sig(i)**2); dy= y(i) - ymod
   do j= 1, ma
      wt= dyda(j)*sig2i
      do k= 1, ma
         A(j,k) = A(j,k) + wt*dyda(k)
      enddo
      B(j)= B(j) + dy*wt
   enddo
enddo
```

(i) Original Code

```
do i0= 1, ndataP
   sig2imat= 0.d0; dymat= 0.d0; dydamat= 0.d0
   i= gfMTBL(i0)
      call FUNCS
      sig2imat(1)= 1.d0/(sig(i)**2)
      dymat(1)= y(i) - ymod
      do j= 1, ma
         dydamat(j)= dyda(j)
      enddo
   call MPI ALLGATHER (sig2imat, dymat, dydamat…)
   do ip= 1, PETOT
      is= (ip-1)*ma
      do j= 1, ma
         wt= dydamat(is+j)*sig2imat(ip)
         do k= 1, maP
            k1= gMTBL(k)
            gA(j,k)= gA(j,k) + wt*dydamat(is+k1)
         enddo
         gB(j)= gB(j) + dymat(ip)*wt
      enddo
   enddo
enddo
```

(ii) Parallel Code using MPI

Fig. 1 Parallelization of the Matrix Assembly Part

4. Examples

Parallel performance was measured for various fault lengths (L) from L = 150 km to L = 600 km where relative plate velocity was set to be 5 cm/year. Depth of the fault (D) is fixed to 45 km. Figure 2 shows history of shear stress accumulation on fault surface in L=300 km case.

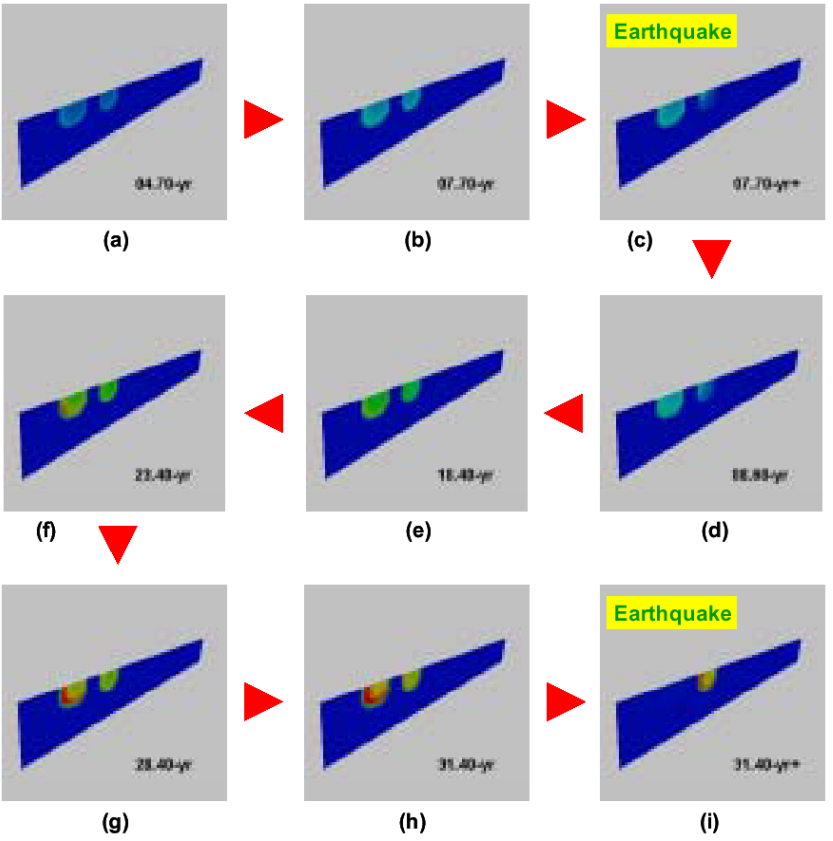

Fig.2 Result of Simulation : History of Shear Stress Accumulation (L=300 km, D= 45 km), Shear Stress : 0~2.50 Mpa, ma=1455, ndata=13846. (a)4.70 year, (b) 7.70 year (before earthquake), (c) 7.70 year (after earthquake), (d) 8.60 year, (e) 18.4 year, (f) 23.4 year, (g) 28.4 year, (h) 31.4 year(before earthquake) and (i) 31.4 year (after earthquake).

In this study, mesh size for the parameter points was 3 km × 3 km and that for the data / integral points was set to be 1 km × 1 km. Therefore, number of data/integral points (ndata) is 9 times as large as that of parameter points (ma) (Table.1). Both *ndata* and *ma* are proportional to the fault length L. Order of the total memory requirement is ma^2 and order of the computation cost is ndata × ma^2 as shown in Table.1. If the fault length is extended from 150 km to 600 km, problem size is about 4.2 times as large as the original case. Ratio of memory requirement is 17.5 and ratio of total computation cost is 70.0.

Table 1. Computational Cost according to Problem Size

Fault Length L(km)	Parameter # (ratio to L=150km case) ma	Data Point # ndata~L×D	Ratio of Total Computation Cost ndata×ma×ma
150	705(1.000)	6946	1.000
210	1005(1.426)	9706	2.840
300	1455(2.064)	13846	8.491
372	1815(2.574)	17158	16.372
450	2205(3.128)	20476	29.217
531	2610(3.702)	24472	48.288
600	2955(4.191)	27646	69.925

Figure 3 shows the parallel performance for various problem settings (L=150 km ~ 600 km and PE number = 1~128) on a Hitachi SR2201. One cycle of matrix assembly and inversion process of the nonlinear computation was extracted for measurement of performance. Performance was calculated according to elapsed computation time including communications and overheads. The results indicate perfect parallel performance and superlinear improvement in computation speed when many processors are applied.

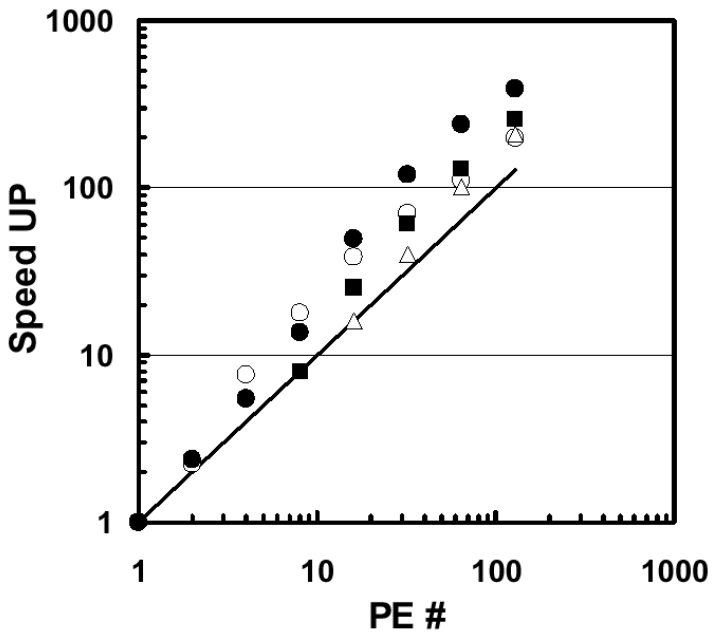

Fig.3 Parallel Performance of Tectonic Loading Process Model at Transcurrent Plate Boundaries on a Hitacih SR2201 up to 128 PEs for Various Problem Sizes. (Black Circles: L=150 km, White Circles : L=300 km, Black Sqares : L=450 km (PE# : 8~128), White

Triangles : L=600 km (PE# : 16~128)). Calculated according to Elapsed Time including Communications and Overheads.

5. Conclusions and Further Study

Code for simulation of tectonic loading process at transcurrent plate boundaries was parallelized by MPI. Matrix assembly and inversion processes have been largely optimized for parallel computing, and 2-way parallelization was developed for local operations for parameter points and data/integral points. Computations using this method on a Hitachi SR2201 with up to 128 processors demonstrated good scalability. The following areas are being considered for further development :

- Larger scale problems for real plate boundaries including subduction plate models around Japan Islands.
- Implementation of *iterative* solvers with robust preconditioning
- Optimization of the single CPU performance and vectorization for the *Earth Simulator*

Acknowledgments

This study is a part of the *Solid Earth Platform for Large Scale Computation* project funded by the Ministry of Education, Culture, Sports, Science and Technology, Japan through *Special Promoting Funds of Science & Technolog*y. The author would like to thank Professor Yasumasa Kanada (Information Technology Center, The University of Tokyo) for helpful discussions on high performance computing.

References

[1] Hashimoto, C. and Matsu'ura, M., 1999, *Physical Modeling of Tectonic Loading Processes at Transcurrent Plate Boundarie*s, 1st ACES Workshop Proceedings, 183-186.

[2] Gropp, W., Lusk, E. and Skjellum, A., 1994, *Using MPI : Portable Parallel Programming with the Message-Passage Interfac*e, MIT Press.

[3] Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P., 1992, *Numerical Recipes in FORTRA*N, Cambridge University Press.

[4] Duff, I.S., Erisman, A.M. and Reid, J.K., 1986, *Direct Methods for Sparse Matrice*s, Oxford Science Publications.

[5] LINPACK Web Site http://www.netlib.org/linpack/index.html

[6] GeoFEM Web Site http://geofem.tokyo.rist.or.jp/