Performance of FDM Simulation of Seismic Wave Propagation using the ppOpen-APPL/FDM Library on the Intel Xeon Phi Coprocessor

Futoshi Mori[1, 2], Masaharu Matsumoto[3], and Takashi Furumura[1,2]

[1] Interfaculty Initiative in Information Studies, The University of Tokyo, 1-1-1 Yayoi, Bunkyo-ku, Tokyo 1130032, JAPAN
[2] Earthquake Research Institute, The University of Tokyo, 1-1-1 Yayoi, Bunkyo-ku, Tokyo 1130032, JAPAN
[3] Information Technology Center, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa, Chiba, JAPAN

{f-mori, furumura}@eri.u-tokyo.ac.jp
matsumoto@cc.u-tokyo.ac.jp

Abstract. We evaluated the performance of a parallel 3D FDM simulation of seismic wave propagation using the Intel Xeon Phi coprocessor. We confirmed that MPI/OpenMP hybrid parallel computing with hyper-threading is more efficient than pure MPI parallelism. The performance of the thread parallel computing was further improved by fusing the original three DO loops of major kernel routines into two DO loops. The performance of the FDM simulation with two fused DO loops was 1.7-5.1 times faster than the original code with three DO loops however no performance acceleration was achieved for a fused single DO loop calculation. This is probably due to restrictions of the current Fortran compiler optimizations.

Keywords: Intel Xeon Phi, Pure MPI, MPI/OpenMP hybrid parallel computing, Seismic Wave Propagation, FDM.

1 Introduction

Recent trends in high-performance computing have focused on achieving high speed-up by parallel computing using multicore processing. Recently, the Intel Xeon Phi coprocessor, a new type of many core architecture coprocessors, has received significant attention with regard to multicore parallel computing. The Intel Xeon Phi installed in the Tianhe-2 computer at Sun Yat-sen University received the first prize in the Top 500 ranking of the world's fastest supercomputers in Nov. 2014 [1]. With such rapid changes in computing strategy, users should design a suitable code for the new parallel architecture each time a new machine is developed. To avoid such complication, we developed a standard parallel FDM library (ppOpen-APPL/FDM) as part of the ppOpen-HPC projects [2], which will enable users smooth and easy transport of current FDM applications to newly developed and future architectures.

Multicore clusters provide a programming environment for different types of parallel computing. Generally, the message passing interface (MPI) is considered optimal for process level coarse parallelism [3] and OpenMP is considered optimal for loop level fine grain parallelism [4]. Combining the MPI and OpenMP parallelization to construct a hybrid program may be more efficient for current multicore processing. MPI/OpenMP hybrid parallelization reduces the communication overhead of MPI but introduces OpenMP overhead because of thread creation and increased memory bandwidth contention. For multicore architecture, such as the T2K Open Supercomputer and Fujitsu FX10 system installed at the Information Technology Center, University of Tokyo, many researchers have demonstrated that MPI/OpenMP hybrid parallel computing is more effective than pure MPI parallel computing [5-8].

In this study, we evaluated the parallel performance of a 3D seismic wave calculation based on the ppOpen-APPL/FDM library, which has been implemented on the Intel Xeon Phi many core coprocessor. In Section 2, we present a brief explanation of the FDM simulation of seismic wave propagation using the ppOpen-APPL/FDM library. Section 3 presents an overview of the Intel Xeon Phi coprocessor. Sections 4 and 5 compare the performance of the parallel FDM simulation based on pure MPI and MPI/OpenMP hybrid parallel computing using the Intel Xeon Phi coprocessor. Conclusions and suggestions for future work are presented in Section 6.

2 Overview of the seismic wave propagation simulation using the ppOpen-APPL/FDM library

Here, we briefly explain the procedure of the FDM simulation of seismic wave propagation using the ppOpen-APPL/FDM library. This simulation explicitly solves an equation of motion in 3D as follows:

$$\dot{u}_p^{n+\frac{1}{2}} = \dot{u}_p^{n-\frac{1}{2}} + \frac{1}{\rho}\left(\frac{\partial \sigma_{xp}^n}{\partial x} + \frac{\partial \sigma_{yp}^n}{\partial y} + \frac{\partial \sigma_{zp}^n}{\partial z} + f_p^n \right)\Delta t,\ (p = x, y, z)\,, \quad (1)$$

where σ_{pq} , f_x , ρ , and Δt are stress, body force, density, and time step, respectively. $\dot{u}_p^{n+1/2}$ and $\dot{u}_p^{n-1/2}$ are the particle velocities at time t = (n±1/2)Δt.

Hereafter, we refer to this calculation procedure as the “update-velocity.” The stress component for the next time step at t = (n+1) Δ t is updated using the spatial derivative of the velocity component derived from Eq. (1) and multiplying the elastic constants (denoted as “update-stress”) as follows:

$$\sigma_{pq}^{n+1} = \sigma_{pq}^{n} + \left[\lambda \left(\frac{\partial \dot{u}_x^{n+\frac{1}{2}}}{\partial x} + \frac{\partial \dot{u}_y^{n+\frac{1}{2}}}{\partial y} + \frac{\partial \dot{u}_z^{n+\frac{1}{2}}}{\partial z} \right) \delta_{pq} + \mu \left(\frac{\partial \dot{u}_p^{n+\frac{1}{2}}}{\partial q} + \frac{\partial \dot{u}_q^{n+\frac{1}{2}}}{\partial p} \right) \right] \Delta t, \ (p,q = x,y,z), \quad \textbf{(2)}$$

where λ and μ are Lamé's constants and δ_{pq} denotes the Kronecker delta.

Space derivatives in Eq. 1 and Eq. 2 are calculated using a staggered-grid fourth-order-central FDM. Calculations of Eq. 1 and Eq. 2 are repeated explicitly for a desired time for seismic wave propagation. The structure of the program for the "update-velocity" kernel and "update-stress" kernels with three DO loops (i, j, k) are schematically illustrated in Fig.1. Note that these two calculation kernels are the most time consuming part of this FDM simulation. This is determined by measuring the calculation time for each part of the simulation using a Fujitsu FX10 [9] and an Intel Xeon Phi 5110P coprocessor (Fig.2).

(a) Update-stress

```
do K = NZ00, NZ01
    do J = NY00, NY01
      do I = NX00, NX01
              ⋮
        SXX(I,J,K)=SXX(I,J,K)&
               + (RLRM2*(D3V3)-RM2*(DZVZ1+DYVY1))*DT
        SYY(I,J,K)=SYY(I,J,K)&
               + (RLRM2*(D3V3)-RM2*(DXVX1+DZVZ1))*DT
        SZZ(I,J,K)=SZZ(I,J,K)&
               + (RLRM2*(D3V3)-RM2*(DXVX1+DYVY1))*DT
        SXY(I,J,K)=SXY(I,J,K)+RM1*DXVYDYVX1*DT
        SXZ(I,J,K)=SXZ(I,J,K)+RM1*DXVZDZVX1*DT
        SYZ(I,J,K)=SYZ(I,J,K)+RM1*DYVZDZVY1*DT
      end do
    end do
  end do
```

(b) Update-velocity

```
do K = NZ00, NZ01
    do J = NY00, NY01
      do I = NX00, NX01
              ⋮
  VX(I,J,K)=VX(I,J,K)&
       +(DXSXX(I,J,K)+DYSXY(I,J,K)+DZSXZ(I,J,K))*ROX*DT
  VY(I,J,K)=VY(I,J,K)&
        +(DXSXY(I,J,K)+DYSYY(I,J,K)+DZSYZ(I,J,K))*ROY*DT
  VZ(I,J,K)=VZ(I,J,K)&
         +(DXSXZ(I,J,K)+DYSYZ(I,J,K)+DZSZZ(I,J,K))*ROZ*DT
      end do
    end do
  end do
```

Fig. 1. Structure of "update-stress" (a) and "update-velocity" (b) kernels in the ppOpen-APPL/FDM

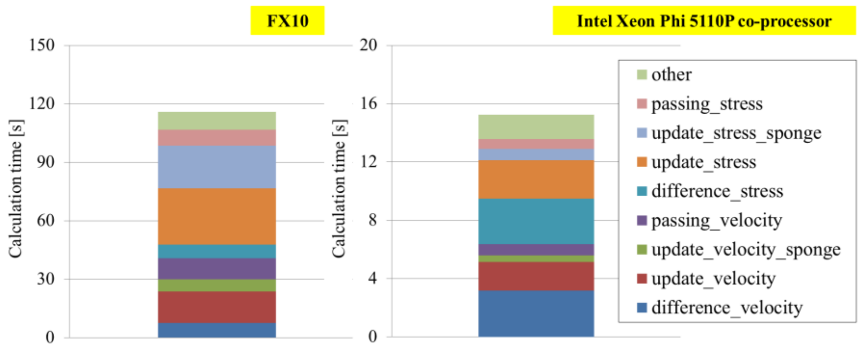

Fig. 2. Calculation cost of each kernel on the Fujitsu FX10 and Intel Xeon Phi 5110P coprocessors for 128^3 grid points

3 Overview of the Intel Xeon Phi

The Intel Xeon Phi 5110P coprocessor has 60 physical cores at a clock speed of 1.053 GHz. Each core offers four hyper-thread computations at hardware level. Thus, it yields 240 logical cores for thread parallel computing with 8 GB shared memory. The processor is connected to a host computer (dual 8-core Intel Xeon E5-2670 processors at 2.60 GHz and 128 GB shared main memory) via the PCI Express bus. Table 1 summarizes specifications of the host PC and Intel Xeon Phi 5110P coprocessor.

We used the Intel Fortran Compiler (version 14.0.2) and Intel MPI Library (version 4.1.0) with appropriate compiler options (mpiifort –mmic –O3 –align array64byte) for pure MPI parallel computing. In addition, we assigned an OpenMP option (–openmp) when conducting MPI/OpenMP hybrid parallel computing.

Table 1. Host PC cluster and Intel Xeon Phi coprocessor specifications

	PC cluster	Intel Xeon Phi 5110P
CPU clocks	2.60 GHz	1.053 GHz
Number of Cores	8	60
Thread/core	2	4
Size of L2 cache	20 MB	30 MB
Size of Shared Memory	128 GB	8 GB
Peak Performance	332.8 GFLOPS	1.01 TFLOPS
Peak Memory Bandwidth	51.2 GB/sec	320 GB/sec

We evaluated the performance of the parallel FDM simulation using the native computing mode of the Intel Xeon Phi. We compared the performance of pure MPI and MPI/OpenMP hybrid parallelization with different MPI processes (P) and number of OpenMP thread processes (T). The calculation time for each kernel was evaluated using 200 time-step calculations.

The 3D FDM simulation with 256 × 96 × 100 grid points required 6.0 GB of computer memory for single precision arithmetic, which is approximately the maximum model that can fit the available memory of the Intel Phi coprocessor (8 GB).

4 Performance of pure MPI and MPI/OpenMP hybrid parallel computing

Fig.3 shows results of a strong scaling test of parallel computing for two kernels based on pure MPI parallel computing. The results show that the calculation time has been well scaled down linearly from 4.9–0.5 s when calculating the update-velocity kernel and from 6.2–0.6 s for the update-stress kernel using 8–240 MPI processes. However, the increase in performance is saturated with large MPI processes (greater than 60 for both cases).

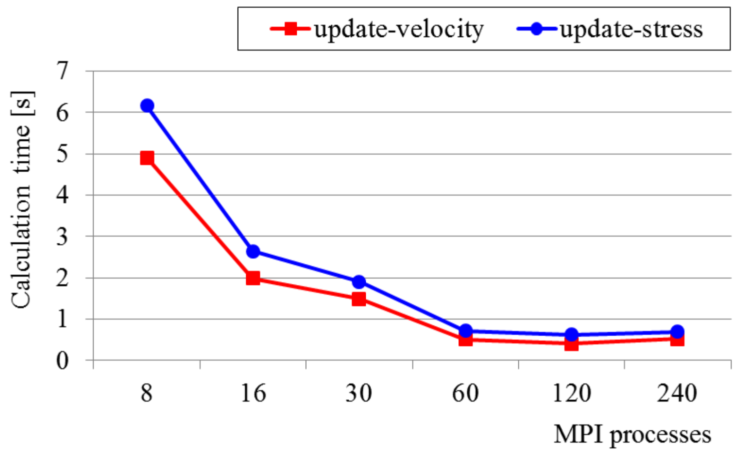

Fig. 3. Strong scaling performance test of pure MPI parallel computing for update-velocity kernel (squares) and update-stress kernel (circles). The horizontal axis is the number of MPI processes and vertical axis is calculation time.

We then conducted a strong scaling test with MPI/OpenMP hybrid parallel computing for two kernels (Fig.4). We inserted the OpenMP directives before the DO loops of the kernels and measured the computation time. For 16-core processing, the computation speed of the MPI/OpenMP hybrid parallel computing (P8T2; eight MPI processes and two OpenMP threads) is slower than that of the pure MPI parallel computing (P16T1). However, the speed of the MPI/OpenMP hybrid parallel computing becomes faster than that of the pure MPI parallel computing by utilizing large thread parallel computing with hyper-threading functions. For example, compared with pure MPI computing (P240T1), the speedup of the MPI/OpenMP hybrid computing (P60T4) was 2.71 times faster when calculating the update-velocity kernel and 2.16 times faster (P8T30) when calculating the update-stress kernel.

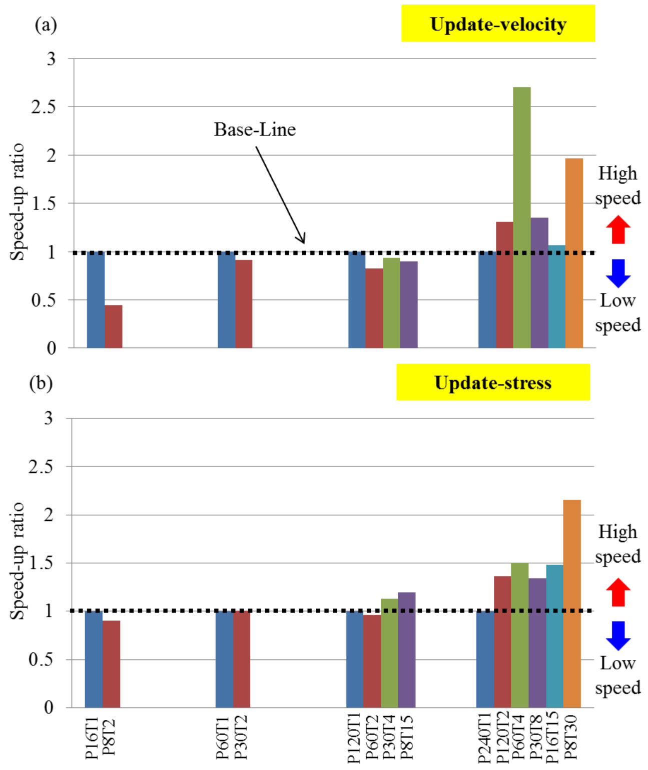

Fig. 4. Performance of strong scaling test of MPI/OpenMP hybrid parallel computing using 16, 60, 120, and 240 cores with different MPI processes (P) and number of OpenMP threads (T): speed-up ratio of update-velocity kernel (a) and update-stress kernel (b). In this graph, over the baseline mean a high-speed and below the baseline mean a low-speed when comparing with pure MPI parallel computing (P16T1, P60T1, P120T1, and P240T1) equals to baseline. The horizontal axis is each case and vertical axis is speed-up ratio.

5 Performance of kernel loop optimization

3D FDM simulation kernels consist of three DO loops with respect to i, j, and k directions. The fusion of many DO loops is expected to yield more efficient thread parallel computing performance because of the increased loop length for each thread computation while decreasing the overhead of the thread parallel computing. We examined the effectiveness of such loop fusion by combining three DO loops into two loops or one large loop.

Fig.5 shows the effect of loop fusion for MPI/OpenMP hybrid parallel computing by using 240-core hyper-thread parallel computing. For double loop fusion, calculation of two threads (P120T2) and four threads (P60T4) required significant computation time; however, larger thread (> 8) calculations showed much better performance than that of the original three DO loop calculations. Consequently, the calculation time of the double DO loop fusion was 1.7-5.1 times faster than that of the original three DO loops for the update-velocity and update-stress kernel calculations, respectively. However, the calculation time increased significantly for single loop fusion, which combined three DO loops into a large DO loop. This increase may be attributed to the fact that the prefetching and caching of the processor cores do not work effectively because of restrictions of the optimization scheme of the present Intel Fortran compiler.

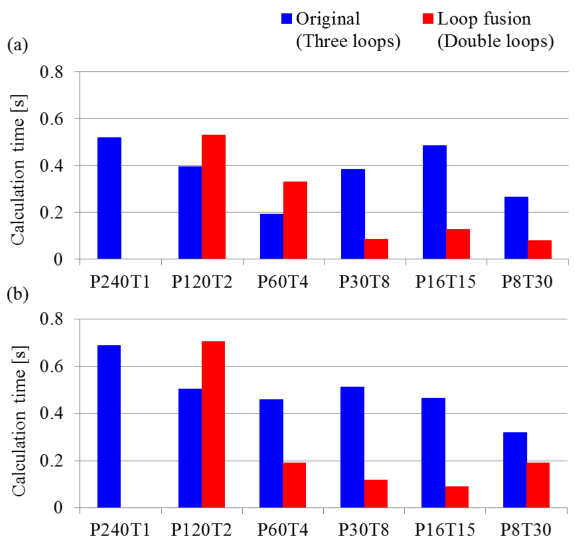

Fig. 5. Effect of loop fusion with MPI/OpenMP hybrid parallel computing using 240-core hyper-thread computation: (a) update-velocity kernel and (b) update-stress kernel. The blue bar indicates calculation time for the original three DO loop, and the red bar denotes the calculation time for double DO loop fusion. The horizontal axis is each case and vertical axis is calculation time.

6 Summary and Future work

We evaluated the parallel performance of the ppOpen-APPL/FDM for 3D FDM simulation of seismic wave propagation implemented on the Intel Xeon Phi 5110P coprocessor. We confirmed that MPI/OpenMP hybrid parallel computing is more effective than pure MPI parallel computing for many (120–240) logical cores with hyper-threading processing. To assess the performance of the thread parallel computing of OpenMP, loop fusion, which combined three DO loops to two DO loops, was very effective owing to the increased loop lengths on each core and decreased overhead of the thread parallel computing.

At present, the Intel Xeon Phi 5110P coprocessor has only 8 GB of local memory and is connected to other Phi coprocessors via the relatively slow PCI Express bus. This is the current bottleneck for performing realistic large-scale applications of the FDM simulation using the coprocessor. To overcome this, we need a comprehensive parallel computing scheme with concurrent use of Xeon coprocessors on the host computer, which is referred to as offload computing.

Acknowledgements

This work is supported by Core Research for Evolution Science and Technology (CREST), the Japan Science and Technology Agency (JST), Japan. We are grateful to Professor Tsuruoka at Earthquake Research Institute, The University of Tokyo for providing the Intel Xeon Phi coprocessor computing environment.

References

1. Top 500, http://www.top500.org
2. ppOpen-HPC project, http://ppopenhpc.cc.u-tokyo.ac.jp/wordpress/
3. MPI Web Site: http://www.mcs.anl.gov/research/projects/mpi/
4. OpenMP Web Site: http//www.openmp.org/
5. Noronha, R., and Panda, DK.: Improving Scalability of OpenMP Applications on Multi-core Systems Using Large Page Support. Parallel and Distributed Processing Symposium 2007 IPDPS 2007, pp.1-8 (2007).
6. Tsuji. M., and Sato, M.: Performance evaluation of OpenMP and MPI hybrid programs on a large scale multi-core multi-socket cluster: T2K Open Supercomputer. Parallel Processing Workshops 2009, pp.206-213 (2009).
7. Nakajima, K.: OpenMP/MPI Hybrid Parallel Multigrid Method on Fujitsu FX10 Supercomputer System. 2012 IEEE International Conference on Cluster Computing Workshops, pp. 199-206 (2012).
8. Furumura, T.: Large-scale parallel simulation of seismic wave propagation and strong ground motions for the past and future earthquakes in Japan. Journal of the Earth Simulator, vol.3, pp.29-38 (2005).
9. Mori, F., Furumura, T.: A generalization of the FDM calculation code for post peta-scale–a test of performance. Proc. International Workshop on Software for Peta-scale Numerical Simulation (SPNS2012), pp.222-235 (2012).