ppOpen-HPC:

Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT).

**ppOpen-APPL/FDM software**

ver. 0.3.1

Reference guide

License

This software is an open source free software application. Permission is granted to copy, distribute and/or modify this software and document under the terms of The MIT license. The license file is included in the software archive.

This software is one of the results of JST CREST ``ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT)” project.

Copyright (c) 2013-2015 T.Furumura

**Change History**

The change history lists the changes from version to version in the ppOpen-APPL/FDM source code. We update this section as we add new features. Note that we tend to update the reference guide at the same time we make changes to ppOpen-APPL/FDM.

**Changes in release 0.2.0**

Functionality added:

* Added the performance results using by ppOpen-APPL/FDM on FX10 for Flat MPI and Hybrid parallel
* Added new module: Visualization in 2D simulation using by AVS/Express

**Changes in release 0.3.0**

* Functionality added: inhomogeneous layered medium structure model
* Added the performance results using by ppOpen-APPL/FDM on Intel Xeon Phi coprocessor
* Added the performance evaluation of the program for decreasing the B/F ratio
* Added the ppOpen-APPPL/FDM with ppOpen-MATH/MP

**Changes in release 0.3.1**

* Added optimization code for NVIDIA GPU with OpenACC directive

Contents

1. Outline of seismic\_2D/seismic\_3D 5

1.1 Parallel simulation of seismic wave propagation in heterogeneous elastic media using ppOpen-APPL/FDM 5

1.2 Grid and coordinate system 5

1.3 Equations of Motion for 3D Seismic Waveﬁelds 6

1.4 Boundary conditions 6

1.4.1 Absorbing boundary 7

1.4.2 Free surface boundary 7

1.5 Anelastic attenuations 7

1.6 Spatial Differentiation 8

1.7 Input parameters 9

1.8 Requirements of time integration (CFL Condition) 9

1.9 Output and visualization of seismic waves 9

2. Parallel FDM simulation and performance 10

2.1 Domain partitioning and MPI 10

2.2 Parallel programming structure 11

2.3 Performance of parallel FDM simulation 11

2.4 Flat MPI model vs. thread/MPI Hybrid model 12

2.5 FlatMPI and Hybrid parallel using by ppOpen-APPL/FDM on FX10 13

2.6 Comparison of the strong scaling of MPI/OpenMP hybrid parallel computing and pure MPI parallel computing by ppOpen-APPL/FDM on Intel Xeon Phi 15

2.7 Performance tuning of kernel loop optimization of MPI/OpenMP hybrid parallel computing by ppOpen-APPL/FDM on Intel Xeon Phi 15

2.8 Performance evaluation of the program for decreasing the B/F ratio on Intel Xeon Phi coprocessor 16

2.8 Simulation using NVIDIA GPUs by using OpenACC 18

References 19

3. **Module/subroutine reference** 20

3.1 seism2d\_psv 21

3.1.1 module ppohFDM\_m\_absorb 23

3.1.2 module ppohFDM\_m\_comvar 24

3.1.3 module ppohFDM\_m\_kernel 25

3.1.4 module ppohFDM\_m\_medium 29

3.1.5 module ppohFDM\_m\_output 30

3.1.6 module ppohFDM\_m\_params 31

3.1.7 module ppohFDM\_m\_report 32

3.1.8 module ppohFDM\_m\_source 34

3.1.9 module ppohFDM\_m\_stdlib 37

3.1.10 module ppohFDM\_m\_surfbc 43

3.1.11 module ppohFDM\_m\_swatch 44

3.1.12 module ppohFDM\_m\_avs 46

3.2 seism3d3n 47

3.2.1 module ppohFDM\_boundary 58

3.2.2 module ppohFDM\_io 60

3.2.3 module ppohFDM\_param 63

3.2.4 module ppohFDM\_pfd3d 65

3.2.5 module ppohFDM\_ppsub 75

3.2.6 module ppohFDM\_set\_condition 78

3.2.7 module ppohFDM\_source 80

3.2.8 module ppohFDM\_sponge\_absorber 81

3.2.9 module ppohFDM\_stdio 82

3.2.10 module ppohFDM\_stress 83

3.2.11 module ppohFDM\_velocity 85

3.2.12 module ppohFDM\_set\_condition (real) 87

3.2.13 module ppohFDM\_param (real) 89

3.3 decrease B/F ratio (Seism3d3n) 91

3.3.1 module ppohFDM\_stress 100

3.3.2 module ppohFDM\_velocity 102

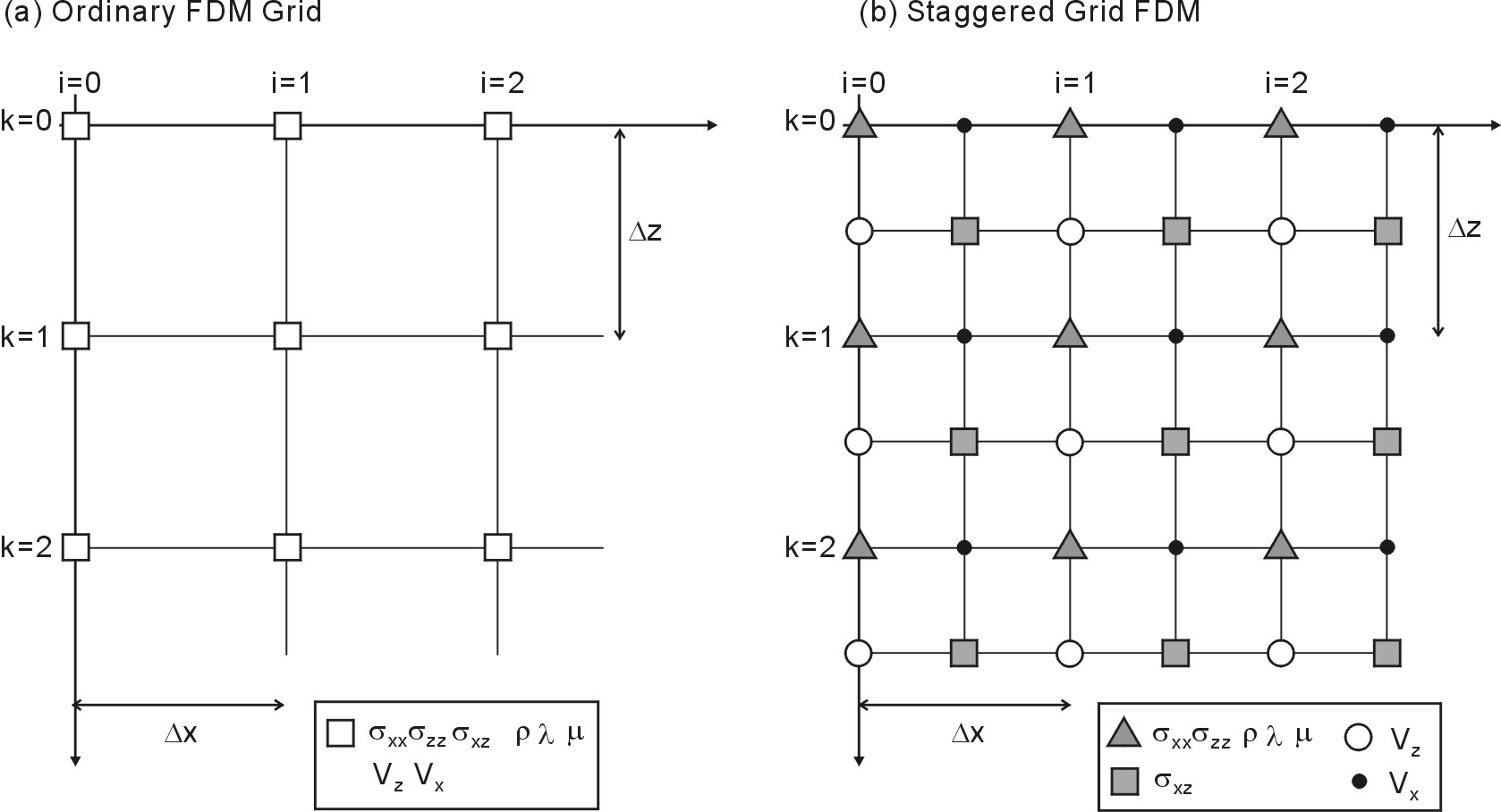
# 1. Outline of seismic\_2D/seismic\_3D

## 1.1 Parallel simulation of seismic wave propagation in heterogeneous elastic media using ppOpen-APPL/FDM

This section provides a brief review of using ppOpen-APPL/FDM for simulation of seismic wave propagation in 3D heterogeneous elastic media based on the solution of equations of motion. The performance of ppOpen-APPL/FDM for parallel simulation based on domain-partitioning procedure is also described below.

## 1.2 Grid and coordinate system

In the 3D FDM simulation of seismic wave propagation, the variables, such as displacement, stress, and a physical-properties value are defined at the same position as if at the position had mutually shifted a half grid size (staggered grid). Spatial derivatives of these variables are defined in the middle of the variables. This staggered-grid coordinate system has been widely used in the FDM simulation of seismic wave propagation since 1980 because it is very effective in improving the accuracy of the FDM simulation even when using a large grid discretization size. Therefore, we adopted the staggered grid coordinate system in the present version of ppOpen-APPL/FDM.



**Fig.1** Layout of each component　on an x-z plane of (a) an ordinary grid and (b) the staggered-grid FDM model.

## 1.3 Equations of Motion for 3D Seismic Waveﬁelds

The seismic waveﬁeld in the 3D Cartesian coordinate system is expressed by equations of motion as

 (1)

where ,  and ρ are stress, body force and density, and  represents particle acceleration. The stresses in an isotropic elastic medium are given by

 (2)

with the Lam´e’s constants λ and µ, and where  denotes Kronecker delta. The strains are deﬁned by

 (3)

With the increment of time of Eq. (1), the velocity at the next time-step t = (n + 1/2)∆t is calculated by integration with time step ∆t as follows.

 (4)

where ˙ is the particle velocity at time t = (n±1/2)∆t.

Differentiation of Eqs. (2) and (3) with respect to time yields

 (5)

## 1.4 Boundary conditions

The boundary conditions for the displacement and stress components at the internal interfaces in the simulation model are implicitly treated in the FDM model by assigning suitable elastic parameters and density at each grid point. The free-surface boundary condition is simply incorporated into the calculation by introducing an air-ﬁlled zone with λ and µ= 0, and a very small value for density ρ over the surface of the Earth.

### 1.4.1 Absorbing boundary

Artiﬁcial reﬂections at the physical boundaries outside of the 3D model are minimized by introducing an absorbing boundary condition based on the tapering approach, or more eﬃciently, by adopting the Perfect Matching Layer. In the present version of ppOpen-APPL/FDM, the absorbing boundary of Cerjan, et al. (1985) is applied at the 20th grid point in the area surrounding the 3D model in order to reduce artificial reflections. More effective absorbing boundary conditions, such as based on the PML, will be introduced in the next release.

### 1.4.2 Free surface boundary

The free surface of air/solid interface (i.e., ground) is described by setting zero to a stress component as:

 (6)



For the FDM simulation of a staggered-grid coordinate system the stress variables are defined as the position at which z=±Dz/2 is separated from the surface of the earth (z= 0), and therefore the zero stress boundary condition cannot be directly incorporated into the FDM calculation. In ppOpen-APPL/FDM a space interpolation of a stress variable, and other effects, are also applied to the stress components  in order to incorporate this boundary condition (6) (see Graves et al., 1996 for more details).

## 1.5 Anelastic attenuations

The seismic waves attenuate in proportion to hypocentral distances due to the spreading of the seismic wavefront and energy; this is called a geometrical spreading. Also, the seismic wave attenuates with the propagation distance due to conversion to thermal energy; this is called intrinsic attenuation. The effect of the geometrical attenuation is estimated by FDM calculation automatically by use of a heterogeneous subsurface structural model. The anelastic intrinsic attenuations for the seismic waves are evaluated in the present time domain FDM simulation by introducing appropriate dampers to Eqs. (4) and (5) by multiplying the following damping coefficient at each time step, as in Graves (1996).

(7)

At this time, the frequency dependence type Q shown below is included in the FDM calculation:

 (8)

where and are the reference frequency and its attenuation coefficient (Q).

Note that the above model results in a strong frequency dependence in the attenuation characteristics with a linearly increasing Q as the frequency increases.

A more advanced procedure, giving constant attenuation over a wide frequency range (), achieved by introducing memory variables for the stress hysteresis curves, is also available.

The present version of ppOpen-APPL/FDM handles seismic wave propagation in a perfectly elastic medium so that the anelastic attenuation model mentioned above will be introduced after the next release.

## 1.6 Spatial Differentiation

The spatial derivatives in Eqs. (4) and (5) are calculated numerically by FDM.

The spatial derivatives of the sequence of data  to be differentiated with respect to x are obtained by means of a centered, staggered-grid FDM, as given by

(9)

where ,(m = 1,2,...,M/2) are the weights of a centered staggered-grid FDM of order M. For example, at the staggered-grid central difference of secondary accuracy scheme of ，and for the 2nd order scheme , and the 8th-order scheme of .

These FDM operation formulas contain the truncation error of the Taylor expansion of the differentiation operation, which causes numerical dispersion noise when the wavelength of the seismic wave is not long enough, compared with the grid size. Higher-order (4th and 8th) FDM schemes have smaller truncation error than a lower-order (2nd) scheme but require much computational cost. In the present release of ppOpen-APPL/FDM the spatial differentiation of the 2nd, 4th and 8th FDM schemes are available. A much higher-order differentiation scheme may be included in the next release.

## 1.7 Input parameters

In the FDM simulation, the area of the 3D model is discretized into elements with uniform grid intervals in the horizontal directions (x and y). A smaller grid interval can be used in the vertical direction in order to introduce lateral variation of structural heterogeneities in the model in more detail. Physical parameters of P-wave speed (Vp; km/s) and S-wave speed (Vs; km/s), and density (ρ; t/m3) are applied at each grid point individually to model a 3D heterogeneous structure. In the present version of ppOpen-APPL/FDM, a simple tool has been prepared to construct a layered structure in the 3D simulation, however, ppOpen-APPL/FDM is capable of simulating an elastic wavefield in a 3D random structure.

The seismic source of a dynamic fault movement is represented by a point source which is described by three fault parameters (dip, strike, and rake) and parameters (t0 and at ) to construct a source time function (duration of source fault rupture). In the present version of ppOpen-APPL/FDM only one point source can be specified by means of an input parameter. However the actual seismic dislocation source for a large earthquake is described by a number of point sources arranged over a fault plane. This multi-source capability will be available for the next release.

## 1.8 Requirements of time integration (CFL Condition)

The propagation of a seismic wave associated with the time integration should be such that it is sufficiently smaller than grid size, which is defined from the following value with minimum grid size in either the x, y, or z directions relative to the maximum distance of the seismic wave propagation in one time step:

(10)

The accuracy of the time integration of a formula (4) and (5) increase when coefficient α is small. However, the number of time steps required for the whole calculation increases, and thus computation time increases. From the viewpoint of the computational cost, an optimal value α is adopted to achieve reasonable accuracy for the FDM simulation. For example, for the 16th- and 2nd-order of the FDM simulation, optimal values of α= 0.24 and 0.45 are often used with a good accuracy balance between the spatial and time differentiation.

## 1.9 Output and visualization of seismic waves

One type of visualization of a 3D simulated seismic wavefield derived by the present version of ppOpen-APPL/FDM is to render the 2D waveﬁeld of surface ground motions, which will be useful to understand the wave propagation and development of strong ground motion on the surface.

A sequence of snapshots of a seismic wavefield of horizontal ground motion (root-mean square of  and ) is obtained by the present simulation. Also, the snapshots of a seismic wavefield can be illustrated as a P and S wavefield, separately, by calculating the divergence and rotation of the 3D seismic wavefield as:

, (11)

We offer visualization utilities in the ppOpen-APPL/FDM /tools subdirectory which will make a visualized image in the ppm image format. Conversion to other image formats, such as jpg and bmp, etc., can be easily accomplished using image converter software such as *Image\_magic* or *xv,* etc. The sequence of rendered images at each time step is finally combined into an MPEG movie using movie maker application software such as *mpeg\_encode* or *ffmpeg,* etc.

The development of effective visualization techniques for 3D seismic waveﬁelds is a challenging issue, particularly when dealing with the large datasets of high-resolution simulations and dense seismic observations. Visualization techniques for a seismic 3D wavefield based on a volume rendering technique will be presented in the next ppOpen-APPL/FDM release.

# 2. Parallel FDM simulation and performance

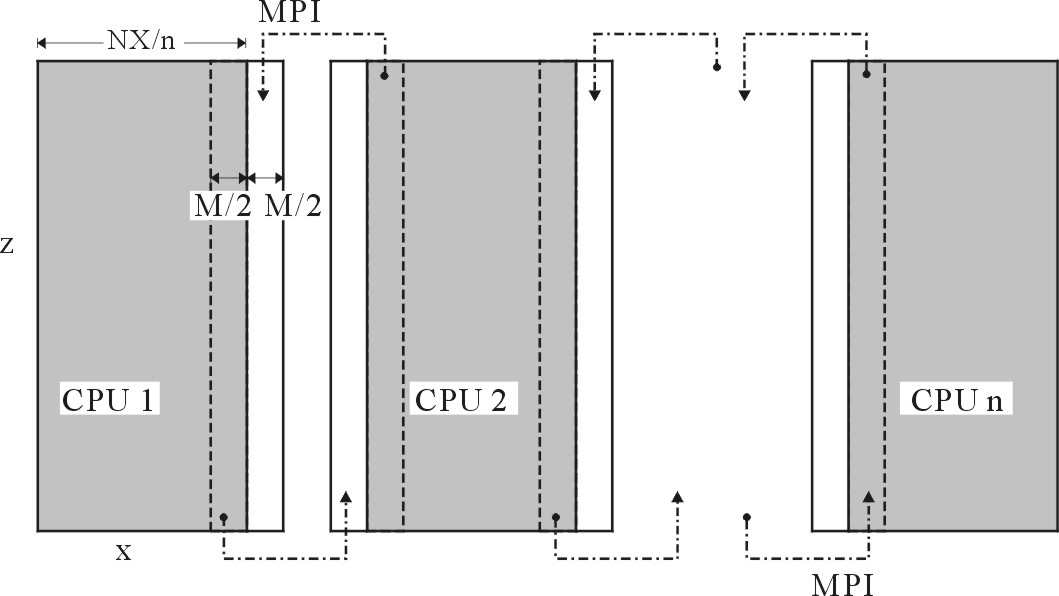
## 2.1 Domain partitioning and MPI

Parallel simulation of a 3D seismic waveﬁeld is achieved by a traditional domain partitioning procedure for massive parallel computing. The present parallel 3D code divides the 3D model into x, y, and z directions and each subregion is assigned to many processors, and a message passing interface (MPI) is employed for the exchange of data between neighboring nodes at each time step.

The data size of MPI communication is defined by the grid size of partitioned subregions in the x,y,z directions and the length of the FDM approximation (e.g., 2nd, 4th, and 8th etc.). For example the data size of inter-processor communication is NX/n\*NY/n\*NZ/n\*M/2 for M-th order FDM with a partitioning of the 3D model into n subregions in the x, y and z directions.

Such MPI communication is required to update stress (six components) and velocity variables (three components) at each time step, before calculating the spatial derivatives of these variables. The cost of communication is usually about 5-10 % relative to the total computation time.

In the presentation version of ppOpen-APPL/FDM an asynchronous MPI communication scheme such as MPI\_ISEND() or MPI\_IRECV() is adopted. Other MPI communication schemes such as using synchronous communication (i.e., MPI\_SENDRECV()), or newly released schemes based on MPI2 (e.g., MPI\_PUT(), MPI\_GET()) will be tested in the next release.



**Fig.2** Domain partitioned parallel FDM simulation. The gray area illustrates subregions assigned to each processor for concurrent computing, and the white bands illustrate the overlapped area of each domain. MPI is used to exchange data between neighboring subregions.

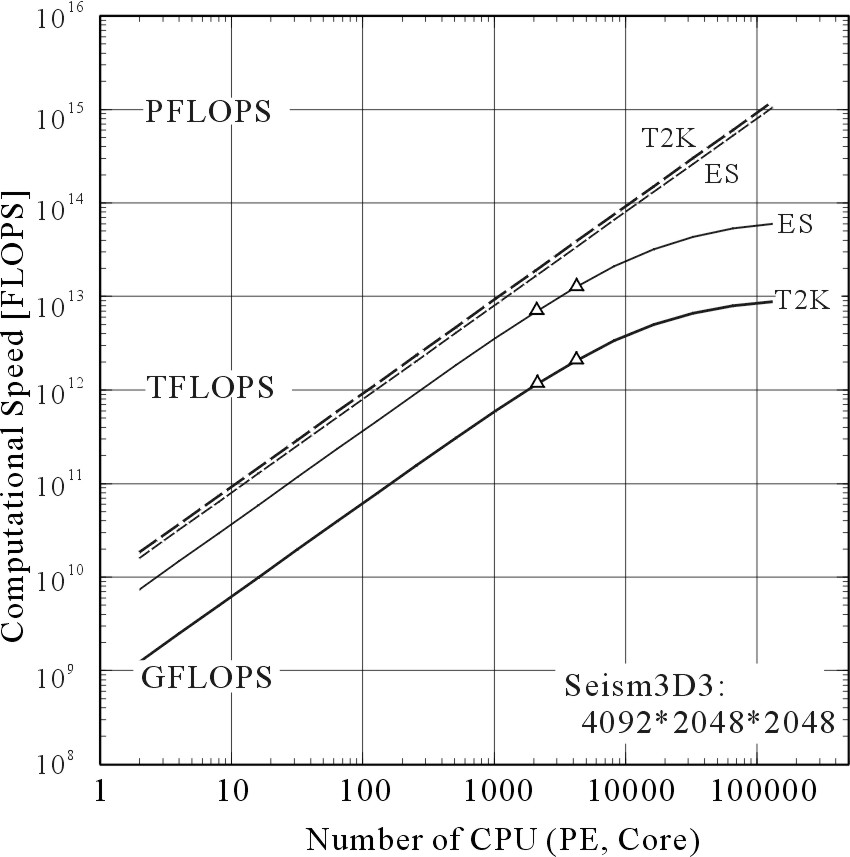
## 2.2 Parallel programming structure

The present version of ppOpen-APPL/FDM has been prepared so it will run on a single-level flat MPI model in which separate single threaded MPI processes are executed on each processor. However it is also possible to run it on a multi-level parallelization using MPI between nodes in combination with loop-level parallelism within each node. Such in-node parallel computing can be achieved by compiler-based automatic thread parallelization and user defined Open MP directives. Note that the FDM in-loop calculation of Eqs. (4) and (5) is rather simple and there is no mutual dependency, so compiler-based automatic parallel computing shows very good performance.

## 2.3 Performance of parallel FDM simulation

Fig. 3 illustrates the speed up of parallel FDM simulation achieved by ppOpen-APPL/FDM as a function of the node number for a range of parallel computers. The size of the simulation is 4092\*2048\*2048 grid points which is equally partitioned into 2048 and 4096 subregions for parallel computing. Computational performance was measured by flat-MPI parallel computing on the T2K supercomputers at University of Tokyo and on the Earth Simulator (ES) (the triangles in Fig.3). An extrapolated performance curve for parallel computing using 2 to 10000 processors of T2K and the ES are examined by Amdar's law (the solid lines), together with theoretical performance for both computers (the broken lines).

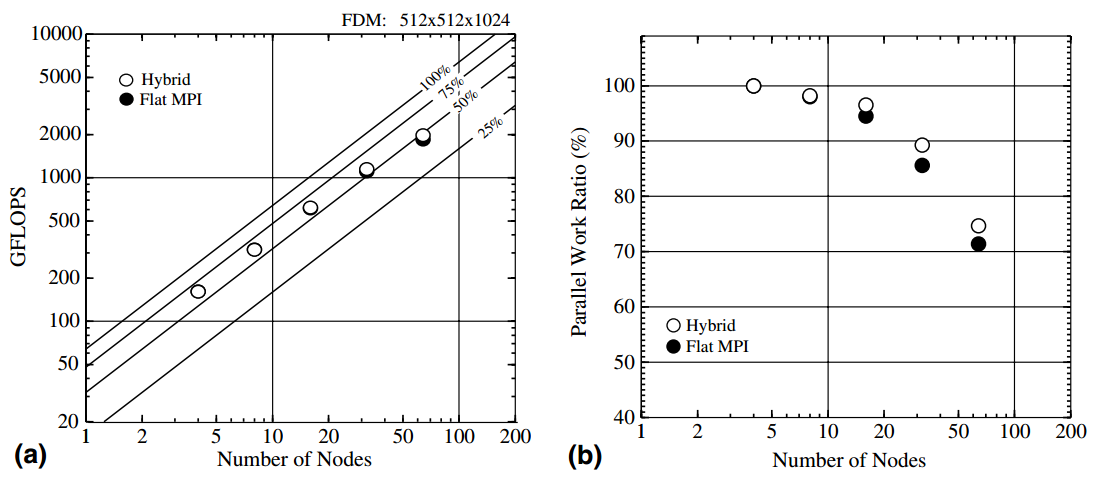
Good speed-up is achieved in parallel computing with a parallel coefficient of 99.9946 % on the ES and 99.9937% on T2K. An almost linear speed-up can be expected with a larger number of processors, up to 18,000 CPUs with good parallel performance larger than 50 %. The sustained performance relative to theoretical performance of the total number of CPUs was 49.2 % for the ES and 8.1 % for T2K.



**Fig. 3** Speed-up of parallel FDM simulation using the Earth Simulator and the T2K supercomputer as a function of processor numbers (nodes). Dashed and solid lines illustrate the theoretical and actual performance of these computer, respectively.

## 2.4 Flat MPI model vs. thread/MPI Hybrid model

The eﬃciency of parallel computing using the ﬂat MPI and the hybrid approaches is shown here by comparing the performance of a small parallel FDM simulation model of 512\*512 \*1024 grid points using the Earth Simulator (8GFLOPS per processor and 64 GFLOPS per node) comparing the two parallel algorithms mentioned above as a function of node numbers (4 to 64 nodes; 32 to 512 processors). For the flat-MPI mode l 8 processors in each node are used individually to run single-thread FDM code, and thus, the process number of the flat-MPI model is 8 times larger than that of the hybrid model on the same test.



**Fig. 4** Parallel performance of the Earth Simulator for 3D parallel FDM simulation using 4–64 nodes with ﬂat MPI and SMP/MPI. (a) Performance in GFLOPS and efficiency (25%,. . ., 100%) relative to peak performance, and (b) parallel work ratio relative to peak performance.

The results shows that both parallel models provide similar levels of performance for small numbers of nodes (processors), the SMP/MPI hybrid model displays somewhat better performance as the number of processors increases. The advantage of the hybrid parallel code is more clearly apparent in terms of the parallel work ratio, deﬁned by the speed-up rate relative to the total number of processors (Fig. 4(b)). The SMP/MPI hybrid approach extracts more performance from the domain-partitioned parallel FDM computing when using large numbers of processors, mainly because the ﬂat MPI requires eight times as many MPI processes as the hybrid model, and so the necessary overhead of MPI data communication between the large numbers of subregions reduces the performance of the FDM simulation.

## 2.5 FlatMPI and Hybrid parallel using by ppOpen-APPL/FDM on FX10

We performed a test of performance on FX10 in the University of Tokyo using by flat MPI and hybrid parallel. The theoretical computing speed is 1.135PFlops per total system. FX10 has many core processers and the theoretical computing speed is 236.5GFlops per node.

For the flat MPI, the performance evaluated a ppOpen-APPL/FDM using by 16 cores to 2048 cores in a model of 256\*256\*256 grid point and 1024\*1024\*1024 grid point. Fig.5 (a) showed the results of total elapse time (blue bar) and speed up ratio (red line) in a model of 256\*256\*256 grid point. Total elapse time is reduced by the number of core number. The calculation time using by 1024 cores is 18 times faster than that using by 16 cores. Moreover, Fig.5 (b) showed the results of total elapse time (blue bar) and speed up ratio (red line) in a model of 1024\*1024\*1024 grid point. Total elapse time was reduced by the number of core number in this model size. This trend is similar to the small model size. The reduction of flops was observed in 2048 cores.

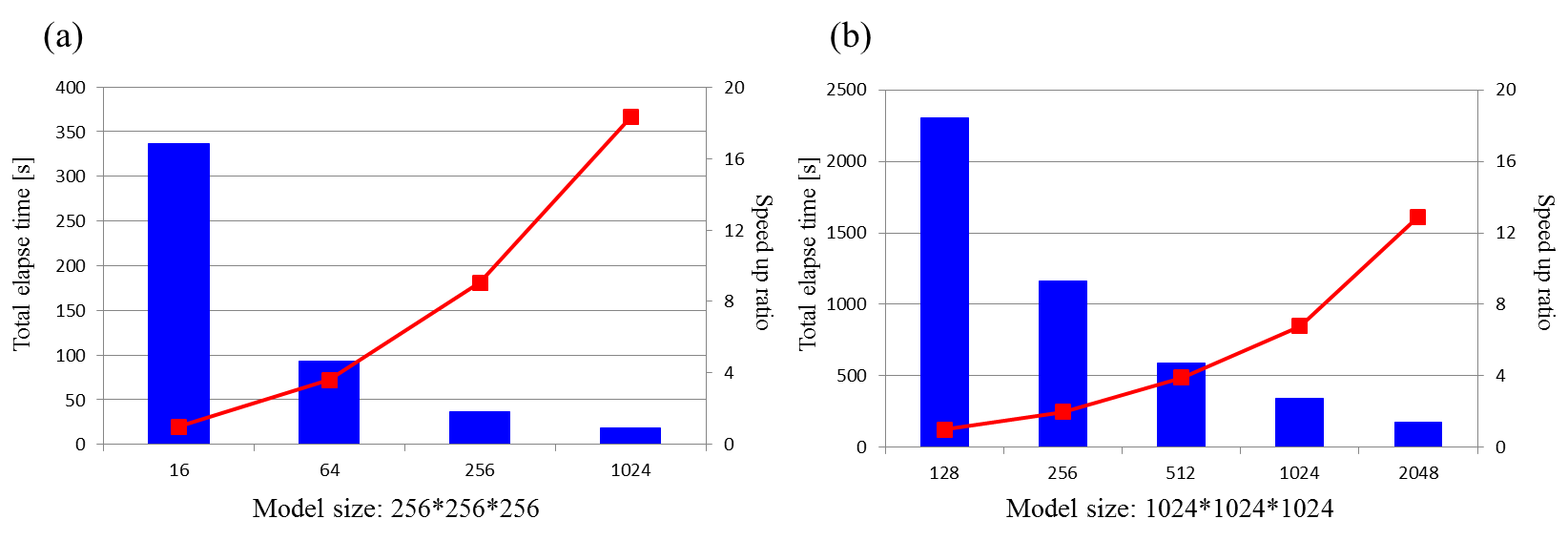


Fig.5 Process parallel performance on FX10 in (a) Model size 256\*256\*256 (b) Model size 1024\*1024\*1024: blue bar indicates total elapse time and red line indicates speed up ratio. X axis is the number of core, left Y axis is total elapse time and right Y axis is speed up ratio.

For the hybrid parallel, the performance evaluated by change process and thread fixed 8 nodes (16 core / node). Model size are 256\*256\*256 grid point and 1024\*1024\*1024 grid point. Fig.6 (a) showed the results of total elapse time (blue bar) and MFlops per peak (red line) in a model of 256\*256\*256 grid point. The minimum of total elapse time was hybrid parallel (64 process and 2 thread). In this model size, hybrid parallel was faster than pure MPI generally. 2× speed up was observed when compared pure MPI to Open MP. Moreover, Fig.6 (b) showed the results of total elapse time (blue bar) and speed up ration (red line) in a model of 1024\*1024\*1024 grid point. The minimum of total elapse time was pure MPI (128 processes).

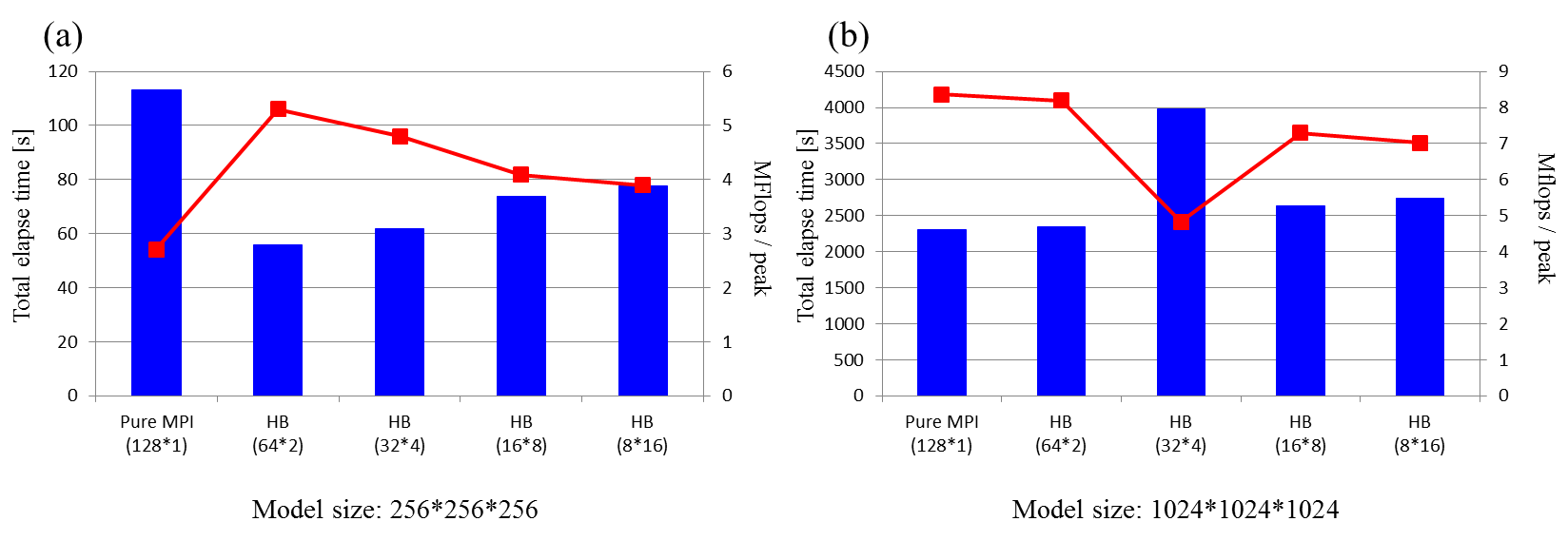


Fig.6 Hybrid parallel performance on FX10 in (a) Model size 256\*256\*256 (b) Model size 1024\*1024\*1024: blue bar indicates total elapse time and red line indicates Mflops per peak. X axis is the number of core, left Y axis is total elapse time and right Y axis is speed up ratio.

## 2.6 Comparison of the strong scaling of MPI/OpenMP hybrid parallel computing and pure MPI parallel computing by ppOpen-APPL/FDM on Intel Xeon Phi

Recently, Intel Xeon Phi coprocessor that is a difference kind of acceleration general purpose graphics processing units (GPGPUs) was developed. By adding the compiler options, the calculation on Intel Xeon Phi coprocessor can be used without changing the code. We then conducted a strong-scaling test of MPI/OpenMP hybrid parallel computing for the two kernels (Fig. 7), for comparison with the results obtained with pure MPI parallel computing. For displaying the results of this test [Fig. 7 (a)], we selected the combinations of MPI process numbers and OpenMP thread numbers that had the shortest calculation times. We then inserted the OpenMP directives in the ppOpen-APPL/FDM prior to executing the outermost DO loops of the kernels and then measured the calculation times. The results of the strong-scaling test of the MPI/OpenMP hybrid parallel computing case show that the calculation time scales down monotonically for up to 240 logical cores. We note that the calculation time for MPI/OpenMP hybrid parallel computing is shorter than that for pure MPI parallel computing using 240 logical cores due to the use of hyper-threading functions. Using 240 logical cores, the speed-up ratio [Fig. 7 (b)] for MPI/OpenMP hybrid parallel computing (8 MPI processes and 30 OpenMP threads, denoted “P8T30”) is 1.57 for the update-velocity and 1.64 for the update-stress over those using pure MPI parallel computing (60 MPI processes). In the case of pure MPI parallel computing using 240 cores (P240T1), 4 MPI processes are assigned to a physical core. In other words, the local cache of the core is shared with the 4 processes. On the other hand, in the case of MPI/OpenMP hybrid parallel computing using 240 cores (P8T30), 1 MPI process is assigned to 7.5 physical cores. Therefore, the cache efficiency of the P8T30 is greater than of the P240T1. MPI/OpenMP hybrid parallel computing is faster than pure MPI parallel computing because of its utilization of large-scale thread-parallel computing with hyper-threading functions.

## 2.7 Performance tuning of kernel loop optimization of MPI/OpenMP hybrid parallel computing by ppOpen-APPL/FDM on Intel Xeon Phi

The 3D FDM simulation kernels consist of triple DO loops with respect to the i, j, and k directions. It is well recognized that the fusion of many DO loops yields efficient thread-parallel computing performance with increasing loop length while decreasing the overhead of thread-parallel computing We examined the effectiveness of loop fusion for MPI/OpenMP hybrid parallel computing by using 240-core hyper-thread parallel computing, with the results as given in Fig. 8. There, we indicate the shortest calculation time for each combination of MPI process (P) numbers and OpenMP thread (T) numbers. With the double DO-loop fusion, the calculation for P30T8 is 2.2 times faster for the update-velocity kernel, compared with the triple DO loop.

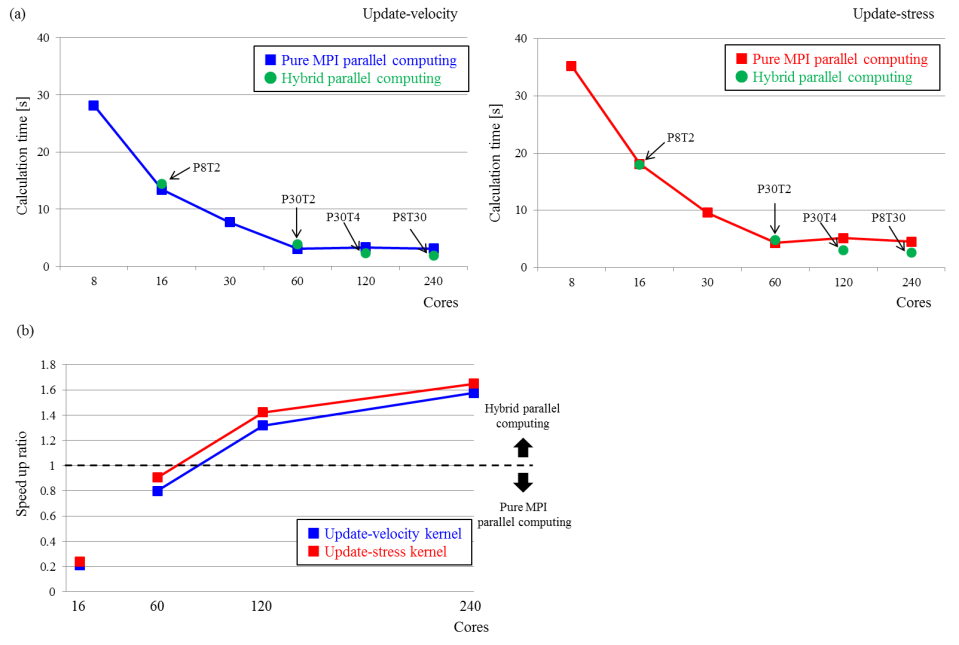


Fig. 7. Comparison of the strong scaling of MPI/OpenMP hybrid parallel computing (green points) and pure MPI parallel computing (blue and red lines) for the (a) update-velocity and update-stress kernels and (b) the speed up ratio comparison between 60 MPI parallel processes and each hybrid parallel computing. In (a), the horizontal axis is the number of cores and the vertical axis is the calculation time. In (b), the black dashed line indicates the reference line, above parallel computing is faster. In MPI/OpenMP hybrid parallel computing, we selected the shortest calculation time for each combination of MPI process (P) numbers and OpenMP thread (T) numbers.

Loop distribution is the inverse process of loop fusion and divides a loop into multiple loops to improve the spatial locality. We investigated the loop distribution of the triple DO loops for MPI/OpenMP hybrid parallel computing by using 240-core hyper-thread parallel computing; the resulting calculation (for P120T2) is 2.32 times faster for the update-velocity kernel than that of the triple DO loop, as is shown in Fig. 8. Loop distribution optimization is applied to prefetch and to the thread parallelization of each loop by its use and reuse on cache data. Moreover, the combination of loop fusion and loop distribution is the most fastest than other parallel case.

## 2.8 Performance evaluation of the program for decreasing the B/F ratio on Intel Xeon Phi coprocessor

The FDM simulation of seismic wave propagation using ppOpen-APPL/FDM requires high memory bandwidth. Thus the memory bandwidth (bytes) is larger than the computational cost of the processors (flop). For example, the required B/F ratios for calculating the update-velocity and the update-stress kernels are 2.7 and 1.7, respectively, much larger than those required by current processors (0.01–0.1). In the original code, the values of each spatial derivative are stored in a corresponding array. Thus, to obtain good computational performance on today’s powerful processors, we need to either make full use of the cache memory of the processors or modify the program structure to reduce the required B/F ratio, or both.

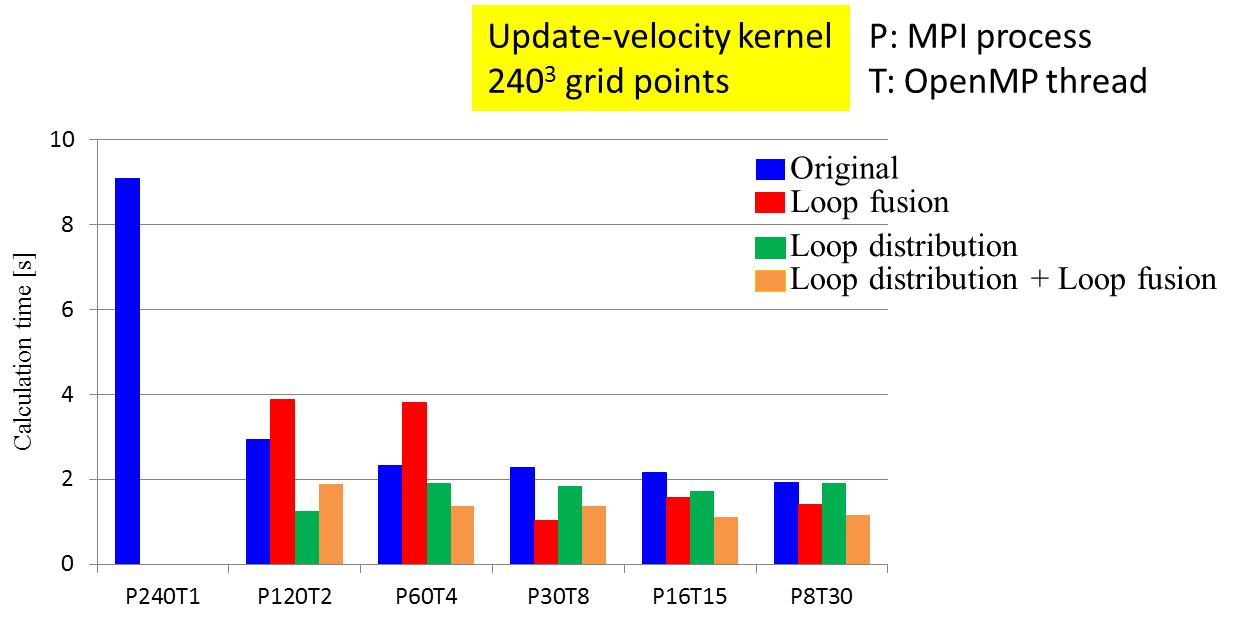


Fig.8 Effect of loop fusion and loop distribution with MPI/OpenMP hybrid parallel computing (HB) using 240-core hyper-thread computation in update-velocity kernel. The blue bar indicates triple DO loops, the red bar indicates double DO-loop fusion, the green bar indicates loop distribution of the triple DO loops and the orange bar indicates combination of loop distribution and loop fusion. The horizontal axis is each parallel case, and the vertical axis is the calculation time. We indicate the shortest calculation time for each combination of MPI process (P) numbers and OpenMP thread (T) numbers in the original and the modified code.

Figure 9 shows a procedure we propose to effectively decrease the required B/F ratio of the FDM simulation of seismic wave propagation. In the original simulation code, we first calculated each spatial derivative of the velocity and stress components and stored them in memory. These derivatives were then used for the kernels of update-stress and update-velocity, requiring large B/F ratios in order to load and store the large number of variables. To overcome this challenge, we modified the FDM code to merge the derivative and update calculations, thereby avoiding the need to store and load variables during the calculation. As a result, the required B/F ratios for both the update-velocity and the update-stress kernels dropped dramatically, to 0.4.

Figure 10 displays the comparison of the performance of the original and modified code in MPI/OpenMP hybrid parallel computing. In Fig. 6, we show the shortest calculation times for each combination of MPI process (P) numbers and OpenMP thread (T) numbers. For parallel computing up to 60 physical cores, the modified code is slower than the original code. However, with the much larger scale parallel simulation using 240 logical cores, the modified code doubles the speed of the original code because of the moderation of the restriction on the memory bandwidth achieved by decreasing the B/F ratio. Effective performance can be obtained even in modifying code on FX10

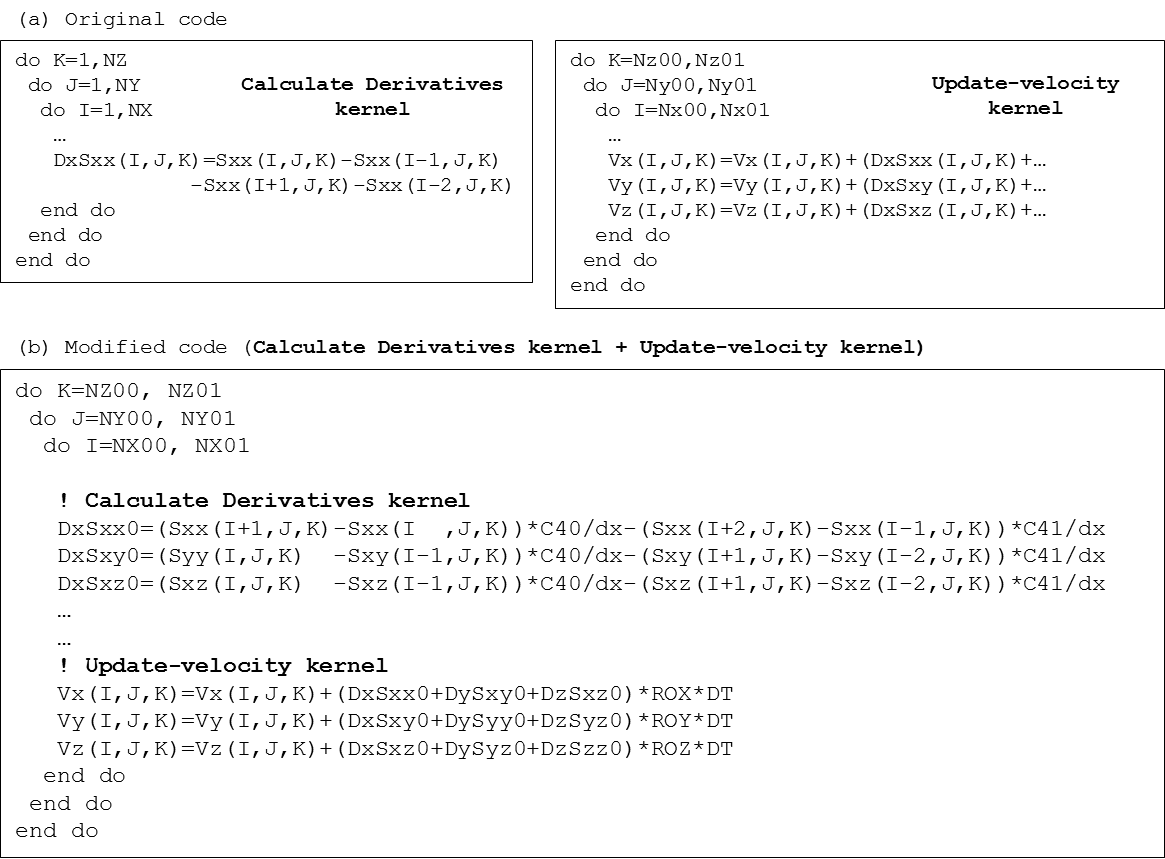


Fig. 9. Procedure for decreasing the B/F ratio: (a) original code and (b) modified code. The required B/F of the update-velocity kernel (2.7) and the update-stress kernel (1.7) kernels can be reduced dramatically to 0.4 in each kernel. In (b), C40, C41, ROX, ROY, ROZ, dx and DT are constants.

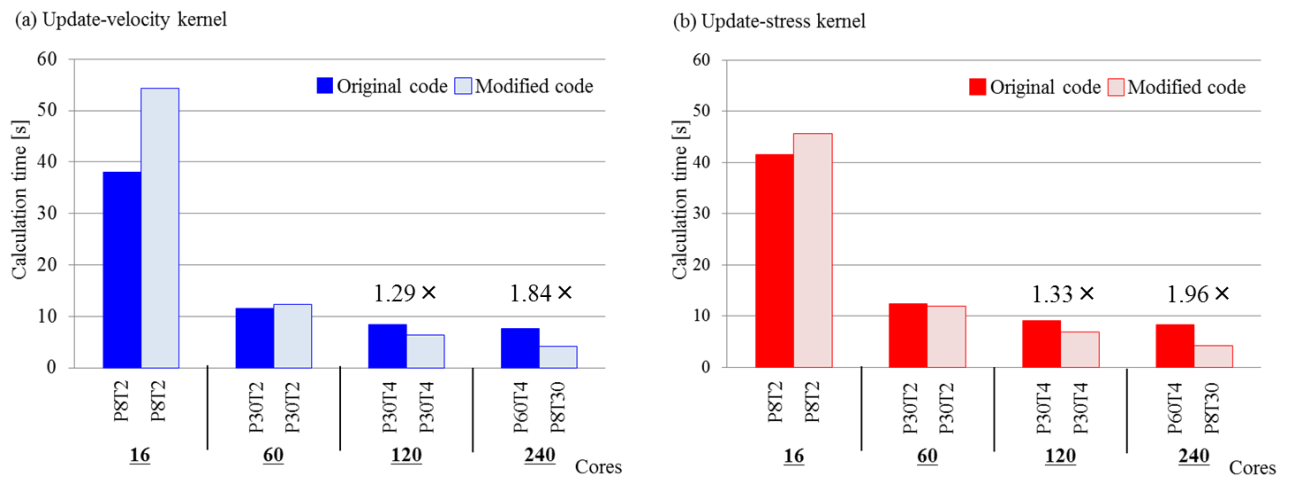


Fig. 10. Comparison of MPI/OpenMP hybrid parallel computing based on the original code and on the modified B/F-reduction code for the (a) update-velocity and (b) update-stress kernels. The horizontal axis is the number of cores and the vertical axis is the calculation time. On the horizontal axis, we indicate the shortest calculation time for each combination of MPI process (P) numbers and OpenMP thread (T) numbers in the original and the modified code.

## 2.9 Simulation using NVIDIA GPUs by using OpenACC directive

Running ppopen-APPL/FDM requires wide memory bandwidth as an essential properties of FDM even after conducting a performance tuning to reduce memory access (section 2.8). On the other hand recent General Purpose Graphics Processing Units (GPGPU) has an advantage in ths issue, having relatively wider memory bandwidth inside the GPGPU in comparison with the contemporary CPUs to memory access. The applications of the GPGPU for the FDM simulation have already been succeed by many groups for application to seismic wave propagation (Aoi et al., 2012; Okamoto, 2010) using CUDA compiler. However, the use of CUDA C or CUDA FORTRAN requests users to totally re-write their codes.

Recently, a new, directive-based, OpenACC programing environment has been developed. It allow the uses to use GPGPU very easily by adding simple directives to the codes, as like as the OpenMP.

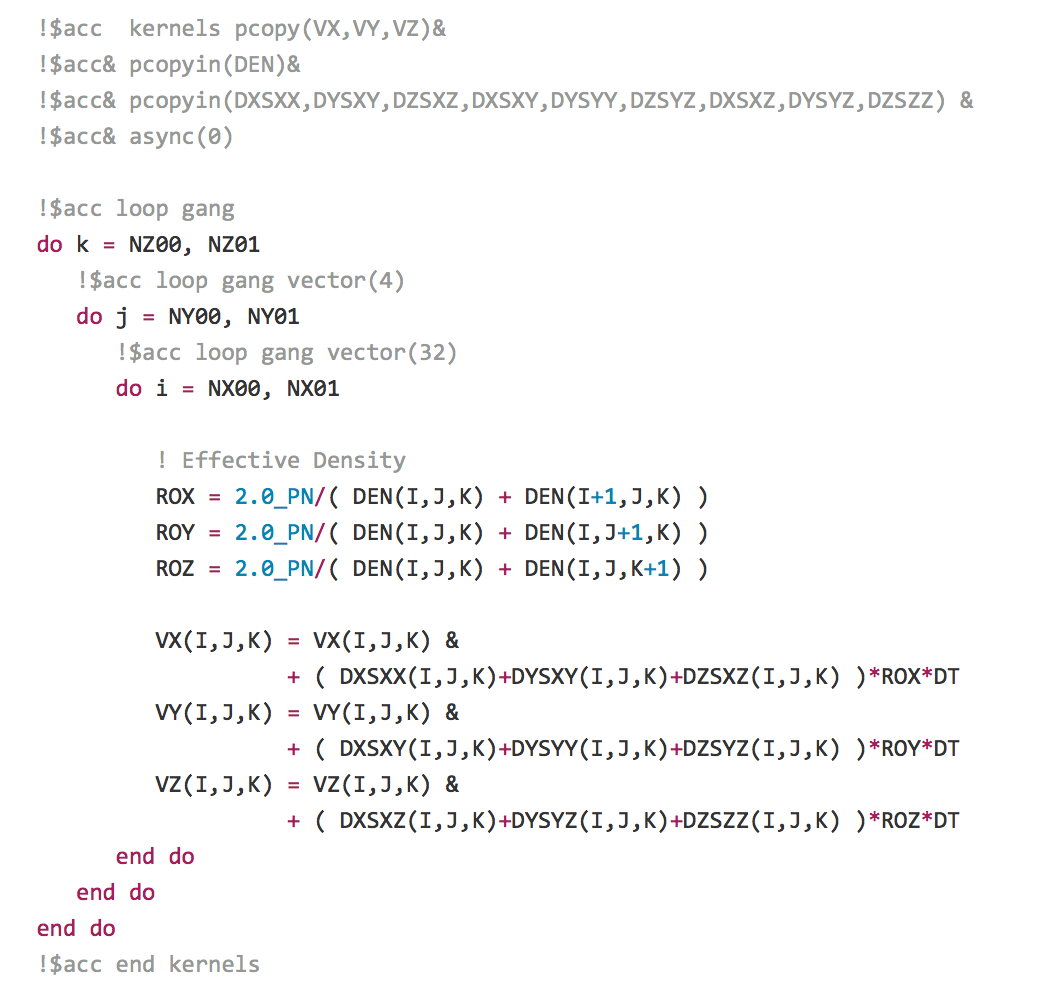


Figure 10. An example of implementation of OpenACC directive to the ppopen-APPL/FDM.

We implemented the OpenACC directives to the Seismic\_3D code with pure-MPI parallel scheme (Fig.10 for example). Codes between "!$acc kernels" and "!$acc end kernels" directives will be calculated on GPU. Variables as density (DEN), velocity (Vx,Vy,Vz), and stress tensor (DXSXX, DYSXY…) are first transferred to GPU from the CPU host at the first use (by pcopyin directive, see Fig. 10)... Therefore, memory on a GPU must be large enough to store all memory required in the subregion. The "!$acc loop" directive before each do loop indicates that parallel computing using many cores of GPU, where the size of threads are indicated by "vector" directive. Finding a proper size of threads requires a fine tuning. We have conducted additional kernel loop optimizations to achieve better computational efficiency on GPGPU. For MPI communication a small amount of data around the edge of subregions (see Fig.2) are transferred at every time step to the host CPU to communicate to neighbor node before and after MPI data communication. During the computation time on GPU no CPU-GPU data transmission occurs. We can confirm this using the NVIDIA Visual Profiler (nvvp; Fig. 11).

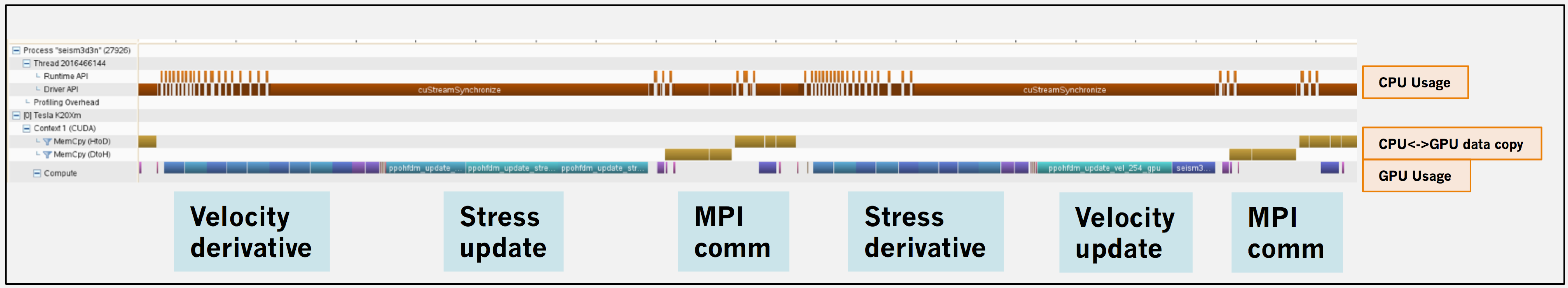


Figure 11. Example of the profiling result from NVIDIA Visual Profiler (nvvp). To demonstrate the GPU and CPU activities during the calculation.

We measured the computational performance of GPGPU computing on Tsubame 2.5 at the Tokyo Institute of Technology. The Tsubame2.5 equipped with the Intel Xeon X5670 2.97GHz (6 core) as CPU and NVIDIA K20x as GPU each have memory bandwidths of 32 GB/s and 250 GB/s, respectively. Computational speed calculated by the same number of sockets (8 CPUs and 8 GPUs) are compared. The 3-D simulation model of 384x512x512 grids are partitioned into 6x4x2 (=48, 6Core x 8CPU) for CPU, and 2x2x2 (=8) for GPU. For GPU computation, each GPU assigned with single core of the CPU for MPI data communication. During the test data output such as storing the snapshot and waveform data on disk was omitted.

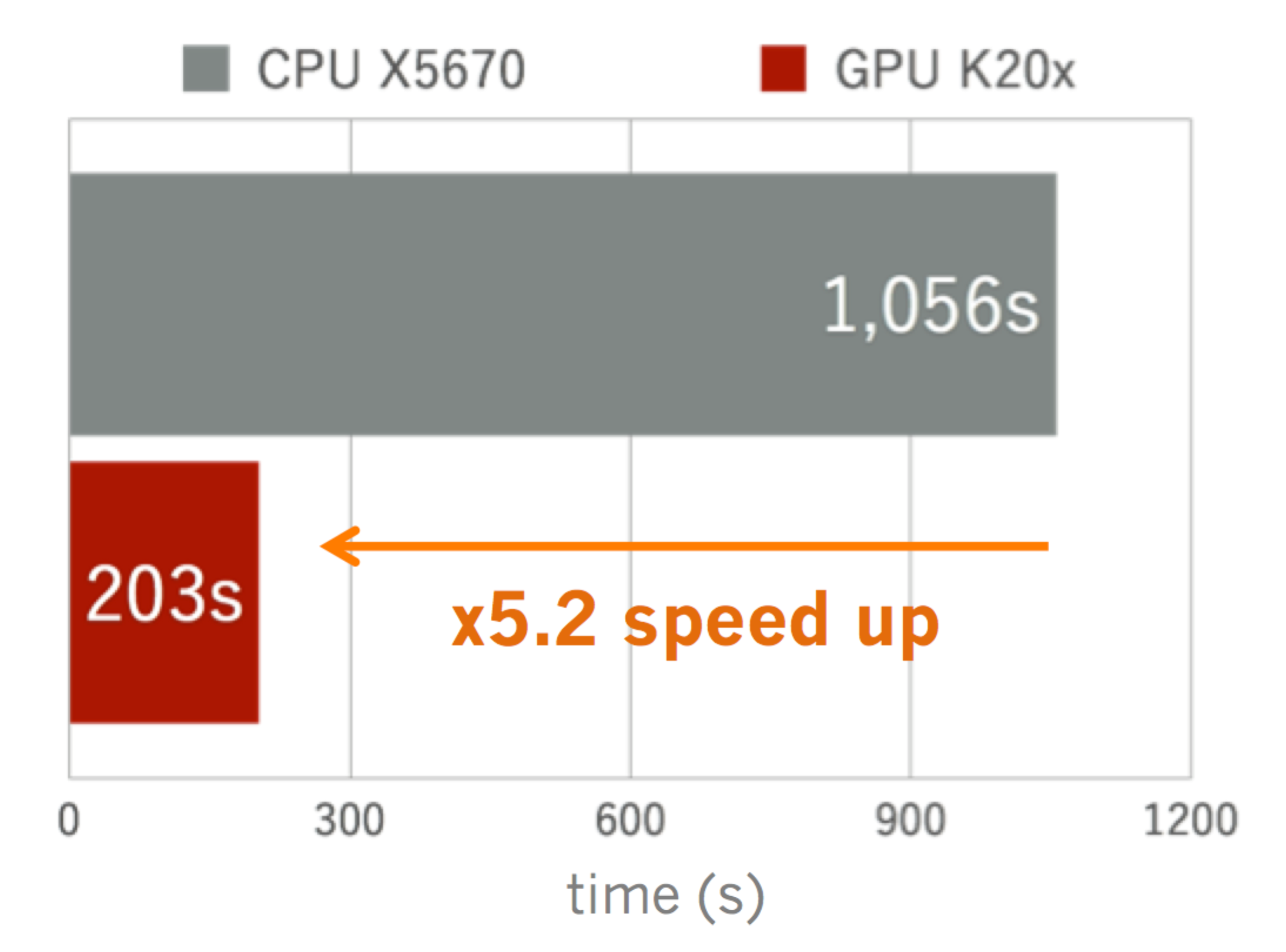


Figure 12. The comparison of computational speed of open-APPL/FDM between CPU and GPU examined by using the Tsubame 2.5.

Figure 12 compares the obtained performance of the open-APPL/FDM on CPU and GPU.. We achieved 5.2 times faster speed-up by using GPUs relative to the CPU on Tsubame 2.5. Because the memory bandwidth of the K20x GPU is 7.81 times wider than that of the X5670 CPU. The results of the present examination corresponds to the 67 % use of the theoretical performance of the GPU by using the OpenACC.

Next, we conducted a week-scaling test of GPGPU assigning the same grid size (64x64x512) per single GPU and examined parallel simulation using large number of GPGPU (Figure 13). In this test, we measured computation times with and without I/O to examine the parallel performance and I/O cost, separately.. In the case without the data I/O, the computation time is nearly constant at number of GPUs larger than 10,indicating good parallel performance on GPU. However, the computation time abruptly increase at larger number of GPUs (>100) when data transmitted from GPU to CPU is conducted to achieve I/O. To overcome this, the overlap between GPU computation and GPU-CPU data transmission should be considered.

Macintosh HD:private:var:folders:6t:60f4ffss3qj32zhntpd493rw00049j:T:com.evernote.Evernote:WebKitDnD.qAB5G3:wscale.pdf

Figure 13. Result of the weak-scaling test of ppOpen-APP/FDM on Tsubame 2.5 by using the OpenACC. The computation times measured with (red circles) and without (blue squares) I/O are plotted as a function of GPU numbers. The grid size per node is fixed to 64x64x512.

# References

Aoi, S., Maeda, T., Nishizawa, N., and Aoki, T. (2012): Large-scale ground motion simulation using GPGPU. AGU Fall Meeting Abstract S53G-06.

Cerjan, C., Kosloff, K., Kosloff, R., Reshef, M. (1985): A nonreﬂecting boundary condition for discrete acoustic and elastic wave equation. Geophysics, vol. 50, pp. 705–708.

Graves, R. (1996): Simulating seismic wave propagation in 3-D elastic media using staggered-grid ﬁnite differences. Bull. Seism. Soc. Am., vol. 86, pp. 1091–1106.

Furumura, T., and L. Chen (2004). Large scale parallel simulation and visualization of 3D seismic waveﬁeld using the Earth Simulator, Comput. Model. Eng. Sci. 6, 143–168.

Furumura, T., and L. Chen (2005). Parallel simulation of strong ground motions during recent and historical damaging earthquakes in Tokyo, Japan, Parallel Comput. 31, 149–165

Mori F, Matsumoto M, Furumura T. "Performance of FDM Simulation of Seismic Wave Propagation using the ppOpen-APPL/FDM Library on the Intel Xeon Phi Coprocessor", 11th International Meeting High Performance Computing for Computational Science (VECPAR2014)

Mori F, Matsumoto M, Furumura T. "Performance Optimization of the 3D FDM Simulation of Seismic Wave Propagation on the Intel Xeon Phi Co-processor using the ppOpen-APPL/FDM Library", Lecture Notes in Computer Science (Accepted).

Okamoto, T., Takenaka, H., Nakamura, T., and Aoki, T. (2010): Accelerating large-scale simulation of seismic wave propagation by multi-GPUs and three-dimensional domain decomposition. Earth Planet. Space, vol. 62, 939-942.

Katagiri, T., Ito, S., Ohshima, S.: Early Experiences for Adaptation of Auto-tuning by ppOpen-AT to an Explicit Method. Proceedings of MCSoC2013, pp.153-158 (2013).

# 3. **Module/subroutine reference**

## 3.1 seism2d\_psv

**Description**

seism2d\_psv.f90 is Fortran code for explicit FDM simulation of elastic (seismic) wave propagation in heterogeneous elastic media in 2D due to the radiation of a seismic wave from a double-couple seismic source in the medium. This simulation code is constructed using the following modules and subroutines. The simulation is formed by a preparation part for initialization and parameter setting and a main part to repeat calculation with the marching of time, and a post processing part for termination of the simulation.

**Dependency**

use ppohFDM\_m\_kernel

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_comvar

use ppohFDM\_m\_source

use ppohFDM\_m\_report

use ppohFDM\_m\_absorb

use ppohFDM\_m\_output

use ppohFDM\_m\_medium

use ppohFDM\_m\_swatch

use ppohFDM\_m\_surfbc

use ppohFDM\_m\_params

use ppohFDM\_m\_kernel

main part

**Description**

The FDM simulation is conducted explicitly to simulate the spread of a seismic wave from a source in heterogeneous media over time. At each time step, calculation of an equation of motion with suitable boundary conditions at an interface is evaluated by calling the following subroutines. The snapshots of the seismic wavefield and waveforms are stored.

**Flow of calculation**

call ppohFDM\_report\_\_init\_counter()

timestep: do it=1, NTMAX

call ppohFDM\_report\_\_progress(it)

call ppohFDM\_output\_\_write\_snap(it)

call ppohFDM\_kernel\_\_stressderiv()

call ppohFDM\_surfbc\_\_stressderiv()

call ppohFDM\_kernel\_\_update\_vel()

call ppohFDM\_absorb\_\_update\_vel()

call ppohFDM\_source\_\_bforce(it)

call ppohFDM\_kernel\_\_velderiv()

call ppohFDM\_surfbc\_\_velderiv()

call ppohFDM\_kernel\_\_update\_stress()

call ppohFDM\_absorb\_\_update\_stress()

call ppohFDM\_surfbc\_\_zerostress()

end do timestep

**finalizing part**

**Description**

The finalizing part of the FDM simulation terminates the program run by closing files and listing simulation performance, etc.

**Flow of calculation**

call ppohFDM\_output\_\_close\_files()

call ppohFDM\_swatch\_\_report(STDERR, .true.)

call ppohFDM\_report\_\_goodby()

stop

### 3.1.1 module ppohFDM\_m\_absorb

**Description**

Sponge-type absorbing boundary based on Cerjan, et al. (1985) to eliminate artificial reflection from the edges of bounded media.

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_params, only : Nx1, Nz1, Nx, Nz, NPM, Dt

use ppohFDM\_m\_comvar

use ppohFDM\_m\_kernel

**Public Procedures**

public :: ppohFDM\_absorb\_\_setup

public :: ppohFDM\_absorb\_\_update\_vel

public :: ppohFDM\_absorb\_\_update\_stress

**Public Variables**

real(PN), save :: gg(Nx1,Nz1) ! Absorption coefficients

**subroutine ppohFDM\_absorb\_\_setup()**

**Description**

Sets up absorbing conditions (attenuation coefficients) in the sponge buffer zone surrounding the medium.

**subroutine ppohFDM\_absorb\_\_update\_vel()**

**Description**

Updates the velocity vector in the absorbing area.

**subroutine ppohFDM\_absorb\_\_update\_stress()**

**Description**

Updates the stress tensor in the absorbing area.

### 3.1.2 module ppohFDM\_m\_comvar

**Description**

Common variables used in the 2D　FDM　simulation of seismic wave propagation.

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_params, only: Nx1, Nz1

**subroutine ppohFDM\_comvar\_\_setup()**

**Description**

Initialization of common variables.

### 3.1.3 module ppohFDM\_m\_kernel

**Description**

The kernel for FDM simulation of a seismic wave in 2D, including spatial differentiation of variables and updating (time integration) of the velocity vector and the stress tensor

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_comvar

use ppohFDM\_m\_params, only : Nx, Nz, Nx1, Nz1, Dx, Dz, Dt, NPM

**Constants for finite difference calculation**

real(PN), parameter :: C20 = 1.0\_PN ! 2nd-order FDM

real(PN), parameter :: C40 = 1.125\_PN ! 4th-order FDM

real(PN), parameter :: C41 = 1.0\_PN / 24.0\_PN

**Public Procedures**

public :: ppohFDM\_kernel\_\_setup

public :: ppohFDM\_kernel\_\_velderiv

public :: ppohFDM\_kernel\_\_stressderiv

public :: ppohFDM\_kernel\_\_update\_vel

public :: ppohFDM\_kernel\_\_update\_stress

**subroutine ppohFDM\_kernel\_\_setup()**

**Description**

Initialization of local variables

**subroutine ppohFDM\_kernel\_\_stressderiv()**

**Description**

Spatial differentiation (dp Spq, pq=xz) of the stress tensor (Spq).

**subroutine ppohFDM\_kernel\_\_velderiv()**

**Description**

Spatial differentiation (dp Vq, pq=xz) of the velocity vector (Vp).

**subroutine ppohFDM\_kernel\_\_update\_vel(i0, i1, k0, k1)**

**Description**

Update of the velocity vector in the area (i0:i1,k0:k1), excluding the absorbing buffer zone.

**Arguments**

integer, intent(in), optional :: i0, i1, k0, k1

**subroutine ppohFDM\_kernel\_\_update\_stress(i0,i1,k0,k1)**

**Description**

Update of the stress tensor in the area (i0:i1,k0:k1), excluding the absorbing buffer zone.

**Arguments**

integer, intent(in), optional :: i0, i1, k0, k1

**subroutine ppohFDM\_fdiffz2\_m4( V, DzV, Nx, Nz, Nx1, Nz1, Dz )**

**Description**

Spatial differentiation of the velocity vector using a 4th-order staggered-grid FDM.

**Arguments**

integer, intent(in) :: Nx

integer, intent(in) :: Nz

integer, intent(in) :: Nx1

integer, intent(in) :: Nz1

real(PN), intent(in) :: V(Nx1, Nz1)

real(PN), intent(in) :: Dz

real(PN), intent(out) :: DzV(Nx1,Nz1)

**subroutine ppohFDM\_fdiffz2\_p4( V, DzV, Nx, Nz, Nx1, Nz1, Dz )**

**Description**

Spatial differentiation of the velocity vector using a 4th-order staggered-grid FDM.

**Arguments**

integer, intent(in) :: Nx

integer, intent(in) :: Nz

integer, intent(in) :: Nx1

integer, intent(in) :: Nz1

real(PN), intent(in) :: V(Nx1, Nz1)

real(PN), intent(in) :: Dz

real(PN), intent(out) :: DzV(Nx1,Nz1)

**subroutine ppohFDM\_fdiffx2\_p4 ( V, DxV, Nx, Nz, Nx1, Nz1, Dx )**

**Description**

Spatial differentiation of the velocity vector using a 4th-order staggered-grid FDM.

**Arguments**

integer, intent(in) :: Nx ! size of area in X

integer, intent(in) :: Nz ! in Z

integer, intent(in) :: Nx1 ! actual size of array in X

integer, intent(in) :: Nz1 !

real(PN), intent(in) :: V(Nx1, Nz1) ! velocity vector

real(PN), intent(in) :: Dx ! grid size in X

real(PN), intent(out) :: DxV(Nx1,Nz1) ! spatial differentiation of V according to X

**subroutine ppohFDM\_fdiffx2\_m4 ( V, DxV, Nx, Nz, Nx1, Nz1, Dx )**

**Description**

Spatial differentiation of the velocity vector using a 4th-order staggered-grid FDM.

**Arguments**

integer, intent(in) :: Nx

integer, intent(in) :: Nz

integer, intent(in) :: Nx1

integer, intent(in) :: Nz1

real(PN), intent(in) :: V(Nx1, Nz1)

real(PN), intent(in) :: Dx

real(PN), intent(out) :: DxV(Nx1,Nz1)

### 3.1.4 module ppohFDM\_m\_medium

**Description**

Sets up elastic parameters in the simulation model.

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_comvar, only: den, rig, lam, RO, VP, VS, RO1, VP1, VS1

use ppohFDM\_m\_params, only: Nx1, Nz1, Nx, NPM, KFS, Dx, Dz, ZDEP, MST

**Public Procedures**

public :: ppohFDM\_medium\_\_setup

**subroutine ppohFDM\_medium\_\_setup()**

**Description**

Sets up user defined elastic parameters (Den, Rig, Ram).

### 3.1.5 module ppohFDM\_m\_output

**Description**

This module describes output of the simulation results.

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_params !, only : NX, NZ, TITLE, DT, NTMAX, dx, dz, I0, K0

use ppohFDM\_m\_source !, only : I0, K0

use ppohFDM\_m\_medium !, only : is\_fs, kfsz

use ppohFDM\_m\_comvar

**Public Procedures**

public :: ppohFDM\_output\_\_setup

public :: ppohFDM\_output\_\_write\_snap

public :: ppohFDM\_output\_\_close\_files

**subroutine ppohFDM\_output\_\_setup()**

**Description**

Opens an output file.

**subroutine ppohFDM\_output\_\_close\_files()**

**Description**

Closes an output file.

**subroutine ppohFDM\_output\_\_write\_snap(it)**

**Description**

Dumps snapshots of a seismic wavefield at time step "it".

**Arguments**

integer, intent(in) :: it ! time step

### 3.1.6 module ppohFDM\_m\_params

**Description**

Sets up model parameters for an FDM simulation of an elastic seismic wavefield

**Dependency**

use ppohFDM\_m\_stdlib

**Parameters**

-<< Execute Title >>

character(99), parameter :: TITLE = "Seism2D" ! Title of this job

-- << Model Size and Grid Width >>

integer, parameter :: NX = 512 ! number of grid points in X

integer, parameter :: NZ = 256 ! number of grid points in Y

integer, parameter :: NX1 = NX+1

integer, parameter :: NZ1 = NZ+1

integer, parameter :: KFS = 50 ! Free Surface Grid

integer, parameter :: NTMAX = 10000 ! maximum time step

real(PN), parameter :: DX = 0.5\_PN ! grid size in X (km)

real(PN), parameter :: DZ = 0.5\_PN ! in Z (km)

real(PN), parameter :: DT = 0.0025\_PN ! time increment (s)

-- << ABSORBING BOUNDARY CONDITION >>

integer, Parameter :: NPM = 20 ! Layer Thickness (10~20) for absorbing zone

### 3.1.7 module ppohFDM\_m\_report

**Description**

This module displays simulation parameters and computational time on the screen.

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_params, only : NX1, NZ1, NX, NZ, DT, DX, DZ, NTMAX, NPM, KFS

use ppohFDM\_m\_medium, only : kfsz

use ppohFDM\_m\_output, only : NWRITE

use ppohFDM\_m\_comvar

**Public Procedures**

public :: ppohFDM\_report\_\_welcome

public :: ppohFDM\_report\_\_goodby

public :: ppohFDM\_report\_\_stblcnd

public :: ppohFDM\_report\_\_progress

public :: ppohFDM\_report\_\_init\_counter

public :: ppohFDM\_report\_\_output\_prm

**subroutine ppohFDM\_report\_\_output\_prm**

**Description**

Displays simulation parameters.

**subroutine ppohFDM\_report\_\_welcome()**

**Description**

Displays the simulation start time.

**subroutine ppohFDM\_report\_\_goodby()**

**Description**

Displays the simulation end time.

**subroutine ppohFDM\_report\_\_stblcnd ()**

**Description**

Examines availability of simulation parameters to satisfy computational stability conditions.

**subroutine ppohFDM\_report\_\_init\_counter()**

**Description**

Starts measurement of time.

**subroutine ppohFDM\_report\_\_progress( it )**

**Description**

Displays the progress of calculation.

**subroutine get\_max(xmax, zmax)**

**Description**

Getting the maximum amplitude for the confirmation screen display

### 3.1.8 module ppohFDM\_m\_source

**Description**

This module sets up the source-time function used for 2D simulation

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_params

**Public Procedures**

public :: ppohFDM\_source\_\_setup

public :: ppohFDM\_source\_\_bforce

**subroutine ppohFDM\_source\_\_setup\_param()**

**Description**

Sets up the source parameters defined in "source.dat".

**subroutine ppohFDM\_source\_\_setup()**

**Description**

Sets up the sample source (preparing for a benchmark test).

**subroutine ppohFDM\_source\_\_bforce(it)**

**Description**

Adopts the effect of a seismic source by using body force.

**Arguments**

integer, intent(in) :: it ! time step

**subroutine ppohFDM\_sld2moment2(strike, dip, rake, M0, Mxx, Mzz, Mxz )**

**Description**

Calculates six moment tensor components from strike, dip, and rake [deg] under the double-couple assumptions for 2D code

**Arguments**

real(PN), intent(in) :: strike, dip, rake ! fault mechanism parameters

real(PN), intent(in) :: M0 ! seismic moment

real(PN), intent(out) :: Mxx, Mzz, Mxz ! moment tensor

**function KUPPER( A, X, X0 )**

**Description**

Single-lobed Kupper function for the moment rate function. Normalized by int\_0^\infty Kupper(t) dt = 1.

**Arguments**

real(PN) :: A ! Characteristic Time (s)

real(PN) :: X 　　! Time (s)

real(PN) :: X0 ! Origin Time (s)

**function IKUPPER( A, X, X0 )**

**Description**

Integrated single-lobed Kupper function for the moment function.

**Arguments**

real(PN) :: A ! Characteristic Time (s)

real(PN) :: X 　　! Time (s)

real(PN) :: X0 ! Origin Time (s)

**function HERRMAN( A, X, X0 )**

**Description**

Single-lobed Herrman function for the moment rate function. Normalized by int\_0^\infty Kupper(t) dt = 1 (Herrman(t) dt = 1).

**Arguments**

real(PN) :: A ! Characteristic Time (s)

real(PN) :: X 　　! Time (s)

real(PN) :: X0 ! Origin Time (s)

**function IHERRMAN( A, X, X0 )**

**Description**

Integrated single-lobed Herrman function for the moment function.

**Arguments**

real(PN) :: A ! Characteristic Time (s)

real(PN) :: X 　　! Time (s)

real(PN) :: X0 ! Origin Time (s)

### 3.1.9 module ppohFDM\_m\_stdlib

**Description**

This module defines units, precision and constants.

**Public Variables**

!-Precision Constant

integer, parameter, public :: DP = selected\_real\_kind(13) ! double precision

integer, parameter, public :: SP = selected\_real\_kind(5) ! single precision

integer, parameter, public :: PN = SP

integer, parameter, public :: PN = DP

!-In/Out

integer, parameter, public :: STDERR = 0

integer, parameter, public :: STDOUT = 6

integer, parameter, public :: STDIN = 5

!-Physical constants

real(PN), parameter, public :: PI = 3.141592653589793238462643383\_PN

real(PN), parameter, public :: R\_EARTH = 6371.0\_PN

real(PN), parameter, public :: DEG2RAD = PI/180.0\_PN

real(PN), parameter, public :: RAD2DEG = 180.0\_PN/PI

complex(PN), parameter, public :: EI = (0.0\_PN,1.0\_PN)

**Private Procedures**

public :: stdlib\_\_getio

public :: stdlib\_\_debug

public :: stdlib\_\_setdebug

public :: stdlib\_\_genfname

public :: stdlib\_\_countline

**subroutine stdlib\_\_getio(io, is\_big)**

**Description**

Returns the unused unit number measured from the io0 constant.

If you define is\_big=.true. then it searches for the unused unit number from IOBIG0.

**Example**

call ppohFDM\_stdlib\_\_getio( io )

call ppohFDM\_stdlib\_\_getio( io, .true. ) ! big\_endian

call ppohFDM\_stdlib\_\_getio( io, 300. ) ! search for unused unit number from 300

**subroutine stdlib\_\_genfname( base, ext, fname )**

**Description**

Generates a filename "base.????.ext" which does not exist in the current directory.

**Arguments**

character(len=\*), intent(in) :: base ! base filename

character(len=\*), intent(in) :: ext ! extension

character(len=\*), intent(out) :: fname ! output

**Example**

call ppohFDM\_GenFname( 'foo','dat', fname )

call ppohFDM\_getIO( fp )

open ( fp, file=fname ) ! fname = 'foo.0000.dat' if there is an identical name file

**subroutine stdlib\_\_countline( fp, n )**

**Description**

Counts line number n included in the file specified by fp

**Arguments**

integer, intent(in) :: fp

integer, intent(out) :: n

**subroutine stdlib\_\_setdebug( t\_or\_f )**

**Description**

Turns the debug mode on and off. The default is defined by DEBUG\_MODE.

**Arguments**

logical, intent(in) :: t\_or\_f

**subroutine getio\_0( io, io00 )**

**Description**

Returns the unused unit number measured from the io0 constant

**Arguments**

integer, intent(out) :: io ! unit number

integer, intent(in), optional :: io00 ! directive number

**Example**

call ppohFDM\_stdlib\_\_getIO( io ) ! find unused number

open ( io, file = fname )

**subroutine getio\_big( io, is\_big )**

**Description**

Returns the unused unit number measured from the io0 constant

If you specify is\_big=.true. then finds unused number (s) from IOBIG0.

**Example**

call ppohFDM\_stdlib\_\_getIO( io )

open ( io, file = fname )

**Arguments**

integer, intent(out) :: io ! unit number

logical, intent(in) :: is\_big ! .true. : BIG\_ENDIAN

**subroutine debug\_m( routine, msg )**

**Description**

Prints Message to standard output.

**Example**

Message( 'sub', 'message')

**Arguments**

character(\*), intent(in) :: routine

character(\*), intent(in) :: msg

**subroutine debug\_i( routine, msg, var )**

**Description**

Prints Message with an integer number to standard output.

**Example**

Message( 'sub', 'data 1 = ', a )

**Arguments**

character(\*), intent(in) :: routine

character(\*), intent(in) :: msg

integer , intent(in) :: var

**subroutine debug\_l( routine, msg, var )**

**Description**

Prints Message with a logical number to standard output.

**Example**

Message( 'sub', 'data 1 = ', a )

**Arguments**

character(\*), intent(in) :: routine

character(\*), intent(in) :: msg

logical , intent(in) :: var

**subroutine debug\_d( routine, msg, var )**

**Description**

Prints Message with a number in double precision to standard output.

**Example**

Message( 'sub', 'data 1 = ', a )

**Arguments**

character(\*), intent(in) :: routine

character(\*), intent(in) :: msg

real(DP) , intent(in) :: var

**subroutine debug\_f( routine, msg, var )**

**Description**

Prints Message with a number in single precision to standard output.

**Example**

Message( 'sub', 'data 1 = ', a )

**Arguments**

character(\*), intent(in) :: routine

character(\*), intent(in) :: msg

real(SP) , intent(in) :: var

**subroutine debug\_c( routine, msg, var )**

**Description**

Prints Message with a complex number to standard output.

**Example**

Message( 'sub', 'data 1 = ', a )

**Arguments**

character(\*), intent(in) :: routine

character(\*), intent(in) :: msg

complex(DP) , intent(in) :: var

**subroutine debug\_a( routine, msg, var )**

**Description**

Prints Message with a character to standard output.

**Example**

Message( 'sub', 'data 1 = ', a )

**Arguments**

character(\*), intent(in) :: routine

character(\*), intent(in) :: msg

character(\*), intent(in) :: var

### 3.1.10 module ppohFDM\_m\_surfbc

**Description**

This module applies free surface boundary conditions on the solid/air interface.

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_params, only : Nx1, Nx, Dx, Dz

use ppohFDM\_m\_comvar

use ppohFDM\_m\_medium, only : nifs, kfsz, ifsx, ifsz, is\_fs, & niob, kobz, iobx, iobz, is\_ob

**Public procedures**

public :: ppohFDM\_surfbc\_\_zerostress

public :: ppohFDM\_surfbc\_\_stressderiv

public :: ppohFDM\_surfbc\_\_velderiv

**subroutine ppohFDM\_surfbc\_\_zerostress()**

**Description**

Applies zero to the shear stress tensor.

**subroutine ppohFDM\_surfbc\_\_stressderiv()**

**Description**

Calculates spatial derivatives of the stress component on a free surface, assuming zero share stress.

**subroutine ppohFDM\_surfbc\_\_velderiv()**

**Description**

Calculates spatial derivatives of the velocity component on a free surface, assuming zero share stress.

### 3.1.11 module ppohFDM\_m\_swatch

**Description**

This module measures the computational time of each module.

**Dependency**

use ppohFDM\_m\_stdlib

**Public Procedures**

public :: ppohFDM\_swatch\_\_setup

public :: ppohFDM\_swatch\_\_setnm

public :: ppohFDM\_swatch\_\_on

public :: ppohFDM\_swatch\_\_off

public :: ppohFDM\_swatch\_\_report

**subroutine ppohFDM\_swatch\_\_setup( sw )**

**Description**

Initializes the elapsed time counter.

**Arguments**

logical, intent(in) :: sw

**subroutine ppohFDM\_swatch\_\_setnm( i, nm )**

**Description**

Sets a user-defined name to the i-th block.

**Arguments**

integer, intent(in) :: i ! block number

character(\*), intent(in) :: nm ! user defined name for the i-th block

**subroutine ppohFDM\_swatch\_\_on( i, nm )**

**Description**

Starts time measurement for the i-th block.

**Arguments**

integer, intent(in) :: i

character(\*), intent(in), optional :: nm

**subroutine ppohFDM\_swatch\_\_off( i )**

**Description**

Stops measurement for the i-th block.

Computation time from the previous call of swatch\_on is accumulated.

**Arguments**

integer, intent(in) :: i

**subroutine ppohFDM\_swatch\_\_report( io, ishead )**

**Description**

Prints total computation time and the relative occupation rate of each io.

If ishead = .true. then a two line header is printed first.

**Arguments**

integer, optional, intent(in) :: io

logical, optional, intent(in) :: ishead

### 3.1.12 module ppohFDM\_m\_avs

**Description**

This module converts the simulation results into AVS format

**Dependency**

use ppohFDM\_m\_stdlib

use ppohFDM\_m\_params

use ppohFDM\_m\_comvar

**subroutine ppohFDM\_fld\_output()**

**Description**

This subroutine output the AVS filed format.

**Arguments**

integer :: i

character(len=80) filename

**subroutine ppohFDM\_coord\_output()**

**Description**

Output X, Y coordinates

**Arguments**

integer :: i

character(len=80) filename

**subroutine ppohFDM\_data\_output()**

**Description**

The data subject holds any numerical data associated with each node in the mesh

**Arguments**

integer :: i

character(len=80) filename

## 3.2 seism3d3n

**Description**

seism3d3n.f90 is a Fortran program used for explicit parallel FDM simulation of elastic (seismic) wave propagation in heterogeneous elastic media in 3D due to a double-couple seismic source in the medium. This simulation code is constructed by the following modules and subroutines. The simulation is formed by a preparation part for initialization and parameter setting, a main part to repeat calculation with the march of time, and a post processing part for termination of the simulation. The parallel simulation is based on a domain partitioning procedure. The 3D simulation model uses a rectangular cartesian-coordinate system (x, y, z) and partial differentiation of the equation of motion can be calculated by choosing an ordinary (4th) and a lower- (2nd-) or higher(8-th)order of the FDM scheme.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

use ppohFDM\_io

use ppohFDM\_pssub

use ppohFDM\_pfd3d

use ppohFDM\_boundary

use ppohFDM\_stress

use ppohFDM\_velocity

use ppohFDM\_source

use ppohFDM\_sponge\_absorber

use mpi

use ppohFDM\_set\_condition

**MPI setup**

**Description**

Sets up MPI environment and an index array to examine the absolute x,y,z position (IA, JA, KA) in the full 3D simulation model from the rank number (myid) of each of the subregions

**Flow of calculation**

call set\_mpi\_environment( myid, itbl, idx, idy, idz )

! Absolute Coordinate

do I=NXP0, NXP1

IA( I ) = NXP\*idx + I

end do

do J=NYP0, NYP1

JA( J ) = NYP\*idy + J

end do

do K=0, NZP1

KA( K ) = NZP\*idz + K

end do

if( myid == 0 ) then

write(STDERR,'(A)') "PROGRAM SEISM3DZ"

write(STDERR,\*)

write(STDERR,'(A,I5,A,I5,A,I5)') "MODEL SIZE: ", NXP, "x", NYP, "x", NZP

end if

!! Initialize elapsed time counter

if( myid == NP/2 ) then

write(STDERR,\*)

call system\_clock( timcount, crate )

timcount0 = timcount

timprev = timcount

end if

**Initial setup**

**Description**

Sets up an absorbing boundary, a seismic source in moment tensor, a free surface boundary condition at the topographic surface, the physical parameters of the 3D elastic media. Also examines the stability condition(s) of the FDM simulation, and the position of stations (waveform recording point). The user can choose either Kupper- or Herrman-type source time functions for the seismic source.

**Flow of calculation**

!!--------------------------------------------------------------------------!!

!! ABSORBER !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_sponge\_absorber( )

!!--------------------------------------------------------------------------!!

!! SET SOURCE POINT !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_source()

!!--------------------------------------------------------------------------!!

!! Moment Tensor !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_sld2moment( STRIKE, DIP, RAKE, 1.0, RMXX, RMYY, RMZZ, RMXY, RMYZ, RMXZ )

!!--------------------------------------------------------------------------!!

!! FREE SURFACE !!

!!--------------------------------------------------------------------------!!

!! 1. Free surface boundary on the absolute grid

do J=0, NY+1

do I=0, NX+1

KFSZA(I,J) = KFS

end do

end do

!! 2. Trimming kfsza, detection of horizontal boundar

call ppohFDM\_set\_free\_surface( KFSZA, KFSZ, NIFS, NJFS, &

IFSX, IFSY, IFSZ, JFSX, JFSY, JFSZ )

!!--------------------------------------------------------------------------!!

!! SET MEDIUM PARAMETERS !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_medium( DEN, RIG, LAM )

!!--------------------------------------------------------------------------!!

!! STABLE CONDITION !!

!!--------------------------------------------------------------------------!!

if( myid == nproc-1 ) then

write(STDERR,'(A,F10.5)') "STABLE CONDITION (SHOULD BE SMALLER THAN 1)", &

DT/( 0.45\*min(DX,DY,DZ)/maxval(sqrt((LAM+2\*RIG)/DEN)))

if( DT > 0.45\*min(DX,DY,DZ)/maxval(sqrt((LAM+2\*RIG)/DEN)) ) then

write(STDERR,\*) "Enlarge Spatial Grid and/or Shorten Time Grid!"

end if

write(STDERR,\*)

end if

!!--------------------------------------------------------------------------!!

!! PARAMETER FILE OUTPUT !!

!!--------------------------------------------------------------------------!!

if( myid == 0 ) call ppohFDM\_output\_prm

!!--------------------------------------------------------------------------!!

!! MOMENT FUNCTION !!

!!--------------------------------------------------------------------------!!

!! Use ikupper/iherrman for Body Force Source, kupper/herrman for Stress Drop Source

do IT = 1, NTMAX

T = (IT-1)\*DT

STIME (IT) = kupper (AT, T, T0) ! Kupper function for Stress Drop

! STIME (IT) = herrman (AT, T, T0) ! Herrman function

end do

!!--------------------------------------------------------------------------!!

!! ZERO-FILL ARRAYS !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_initialize\_arrays()

!!--------------------------------------------------------------------------!!

!! STATION !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_station()

call ppohFDM\_station\_func()

!!--------------------------------------------------------------------------!!

!! OUTPUT FILES !!

!!--------------------------------------------------------------------------!!

!!Please see io.f90

call ppohFDM\_io\_open()

**main part**

**Description**

The FDM simulation is conducted explicitly to simulate seismic wave propagation spreading from a source in heterogeneous media and propagating in 3D media with time. At each time step calculation of the equation of motion with suitable boundary conditions at the interface is evaluated by calling the following subroutines. The snapshots of the seismic wavefield and waveforms are stored. For parallel computing an MPI is used to update the simulation results between neighboring processors. The user can choose a scheme for spatial differentiation (2-nd, 4-th and 8th-order FDM) by choosing ppohFDM\_pdiff\*3\_p2, ppohFDM\_pdiff\*3\_p4, or ppohFDM\_pdiff\*3\_p8 \*=x,y,z).

**Flow of calculation**

!!--------------------------------------------------------------------------!!

!! TIME STEP START !!

!!--------------------------------------------------------------------------!!

if( myid == NP/2 ) then

write(STDERR,\*)

call system\_clock( timcount, crate )

timprev = timcount

end if

ttotal = 0.0\_PN

DXI = 1.0\_PN / DX

DYI = 1.0\_PN / DY

DZI = 1.0\_PN / DZ

xmax = 0.0; ymax = 0.0; zmax = 0.0

timestep: do IT=1, NTMAX

T = DT \* (IT-1)

!!--- Time Measurement

if( mod(IT, NWRITE) == 0 ) then

! max value for debug output

call get\_max( xmax, ymax, zmax )

if( myid == NP/2 ) then

call system\_clock( timcount, crate )

tstep = real( timcount - timprev ) / real( crate )

ttotal = ttotal + tstep

etas = (ntmax -it)/ real(it) \* (timcount-timcount0)/real( crate )

etah = int( etas/( 60\*60) ); etas = etas - etah \*60\*60

etam = int( etas/( 60) ); etas = etas - etam \*60

etasi = int(etas)

timprev = timcount

write(STDERR,'(A,I6,A,I6,A,I2.2,A,I2.2,A,I2.2,A,F9.4,A,3ES9.2,A)') &

"IT=(", IT, "/",NTMAX,"), ETA=", &

etah,":",etam,":",etasi, &

", Time/Step=", ttotal / IT, "[s], MAX = ( ", &

xmax, ymax, zmax, " )"

end if

end if

!!-----------------------------------------------------------------------!!

!! Velocity t=(n+1/2)\*dt !!

!!-----------------------------------------------------------------------!!

if( is\_fs .or. is\_nearfs ) then

call ppohFDM\_bc\_zero\_stress( KFSZ,NIFS,NJFS,IFSX,IFSY,IFSZ,JFSX,JFSY,JFSZ )

end if

call ppohFDM\_pdiffx3\_p4( SXX,DXSXX, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DX )

call ppohFDM\_pdiffy3\_p4( SYY,DYSYY, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DY )

call ppohFDM\_pdiffx3\_m4( SXY,DXSXY, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DX )

call ppohFDM\_pdiffx3\_m4( SXZ,DXSXZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DX )

call ppohFDM\_pdiffy3\_m4( SXY,DYSXY, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DY )

call ppohFDM\_pdiffy3\_m4( SYZ,DYSYZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DY )

call ppohFDM\_pdiffz3\_p4( SZZ,DZSZZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DZ )

call ppohFDM\_pdiffz3\_m4( SXZ,DZSXZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DZ )

call ppohFDM\_pdiffz3\_m4( SYZ,DZSYZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DZ )

!! Substitute Reduced-order derivatives at around model boundary

call ppohFDM\_truncate\_diff\_stress(idx,idy,idz)

if( is\_fs .or. is\_nearfs ) then

call ppohFDM\_bc\_stress\_deriv( KFSZ,NIFS,NJFS,IFSX,IFSY,IFSZ,JFSX,JFSY,JFSZ )

end if

!! Velocity Update

call ppohFDM\_update\_vel ( 1, NXP, 1, NYP, 1, NZP )

call ppohFDM\_update\_vel\_sponge( 1, NXP, 1, NYP, 1, NZP )

!!-----------------------------------------------------------------------!!

!! BODY FORCE !!

!!-----------------------------------------------------------------------!!

! call ppohFDM\_source\_term\_bodyforce ! comment out if stress drop source

!!-----------------------------------------------------------------------!!

!! Message Passing !!

!!-----------------------------------------------------------------------!!

call ppohFDM\_passing\_velocity()

!!-----------------------------------------------------------------------!!

!! Stress t=(n+1)\*dt !!

!!-----------------------------------------------------------------------!!

call ppohFDM\_pdiffx3\_m4( VX,DXVX, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DX )

call ppohFDM\_pdiffy3\_m4( VY,DYVY, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DY )

call ppohFDM\_pdiffx3\_p4( VY,DXVY, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DX )

call ppohFDM\_pdiffx3\_p4( VZ,DXVZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DX )

call ppohFDM\_pdiffy3\_p4( VX,DYVX, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DY )

call ppohFDM\_pdiffy3\_p4( VZ,DYVZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DY )

call ppohFDM\_pdiffz3\_m4( VZ,DZVZ, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DZ )

call ppohFDM\_pdiffz3\_p4( VX,DZVX, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DZ )

call ppohFDM\_pdiffz3\_p4( VY,DZVY, NXP,NYP,NZP,NXP0,NXP1,NYP0,NYP1,NZP0,NZP1, DZ )

!! Substitute reduced order derivatives at around model boundary

call ppohFDM\_truncate\_diff\_vel(idx,idy,idz)

if( is\_fs .or. is\_nearfs ) then

call ppohFDM\_bc\_vel\_deriv( KFSZ,NIFS,NJFS,IFSX,IFSY,IFSZ,JFSX,JFSY,JFSZ )

end if

!!-- Update Stress Components

call ppohFDM\_update\_stress ( 1, NXP, 1, NYP, 1, NZP )

call ppohFDM\_update\_stress\_sponge ( 1, NXP, 1, NYP, 1, NZP )

!!-----------------------------------------------------------------------!!

!! STRESS DROP SOURCE !!

!!-----------------------------------------------------------------------!!

call ppohFDM\_source\_term\_stressdrop()

!!-----------------------------------------------------------------------!!

!! Message Passing !!

!!-----------------------------------------------------------------------!!

call ppohFDM\_passing\_stress()

!!-----------------------------------------------------------------------!!

!! SNAPSHOT DATA EXPORT !!

!!-----------------------------------------------------------------------!!

!!please see io.f90

call ppohFDM\_io\_write()

!!-----------------------------------------------------------------------!!

!! VOLUME RENDERING DATA EXPORT !!

!!-----------------------------------------------------------------------!!

! call ppohFDM\_io\_vol\_psdiff( it, IOVOL )

! call ppohFDM\_io\_vol\_dis( it, IOVOL )

end do timestep

**Finalize**

**Description**

The finalizing part of the FDM simulation terminates program run by closing files and stopping MPI processes.

**Flow of calculation**

if( is\_fs ) then

close( IOSPS )

close( IOSNP )

close( IOWAV )

end if

if( is\_ioxy ) close( ioxy )

if( is\_ioyz ) close( ioyz )

if( is\_ioxz ) close( ioxz )

if( myid == NP/2 ) then

call system\_clock( timcount, crate )

ttotal = ttotal + tstep

write(STDERR,'(A,I15,A)') &

"Finished Computation. Total Time = ", int(ttotal), " sec."

end if

call mpi\_finalize( ierr )

stop

### 3.2.1 module ppohFDM\_boundary

**Description**

This module applies a zero-stress boundary condition on a free surface. A zero stress value is applied to stress components (S\_pz, p=x,y,z) and the results of spatial derivatives just above and below the free-surface boundary are recalculated by using a one-side differentiation scheme. This scheme can treat an irregular boundary as well as a flat boundary.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

**subroutine ppohFDM\_bc\_zero\_stress( KFSZ, NIFS, NJFS, IFSX, IFSY, IFSZ, JFSX, JFSY, JFSZ)**

**Description**

Applies a zero stress value to the stress components (S\_pz, p=x,y,z) on a free surface.

**Arguments**

integer, intent(in) :: KFSZ(NXP0:NXP1,NYP0:NYP1) ! depth of the free surface

integer, intent(in) :: NIFS, NJFS ! number of points in x and y directions to examine free surface conditions

integer, intent(in) :: IFSX(NFSMAX), IFSY(NFSMAX), IFSZ(NFSMAX) ! evaluation points of the free surface condition in x,y,z

integer, intent(in) :: JFSX(NFSMAX), JFSY(NFSMAX), JFSZ(NFSMAX)

**subroutine ppohFDM\_bc\_stress\_deriv( KFSZ, NIFS, NJFS, IFSX, IFSY, IFSZ, JFSX, JFSY, JFSZ)**

**Description**

One-side differentiation scheme used for calculating spatial derivatives of the stress components just below and above a free surface.

**Arguments**

integer, intent(in) :: KFSZ(NXP0:NXP1,NYP0:NYP1)

integer, intent(in) :: NIFS, NJFS

integer, intent(in) :: IFSX(NFSMAX), IFSY(NFSMAX), IFSZ(NFSMAX)

integer, intent(in) :: JFSX(NFSMAX), JFSY(NFSMAX), JFSZ(NFSMAX)

**subroutine ppohFDM\_bc\_vel\_deriv( KFSZ, NIFS, NJFS, IFSX, IFSY, IFSZ, JFSX, JFSY, JFSZ)**

**Description**

One-side differentiation scheme used for calculating spatial derivatives of the velocity components just below and above a free surface.

**Arguments**

integer, intent(in) :: KFSZ(NXP0:NXP1,NYP0:NYP1)

integer, intent(in) :: NIFS, NJFS

integer, intent(in) :: IFSX(NFSMAX), IFSY(NFSMAX), IFSZ(NFSMAX)

integer, intent(in) :: JFSX(NFSMAX), JFSY(NFSMAX), JFSZ(NFSMAX)

### 3.2.2 module ppohFDM\_io

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

use ppohFDM\_pssub

**subroutine ppohFDM\_io\_write\_wav( it, io )**

**Description**

Exports a waveform trace of velocity and displacements at stations at time step (it) to file number (io).

**Arguments**

integer, intent(in) :: it ! time step

integer, intent(in) :: io ! i/o file number

**Arrays**

real(PN) :: VXALL(NSTMAX,NTMAX1), VYALL(NSTMAX,NTMAX1), VZALL(NSTMAX,NTMAX1) ! three-component velocity waveform

real(PN) :: UXALL(NSTMAX,NTMAX1), UYALL(NSTMAX,NTMAX1), UZALL(NSTMAX,NTMAX1) ! three-component displacement waveform

**subroutine ppohFDM\_io\_open\_snp( io, fname, n1, n2, flag )**

**Description**

Creates a file (fname) for exporting snapshots of a seismic wavefield with a size of w\*h grid points.

**Arguments**

integer, intent(in) :: io ! i/o equipment number

character(\*), intent(in) :: fname ! file name of file to be exported

integer, intent(in) :: n1 ! file size (w)

integer, intent(in) :: n2 ! file size (h)

logical, intent(in) :: flag ! open file if flag=.true.

**subroutine ppohFDM\_io\_write\_sur( it, io )**

**Description**

Exports snapshots of a seismic wavefield of a free surface at time step (it) to file number (io).

**Arguments**

integer, intent(in) :: it

integer, intent(in) :: io

**subroutine ppohFDM\_io\_write\_xy( it, io )**

**Description**

Exports snapshots of a seismic wavefield of a user-specified horizontal (x-y) plane at time step (it) to file number (io).

**Arguments**

integer, intent(in) :: it

integer, intent(in) :: io

**subroutine ppohFDM\_io\_write\_xz( it, io )**

**Description**

Exports snapshots of a seismic wavefield of a user-specified vertical (x-z) plane at time step (it) to file number (io).

**Arguments**

integer, intent(in) :: it

integer, intent(in) :: io

**subroutine ppohFDM\_io\_write\_yz( it, io )**

**Description**

Exports snapshots of a seismic wavefield of a user-specified vertical (y-z) plane at time step (it) to file number (io).

**Arguments**

integer, intent(in) :: it

integer, intent(in) :: io

**subroutine ppohFDM\_output\_prm**

**Description**

Outputs the parameter file used in the present simulation to the .prm file.

**Parameters**

integer, parameter :: io = 86

### 3.2.3 module ppohFDM\_param

**Description**

Declares all of the parameters used to specify the FDM simulation model.

**Dependency**

use ppohFDM\_stdio

**Parameters**

--<< Execute Title >>

character(99), parameter :: TITLE="SEISM3D3" ! Title of Simulation

-- << Model Size and Grid Width >>

integer, parameter :: NX = 256 ! Number of grid points in X,Y,Z

integer, parameter :: NY = 256

integer, parameter :: NZ = 256

integer, parameter :: KFS = 25 ! Free Surface Grid

integer, parameter :: NX1 = NX+1

integer, parameter :: NY1 = NY+1

integer, parameter :: NZ1 = NZ+1

integer, parameter :: NTMAX = 2000 ! Number of time steps

integer, parameter :: NWRITE = 10 ! Interval of snapshots

real(PN), parameter :: DX = 0.5\_PN ! Grid size in X,Y,Z

real(PN), parameter :: DY = 0.5\_PN

real(PN), parameter :: DZ = 0.5\_PN

real(PN), parameter :: DT = 0.025\_PN ! Time increment (s)

integer, parameter :: NDUMP = 5 ! Volume data dumping

**Arrays**

!-- << Medium Parameters >>

real(PN) :: DEN(0:NXP1,0:NYP1,0:NZP1) ! Density

real(PN) :: RIG(0:NXP1,0:NYP1,0:NZP1) ! Rigidity

real(PN) :: LAM(0:NXP1,0:NYP1,0:NZP1) ! Lame's coefficient

real(PN) :: QP (0:NXP1,0:NYP1,0:NZP1) ! P intrinsic Q

real(PN) :: QS (0:NXP1,0:NYP1,0:NZP1) ! S intrinsic Q

real(PN) :: TU (0:NXP1,0:NYP1,0:NZP1) ! Relaxiation time

!-- << Independent Variables >>

real(PN) :: SXX(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Six-components Stress

real(PN) :: SYY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SZZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SXY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SYZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SXZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Three-components Velocity

real(PN) :: VX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: VY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: VZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

!-- << Derivatives wrt Space >>

real(PN) :: DXVX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Derivatives of Velocity

real(PN) :: DXVY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DXVZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYVX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYVY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYVZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZVX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZVY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZVZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DXSXX(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Derivatives of Stress

real(PN) :: DXSXY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DXSXZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYSYY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYSXY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYSYZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZSZZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZSXZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZSYZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

### 3.2.4 module ppohFDM\_pfd3d

**Description**

Finite difference engines for parallel 3D FDM simulation of seismic wave propagation.

**Dependency**

uses ppohFDM\_stdio.

**Procedures**

public :: ppohFDM\_pdiffx3\_p4

public :: ppohFDM\_pdiffx3\_m4

public :: ppohFDM\_pdiffy3\_p4

public :: ppohFDM\_pdiffy3\_m4

public :: ppohFDM\_pdiffz3\_p4

public :: ppohFDM\_pdiffz3\_m4

public :: ppohFDM\_pdiffz3\_p2

public :: ppohFDM\_pdiffz3\_m2

public :: ppohFDM\_pdiffz3\_p2

public :: ppohFDM\_pdiffy3\_m2

public :: ppohFDM\_pdiffx3\_p2

public :: ppohFDM\_pdiffx3\_m2

public :: ppohFDM\_pdiffz3\_p8

public :: ppohFDM\_pdiffz3\_m8

public :: ppohFDM\_pdiffy3\_p8

public :: ppohFDM\_pdiffy3\_m8

public :: ppohFDM\_pdiffx3\_p8

public :: ppohFDM\_pdiffx3\_m8

**Constants for finite-difference calculation**

real(PN), parameter :: C20 = 1.0\_PN ! constants for 2nd-order FDM calculation

real(PN), parameter :: C40 = 1.125\_PN ! constants for 4th-order FDM calculation

real(PN), parameter :: C41 = 1.0\_PN / 24.0\_PN

real(PN), parameter :: C80 = 1.19628906E+00! constants for 8th-order FDM calculation

real(PN), parameter :: C81 = 7.97526017E-02

real(PN), parameter :: C82 = 9.57031269E-03

real(PN), parameter :: C83 = 6.97544659E-04

**subroutine ppohFDM\_pdiffx3\_p4( V, DXV, NX, NY, NZ, NX0, NX1, NY0, NY1, NZ0, NZ1, DX )**

**Description**

4th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the x direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1) ! target variables

real(PN), intent(out) :: DXV (NX0:NX1,NY0:NY1,NZ0:NZ1) ! spatial derivatives

integer, intent(in) :: NX, NY, NZ ! number of grid points in X,Y,Z

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1 ! boundary of array

real(PN), intent(in) :: DX ! grid size in the X direction

**subroutine ppohFDM\_pdiffx3\_m4( V, DXV, NX, NY, NZ, NX0, NX1, NY0, NY1, NZ0, NZ1, DX )**

**Description**

4th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the x direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DXV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX, NY, NZ

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DX

**subroutine ppohFDM\_pdiffy3\_p4( V, DYV, NX, NY, NZ, NX0, NX1, NY0, NY1, NZ0, NZ1, DY )**

**Description**

4th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the y direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DYV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX, NY, NZ

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DY

**subroutine ppohFDM\_pdiffy3\_m4( V, DYV, NX, NY, NZ, NX0, NX1, NY0, NY1, NZ0, NZ1, DY )**

**Description**

4th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the y direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DYV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX, NY, NZ

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DY

**subroutine ppohFDM\_pdiffz3\_p4( V, DZV, NX, NY, NZ, NX0, NX1, NY0, NY1, NZ0, NZ1, DZ )**

**Description**

4th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the z direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DZV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX, NY, NZ

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DZ

**subroutine ppohFDM\_pdiffz3\_m4( V, DZV, NX, NY, NZ, NX0, NX1, NY0, NY1, NZ0, NZ1, DZ )**

**Description**

4th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the z direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DZV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX, NY, NZ

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DZ

**subroutine ppohFDM\_pdiffz3\_p2 ( V, DZV, NX0,NX1,NY0,NY1,NZ0,NZ1, DZ )**

**Description**

2nd-order Finite difference calculation for calculating spatial derivatives of variables with respect to the z direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DZV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DZ

**subroutine ppohFDM\_ pdiffz3\_p2 ( V, DZV, NX0,NX1,NY0,NY1,NZ0,NZ1, DZ )**

**Description**

2nd-order Finite difference calculation for calculating spatial derivatives of variables with respect to the z direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DZV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DZ

**subroutine ppohFDM\_ pdiffy3\_p2 ( V, DYV, NX0,NX1,NY0,NY1,NZ0,NZ1, DY )**

**Description**

2nd-order Finite difference calculation for calculating spatial derivatives of variables with respect to the y direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DYV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DY

**subroutine ppohFDM\_ pdiffy3\_m2 ( V, DYV, NX0,NX1,NY0,NY1,NZ0,NZ1, DY )**

**Description**

2nd-order Finite difference calculation for calculating spatial derivatives of variables with respect to the y direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DYV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DY

**subroutine ppohFDM\_ pdiffx3\_p2 ( V, DXV, NX0,NX1,NY0,NY1,NZ0,NZ1, DX )**

**Description**

2nd-order Finite difference calculation for calculating spatial derivatives of variables with respect to the x direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DXV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DX

**subroutine ppohFDM\_ pdiffx3\_m2 ( V, DXV, NX0,NX1,NY0,NY1,NZ0,NZ1, DX )**

**Description**

2nd-order Finite difference calculation for calculating spatial derivatives of variables with respect to the x direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DXV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DX

**subroutine ppohFDM\_ pdiffz3\_p8 ( V, DZV, NX0,NX1,NY0,NY1,NZ0,NZ1, DZ )**

**Description**

8th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the z direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DZV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DZ

**subroutine ppohFDM\_ pdiffz3\_m8 ( V, DZV, NX0,NX1,NY0,NY1,NZ0,NZ1, DZ )**

**Description**

8th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the z direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DZV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DZ

**subroutine ppohFDM\_ pdiffy3\_p8 ( V, DYV, NX0,NX1,NY0,NY1,NZ0,NZ1, DY )**

**Description**

8th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the y direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DYV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DY

**subroutine ppohFDM\_ pdiffy3\_p8 ( V, DYV, NX0,NX1,NY0,NY1,NZ0,NZ1, DY )**

**Description**

8th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the y direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DYV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DY

**subroutine ppohFDM\_ pdiffx3\_p8 ( V, DXV, NX0,NX1,NY0,NY1,NZ0,NZ1, DX )**

**Description**

8th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the x direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DXV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DX

**subroutine ppohFDM\_ pdiffx3\_m8 ( V, DXV, NX0,NX1,NY0,NY1,NZ0,NZ1, DX )**

**Description**

8th-order Finite difference calculation for calculating spatial derivatives of variables with respect to the x direction.

**Arguments**

real(PN), intent(in) :: V (NX0:NX1,NY0:NY1,NZ0:NZ1)

real(PN), intent(out) :: DXV (NX0:NX1,NY0:NY1,NZ0:NZ1)

integer, intent(in) :: NX0, NX1, NY0, NY1, NZ0, NZ1

real(PN), intent(in) :: DX

### 3.2.5 module ppohFDM\_ppsub

**Description**

Subroutines for FDM simulation of seismic wave propagation, such as for setting a seismic source time function.

**Dependency**

use ppohFDM\_stdio

**function kupper( A, X, X0 )**

**Description**

Single-lobed Kupper function for the moment rate function.

Normalized by int\_0^\infty Kupper(t) dt = 1.

**Arguments**

real(PN) :: A ! Characteristic Time

real(PN) :: X ! Time

real(PN) :: X0 ! Origin Time

real(PN) :: kupper ! Herrman function for the moment rate (??)

**function ikupper ( A, X, X0 )**

**Description**

Integrates a single-lobed Kupper function for the moment function.

**Arguments**

real(PN) :: A

real(PN) :: X

real(PN) :: X0

real(PN) :: ikupper

**function herrmann ( A, X, X0 )**

**Description**

Herrmann's function for the moment rate function

**Arguments**

real(PN), intent(in) :: A

real(PN), intent(in) :: X

real(PN), intent(in) :: X0

**function iherrmann ( A, X, X0 )**

**Description**

Integration of Herrmann's function for the moment function (See above ??)

**Arguments**

real(PN), intent(in) :: A

real(PN), intent(in) :: X

real(PN), intent(in) :: X0

**subroutine ppohFDM\_sld2moment( strike, dip, rake, M0, Mxx, Myy, Mzz, Mxy, Myz, Mxz )**

Description

Calculates six moment tensor components from strike, dip, and rake [deg] under the double-couple source assumptions.

Arguments

real(PN), intent(in) :: strike, dip, rake ! fault mechanism parameters

real(PN), intent(in) :: M0 ! seismic moment

real(PN), intent(out) :: Mxx, Myy, Mzz ! moment tensor (6 component)

real(PN), intent(out) :: Mxy, Myz, Mxz

**subroutine ppohFDM\_clear2d(nx0, nx1, nz0, nz1, a, value)**

**Description**

Clears a 2D array by setting a special value.

**Arguments**

integer, intent(in) :: nx0, nx1

integer, intent(in) :: nz0, nz1

real(PN), intent(out) :: a( nx0:nx1, nz0:nz1 ) ! array to be cleared

real(PN), intent(in) :: value

**subroutine ppohFDM\_clear3d( nx0, nx1, ny0, ny1, nz0, nz1, a, value )**

**Description**

Clears a 3D array by setting a special value.

**Arguments**

integer, intent(in) :: nx0, nx1

integer, intent(in) :: ny0, ny1

integer, intent(in) :: nz0, nz1

real(PN), intent(out) :: a(nx0:nx1,ny0:ny1,nz0:nz1)

real(PN), intent(in) :: value

### 3.2.6 module ppohFDM\_set\_condition

**Descriptions**

This module sets up the medium parameters of subsurface structure, source, and station locations in the simulation model.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

**subroutine ppohFDM\_set\_medium\_param()**

**Descriptions**

Sets up the test medium for the subsurface structure.

**subroutine ppohFDM\_set\_medium( DEN, RIG, LAM )**

**Descriptions**

Sets up elastic parameters (density, rigidity, and lam.) in the subsurface structure.

**Arguments**

filename="source.dat"

**subroutine ppohFDM\_set\_source()**

**Descriptions**

Sets up the source location and mechanism.

**Arguments**

real :: Xtmp, Ytmp, Ztmp

**subroutine ppohFDM\_set\_station()**

**Descriptions**

Sets up the number of stations and their locations.

**Arguments**

real :: Xtmp, Ytmp, Ztmp

### 3.2.7 module ppohFDM\_source

**Description**

Moment-tensor point source implementation based on either the stress-drop formulation and using body force.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

**subroutine ppohFDM\_source\_term\_stressdrop**

**Description**

Stress drop source implementation according to the stress drop formulation

**subroutine ppohFDM\_source\_term\_bodyforce ()**

**Description**

Equivalent body force implementation by using body force.

### 3.2.8 module ppohFDM\_sponge\_absorber

**Description**

A sponge-type absorbing buffer to suppress reflection of a signal from artificial boundaries in the bounded domain.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

**subroutine ppohFDM\_set\_sponge\_absorber( )**

**Description**

Sets absorption parameters for absorbing a buffer zone which is multiplied by stress and velocity components at each time step to weaken signals in the absorbing zone. See Cerjan, et al. (1995) for more details.

**Parameters**

real(PN), parameter:: ALPHA = 0.015\_PN ! constants of absorber (fixed)

integer:: NXP, NYP, NZP ! size of absorbing buffer zone (>=20)

### 3.2.9 module ppohFDM\_stdio

**Description**

Standard I/O, constants and Precision Definition. A "big\_endian" binary format is used throughout this simulation.

**Parameters**

integer, parameter :: DP = selected\_real\_kind(13)

integer, parameter :: SP = selected\_real\_kind(5)

integer, parameter :: PN = SP ! Precision Number

integer, parameter :: STDERR = 0

integer, parameter :: STDIN = 5

integer, parameter :: STDOUT = 6

!-<< BIG ENDIAN NUMBER >>

!assumes the following environmental variables in local machines

"setenv F\_UFMTENDIAN 900-999"

integer, parameter :: IOBIG1 = 901

integer, parameter :: IOBIG2 = 902

integer, parameter :: IOBIG3 = 903

integer, parameter :: IOBIG4 = 904

integer, parameter :: IOBIG5 = 905

integer, parameter :: IOBIG6 = 906

integer, parameter :: IOBIG7 = 907

real(PN), parameter :: PI = 3.14159265358979323846264338327950288419\_PN ! pi

real(PN), parameter :: DEG2RAD = PI / 180.\_PN ! coefficient used to convert degrees to radians

real(PN), parameter :: RAD2DEG = 180.0\_PN / PI ! coefficient used to convert radians to degrees

### 3.2.10 module ppohFDM\_stress

**Description**

This module supports updating stress variables with time by inter-processor communication using MPI. Also contains utilities for recalculating the stress derivative around the boundary and updating the stress tensor in the sponge absorber zone.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

use mpi

**subroutine ppohFDM\_truncate\_diff\_stress(idx,idy,idz)**

**Description**

Substitute the derivatives at around the boundary of the reduced-order derivatives.

**Arguments**

integer, intent(in) :: idx, idy, idz ! mpi rank in x,y, and z.

**subroutine ppohFDM\_update\_stress( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the stress tensor.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the area in the X-loop

integer, intent(in) :: NY00, NY01 ! index for updating the area in the Y-loop

integer, intent(in) :: NZ00, NZ01 ! index for updating the area in the Z-loop

**subroutine ppohFDM\_update\_stress\_sponge( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the stress tensor in the sponge buffer zone.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the sponge area in the X-loop

integer, intent(in) :: NY00, NY01 ! Y-loop

integer, intent(in) :: NZ00, NZ01 ! Z-loop

**subroutine ppohFDM\_passing\_stress()**

**Description**

Data buffering & passing for stress tensor and sending/receiving data between neighboring processors using MPI\_ISEND(), MPI\_IRECV()

### 3.2.11 module ppohFDM\_velocity

**Description**

This module supports updating the velocity variables with time by inter-processor communication using MPI. Also contains utilities for recalculating the velocity variables in the sponge absorber zone.

**Declarations**

use ppohFDM\_stdio

use ppohFDM\_param

use mpi

**subroutine ppohFDM\_passing\_velocity ()**

**Description**

Data buffering & passing for the velocity vector to neighboring processors using MPI\_ISEND()/MPI\_IRECV().

**subroutine ppohFDM\_update\_vel( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the velocity vector.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the area in the X-loop

integer, intent(in) :: NY00, NY01 ! Y-loop

integer, intent(in) :: NZ00, NZ01 ! Z-loop

**subroutine ppohFDM\_update\_vel\_sponge( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the velocity vector in the sponge buffer zone.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the sponge area in the X-loop

integer, intent(in) :: NY00, NY01 ! Y-loop

integer, intent(in) :: NZ00, NZ01 ! Z-loop

### 3.2.12 module ppohFDM\_set\_condition (real)

**Descriptions**

This module sets up the medium parameters of subsurface structure, source, and station locations in the simulation model of inhomogeneous layer structure.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

**subroutine ppohFDM\_set\_medium\_param()**

**Descriptions**

Sets up the test medium for the subsurface structure.

**Arguments**

filename="medium\_real.dat"

**subroutine ppohFDM\_read\_geom\_file\_name()**

**Descriptions**

Reading geometry list.

**Arguments**

filename="geometry\_real.dat"

**subroutine ppohFDM\_read\_geom\_data()**

**Descriptions**

Sets up the test medium for the inhomogeneous layer structure

**subroutine ppohFDM\_set\_medium( DEN, RIG, LAM )**

**Descriptions**

Sets up elastic parameters (density, rigidity, and lam.) in the subsurface structure.

**Arguments**

filename="source.dat"

**subroutine ppohFDM\_set\_source()**

**Descriptions**

Sets up the source location and mechanism.

**Arguments**

real :: Xtmp, Ytmp, Ztmp

**subroutine ppohFDM\_set\_station()**

**Descriptions**

Sets up the number of stations and their locations.

**Arguments**

real :: Xtmp, Ytmp, Ztmp

### 3.2.13 module ppohFDM\_param (real)

**Description**

Declares all of the parameters used to specify the FDM simulation model of inhomogeneous layer structure.

**Dependency**

use ppohFDM\_stdio

**Parameters**

--<< Execute Title >>

character(99), parameter :: TITLE="SEISM3D3" ! Title of Simulation

-- << Model Size and Grid Width >>

integer, parameter :: NX = 200 ! Number of grid points in X,Y,Z

integer, parameter :: NY = 200

integer, parameter :: NZ = 200

integer, parameter :: KFS = 25 ! Free Surface Grid

integer, parameter :: NX1 = NX+1

integer, parameter :: NY1 = NY+1

integer, parameter :: NZ1 = NZ+1

integer, parameter :: NTMAX = 2000 ! Number of time steps

integer, parameter :: NWRITE = 100 ! Interval of snapshots

real(PN), parameter :: DX = 0.5\_PN ! Grid size in X,Y,Z

real(PN), parameter :: DY = 0.5\_PN

real(PN), parameter :: DZ = 0.5\_PN

real(PN), parameter :: DT = 0.025\_PN ! Time increment (s)

integer, parameter :: NDUMP = 5 ! Volume data dumping

**Arrays**

!-- << Medium Parameters >>

real(PN) :: DEN(0:NXP1,0:NYP1,0:NZP1) ! Density

real(PN) :: RIG(0:NXP1,0:NYP1,0:NZP1) ! Rigidity

real(PN) :: LAM(0:NXP1,0:NYP1,0:NZP1) ! Lame's coefficient

real(PN) :: QP (0:NXP1,0:NYP1,0:NZP1) ! P intrinsic Q

real(PN) :: QS (0:NXP1,0:NYP1,0:NZP1) ! S intrinsic Q

real(PN) :: TU (0:NXP1,0:NYP1,0:NZP1) ! Relaxiation time

!-- << Independent Variables >>

real(PN) :: SXX(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Six-components Stress

real(PN) :: SYY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SZZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SXY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SYZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: SXZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Three-components Velocity

real(PN) :: VX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: VY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: VZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

!-- << Derivatives wrt Space >>

real(PN) :: DXVX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Derivatives of Velocity

real(PN) :: DXVY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DXVZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYVX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYVY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYVZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZVX (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZVY (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZVZ (NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DXSXX(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1) ! Derivatives of Stress

real(PN) :: DXSXY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DXSXZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYSYY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYSXY(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DYSYZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZSZZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZSXZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

real(PN) :: DZSYZ(NXP0:NXP1,NYP0:NYP1,NZP0:NZP1)

! inhomogeneous layer structure (sample is 9 layers)

real(PN) :: ZDEP(23,0:201,0:201), R01(100), VP1(100), VS1(100)

integer :: kza(24,0:201,0:201)

## 3.3 decrease B/F ratio (Seism3d3n)

**Description**

we propose to effectively decrease the required B/F ratio of the FDM simulation of seismic wave propagation. In the original simulation code, we first calculated each spatial derivative of the velocity and stress components and stored them in memory. These derivatives were then used for the kernels of update-stress and update-velocity, requiring large B/F ratios in order to load and store the large number of variables. To overcome this challenge, we modified the FDM code to merge the derivative and update calculations, thereby avoiding the need to store and load variables during the calculation. As a result, the required B/F ratios for both the update-velocity and the update-stress kernels dropped dramatically, to 0.4

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

use ppohFDM\_io

use ppohFDM\_pssub

use ppohFDM\_boundary

use ppohFDM\_stress

use ppohFDM\_velocity

use ppohFDM\_source

use ppohFDM\_sponge\_absorber

use mpi

use ppohFDM\_set\_condition

**MPI setup**

**Description**

Sets up MPI environment and an index array to examine the absolute x,y,z position (IA, JA, KA) in the full 3D simulation model from the rank number (myid) of each of the subregions

**Flow of calculation**

call set\_mpi\_environment( myid, itbl, idx, idy, idz )

! Absolute Coordinate

do I=NXP0, NXP1

IA( I ) = NXP\*idx + I

end do

do J=NYP0, NYP1

JA( J ) = NYP\*idy + J

end do

do K=0, NZP1

KA( K ) = NZP\*idz + K

end do

if( myid == 0 ) then

write(STDERR,'(A)') "PROGRAM SEISM3DZ"

write(STDERR,\*)

write(STDERR,'(A,I5,A,I5,A,I5)') "MODEL SIZE: ", NXP, "x", NYP, "x", NZP

end if

!! Initialize elapsed time counter

if( myid == NP/2 ) then

write(STDERR,\*)

call system\_clock( timcount, crate )

timcount0 = timcount

timprev = timcount

end if

**Initial setup**

**Description**

Sets up an absorbing boundary, a seismic source in moment tensor, a free surface boundary condition at the topographic surface, the physical parameters of the 3D elastic media. Also examines the stability condition(s) of the FDM simulation, and the position of stations (waveform recording point). The user can choose either Kupper- or Herrman-type source time functions for the seismic source.

**Flow of calculation**

!!--------------------------------------------------------------------------!!

!! ABSORBER !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_sponge\_absorber( )

!!--------------------------------------------------------------------------!!

!! SET SOURCE POINT !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_source()

!!--------------------------------------------------------------------------!!

!! Moment Tensor !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_sld2moment( STRIKE, DIP, RAKE, 1.0, RMXX, RMYY, RMZZ, RMXY, RMYZ, RMXZ )

!!--------------------------------------------------------------------------!!

!! FREE SURFACE !!

!!--------------------------------------------------------------------------!!

!! 1. Free surface boundary on the absolute grid

do J=0, NY+1

do I=0, NX+1

KFSZA(I,J) = KFS

end do

end do

!! 2. Trimming kfsza, detection of horizontal boundar

call ppohFDM\_set\_free\_surface( KFSZA, KFSZ, NIFS, NJFS, &

IFSX, IFSY, IFSZ, JFSX, JFSY, JFSZ )

!!--------------------------------------------------------------------------!!

!! SET MEDIUM PARAMETERS !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_medium( DEN, RIG, LAM )

!!--------------------------------------------------------------------------!!

!! STABLE CONDITION !!

!!--------------------------------------------------------------------------!!

if( myid == nproc-1 ) then

write(STDERR,'(A,F10.5)') "STABLE CONDITION (SHOULD BE SMALLER THAN 1)", &

DT/( 0.45\*min(DX,DY,DZ)/maxval(sqrt((LAM+2\*RIG)/DEN)))

if( DT > 0.45\*min(DX,DY,DZ)/maxval(sqrt((LAM+2\*RIG)/DEN)) ) then

write(STDERR,\*) "Enlarge Spatial Grid and/or Shorten Time Grid!"

end if

write(STDERR,\*)

end if

!!--------------------------------------------------------------------------!!

!! PARAMETER FILE OUTPUT !!

!!--------------------------------------------------------------------------!!

if( myid == 0 ) call ppohFDM\_output\_prm

!!--------------------------------------------------------------------------!!

!! MOMENT FUNCTION !!

!!--------------------------------------------------------------------------!!

!! Use ikupper/iherrman for Body Force Source, kupper/herrman for Stress Drop Source

do IT = 1, NTMAX

T = (IT-1)\*DT

STIME (IT) = kupper (AT, T, T0) ! Kupper function for Stress Drop

! STIME (IT) = herrman (AT, T, T0) ! Herrman function

end do

!!--------------------------------------------------------------------------!!

!! ZERO-FILL ARRAYS !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_initialize\_arrays()

!!--------------------------------------------------------------------------!!

!! STATION !!

!!--------------------------------------------------------------------------!!

call ppohFDM\_set\_station()

call ppohFDM\_station\_func()

!!--------------------------------------------------------------------------!!

!! OUTPUT FILES !!

!!--------------------------------------------------------------------------!!

!!Please see io.f90

call ppohFDM\_io\_open()

**main part**

**Description**

The FDM simulation is conducted explicitly to simulate seismic wave propagation spreading from a source in heterogeneous media and propagating in 3D media with time. At each time step calculation of the equation of motion with suitable boundary conditions at the interface is evaluated by calling the following subroutines. The snapshots of the seismic wavefield and waveforms are stored. For parallel computing an MPI is used to update the simulation results between neighboring processors. The user can choose a scheme for spatial differentiation (2-nd, 4-th and 8th-order FDM) by choosing ppohFDM\_pdiff\*3\_p2, ppohFDM\_pdiff\*3\_p4, or ppohFDM\_pdiff\*3\_p8 \*=x,y,z).

**Flow of calculation**

!!--------------------------------------------------------------------------!!

!! TIME STEP START !!

!!--------------------------------------------------------------------------!!

if( myid == NP/2 ) then

write(STDERR,\*)

call system\_clock( timcount, crate )

timprev = timcount

end if

ttotal = 0.0\_PN

DXI = 1.0\_PN / DX

DYI = 1.0\_PN / DY

DZI = 1.0\_PN / DZ

xmax = 0.0; ymax = 0.0; zmax = 0.0

timestep: do IT=1, NTMAX

T = DT \* (IT-1)

!!--- Time Measurement

if( mod(IT, NWRITE) == 0 ) then

! max value for debug output

call get\_max( xmax, ymax, zmax )

if( myid == NP/2 ) then

call system\_clock( timcount, crate )

tstep = real( timcount - timprev ) / real( crate )

ttotal = ttotal + tstep

etas = (ntmax -it)/ real(it) \* (timcount-timcount0)/real( crate )

etah = int( etas/( 60\*60) ); etas = etas - etah \*60\*60

etam = int( etas/( 60) ); etas = etas - etam \*60

etasi = int(etas)

timprev = timcount

write(STDERR,'(A,I6,A,I6,A,I2.2,A,I2.2,A,I2.2,A,F9.4,A,3ES9.2,A)') &

"IT=(", IT, "/",NTMAX,"), ETA=", &

etah,":",etam,":",etasi, &

", Time/Step=", ttotal / IT, "[s], MAX = ( ", &

xmax, ymax, zmax, " )"

end if

end if

!!-----------------------------------------------------------------------!!

!! Velocity t=(n+1/2)\*dt !!

!!-----------------------------------------------------------------------!!

if( is\_fs .or. is\_nearfs ) then

call ppohFDM\_bc\_zero\_stress( KFSZ,NIFS,NJFS,IFSX,IFSY,IFSZ,JFSX,JFSY,JFSZ )

end if

!! Velocity Update

call ppohFDM\_update\_vel ( 1, NXP, 1, NYP, 1, NZP )

call ppohFDM\_update\_vel\_sponge( 1, NXP, 1, NYP, 1, NZP )

!!-----------------------------------------------------------------------!!

!! BODY FORCE !!

!!-----------------------------------------------------------------------!!

! call ppohFDM\_source\_term\_bodyforce ! comment out if stress drop source

!!-----------------------------------------------------------------------!!

!! Message Passing !!

!!-----------------------------------------------------------------------!!

call ppohFDM\_passing\_velocity()

!!-----------------------------------------------------------------------!!

!! Stress t=(n+1)\*dt !!

!!-----------------------------------------------------------------------!!

!!-- Update Stress Components

call ppohFDM\_update\_stress ( 1, NXP, 1, NYP, 1, NZP )

call ppohFDM\_update\_stress\_sponge ( 1, NXP, 1, NYP, 1, NZP )

!!-----------------------------------------------------------------------!!

!! STRESS DROP SOURCE !!

!!-----------------------------------------------------------------------!!

call ppohFDM\_source\_term\_stressdrop()

!!-----------------------------------------------------------------------!!

!! Message Passing !!

!!-----------------------------------------------------------------------!!

call ppohFDM\_passing\_stress()

!!-----------------------------------------------------------------------!!

!! SNAPSHOT DATA EXPORT !!

!!-----------------------------------------------------------------------!!

!!please see io.f90

call ppohFDM\_io\_write()

!!-----------------------------------------------------------------------!!

!! VOLUME RENDERING DATA EXPORT !!

!!-----------------------------------------------------------------------!!

! call ppohFDM\_io\_vol\_psdiff( it, IOVOL )

! call ppohFDM\_io\_vol\_dis( it, IOVOL )

end do timestep

**Finalize**

**Description**

The finalizing part of the FDM simulation terminates program run by closing files and stopping MPI processes.

**Flow of calculation**

if( is\_fs ) then

close( IOSPS )

close( IOSNP )

close( IOWAV )

end if

if( is\_ioxy ) close( ioxy )

if( is\_ioyz ) close( ioyz )

if( is\_ioxz ) close( ioxz )

if( myid == NP/2 ) then

call system\_clock( timcount, crate )

ttotal = ttotal + tstep

write(STDERR,'(A,I15,A)') &

"Finished Computation. Total Time = ", int(ttotal), " sec."

end if

call mpi\_finalize( ierr )

stop

### 3.3.1 module ppohFDM\_stress

**Description**

This module supports updating stress variables with time by inter-processor communication using MPI. Also contains utilities for recalculating the stress derivative around the boundary and updating the stress tensor in the sponge absorber zone. This module combines the module of spatial derivative in subroutine part of ppohFDM\_update\_stress.

**Dependency**

use ppohFDM\_stdio

use ppohFDM\_param

use mpi

**subroutine ppohFDM\_truncate\_diff\_stress(idx,idy,idz)**

**Description**

Substitute the derivatives at around the boundary of the reduced-order derivatives.

**Arguments**

integer, intent(in) :: idx, idy, idz ! mpi rank in x,y, and z.

**subroutine ppohFDM\_update\_stress( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the stress tensor.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the area in the X-loop

integer, intent(in) :: NY00, NY01 ! index for updating the area in the Y-loop

integer, intent(in) :: NZ00, NZ01 ! index for updating the area in the Z-loop

integer :: i, j, k, k\_j

real(PN) :: RL1, RM1, RM2, RLRM2

real(PN) :: DXVX1, DYVY1, DZVZ1, D3V3

real(PN) :: DXVYDYVX1, DXVZDZVX1, DYVZDZVY1

real(PN) :: DXVX0,DYVX0,DZVX0,DXVY0,DYVY0,DZVY0,DXVZ0,DYVZ0,DZVZ0

real(PN), parameter :: C40 = 1.125\_PN

real(PN), parameter :: C41 = 1.0\_PN / 24.0\_PN

**subroutine ppohFDM\_update\_stress\_sponge( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the stress tensor in the sponge buffer zone.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the sponge area in the X-loop

integer, intent(in) :: NY00, NY01 ! Y-loop

integer, intent(in) :: NZ00, NZ01 ! Z-loop

**subroutine ppohFDM\_passing\_stress()**

**Description**

Data buffering & passing for stress tensor and sending/receiving data between neighboring processors using MPI\_ISEND(), MPI\_IRECV()

### 3.3.2 module ppohFDM\_velocity

**Description**

This module supports updating the velocity variables with time by inter-processor communication using MPI. Also contains utilities for recalculating the velocity variables in the sponge absorber zone. This module combines the module of spatial derivative in subroutine part of ppohFDM\_update\_velocity.

**Declarations**

use ppohFDM\_stdio

use ppohFDM\_param

use mpi

**subroutine ppohFDM\_passing\_velocity ()**

**Description**

Data buffering & passing for the velocity vector to neighboring processors using MPI\_ISEND()/MPI\_IRECV().

**subroutine ppohFDM\_update\_vel( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the velocity vector.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the area in the X-loop

integer, intent(in) :: NY00, NY01 ! Y-loop

integer, intent(in) :: NZ00, NZ01 ! Z-loop

integer :: i, j, k

integer :: k\_j

real(PN) :: ROX, ROY, ROZ

real(PN), parameter :: C40 = 1.125\_PN

real(PN), parameter :: C41 = 1.0\_PN / 24.0\_PN

real(PN) :: DXSXX0,DXSXY0,DXSXZ0

real(PN) :: DYSXY0,DYSYY0,DYSYZ0

real(PN) :: DZSXZ0,DZSYZ0,DZSZZ0

**subroutine ppohFDM\_update\_vel\_sponge( NX00, NX01, NY00, NY01, NZ00, NZ01 )**

**Description**

Updates the velocity vector in the sponge buffer zone.

**Arguments**

integer, intent(in) :: NX00, NX01 ! index for updating the sponge area in the X-loop

integer, intent(in) :: NY00, NY01 ! Y-loop

integer, intent(in) :: NZ00, NZ01 ! Z-loop