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/AMR-FDM

ver. 0.3.0

User's Guide

Contents

1. Introduction

1.1 Overview of ppOpen-APPL/AMR-FDM

ppOpen-APPL/AMR-FDM is a set of an adaptive mesh refinement (AMR) framework including a dynamic load balancing technique for finite-difference method (FDM) simulations. This library includes two main features. One is the AMR technique, and the other is the dynamic domain decomposition (DDD) technique for dynamic load-balancing. Although both the explicit and implicit time-marching methods are supported in ppOpen-APPL/AMR-FDM, the current version (ver.0.3.0) does not include linear solvers for the implicit method. The explicit advection equation solver by constrained interpolation profile (CIP) scheme is provided as a function of "advAMR3D" code developed on ppOpen-APPL/AMR-FDM in the ver. 0.3.0 archives. ppOpen-APPL/AMR-FDM is written in Fortran 90 with MPI and OpenMP. The subroutines and functions of ppOpen-APPL/AMR-FDM can be called from simulation code written in Fortran 90. Both flat-MPI and OpenMP/MPI hybrid programming models are available.

AMR is one of the common approaches for computational savings. In common FDM codes using the AMR technique, computational grids with different spacing are dynamically created in hierarchical layers in accordance with the local conditions of the phenomena. Fine grids suitable to the local domain that need high resolution are applied only there, and other regions are simulated by using moderate size grids (in Fig.1). Therefore, increments to the numerical cost owing to the localized region are reduced if the AMR technique is adopted. Generally, the implementation of an AMR simulation code is cumbersome procedure because the data structure is complicated by the use of hierarchical layers. The ppOpen-APPL/AMR-FDM aims easy-to-use tools for FDM simulations introducing a cell-based AMR technique.

To realize high-performance computation in FDM simulations using parallel processors, simulation codes are generally parallelized by adopting a domain decomposition method. The entire computational domain in a simulation is divided into sub-domains, and each sub-domain corresponds to a process in the domain decomposition parallelization. In AMR simulations with parallelization, hierarchical grid layers are dynamically created or deleted in each sub-domain. Then, the amount of computational load corresponding to each process becomes different, and the load balancing between processes cannot be guaranteed throughout the simulation run. Such a load imbalance leads to low efficiency of parallel computation. In AMR simulations, the dynamic load balancing technique is very important for high-performance computation. To overcome the problem of load imbalance in parallelized AMR simulations, in the ppOpen-APPL/AMR-FDM framework a dynamic domain decomposition (DDD) technique with which the whole computational domain is dynamically re-decomposed into new sub-domains so that the computational load on each process becomes nearly the same (in Fig.2) is implemented.

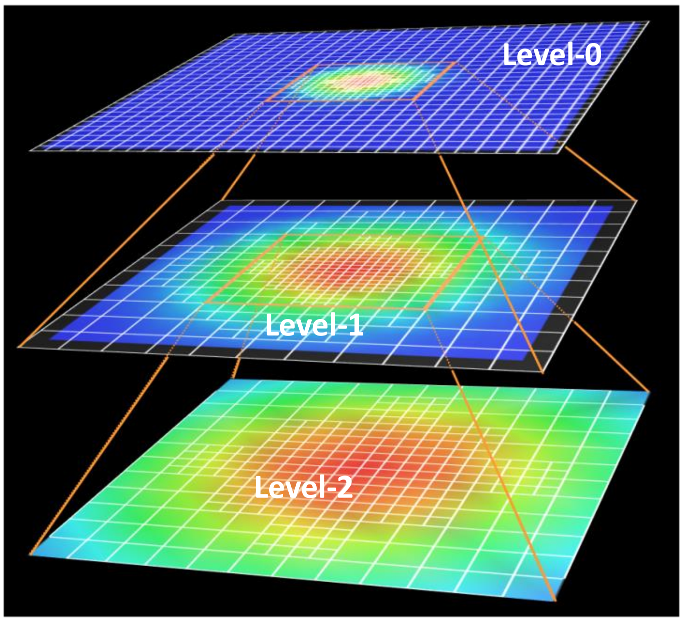


Figure 1 The conceptual diagram of adaptive mesh refinement

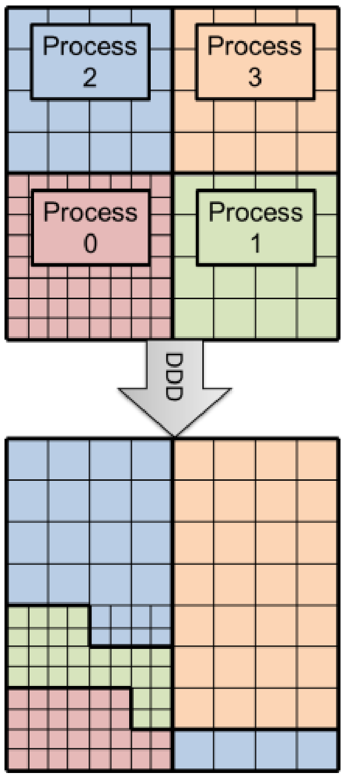


Figure 2 The conceptual diagram of dynamic domain decomposition

1.2 Main Algorithm Flow

Figure 3 shows the main algorithm flowchart of the ppOpen-APPL/AMR-FDM. The simulation code is divided into three main parts in the main loop: the AMR part, kernel part, and DDD part. First, in the AMR part, a computational domain is refined in accordance with refinement criteria that decide where or when to refine or unrefine a mesh. The criteria are arranged in the initial settings in advance and should be specified depending on each kernel, working condition and intended physical parameters. In addition, the different criteria for refinement and unrefinement can be arranged. Second, in the kernel part, a main kernel is calculated on each hierarchical layer fixed in the AMR part. The kernel part is not included in the ppOpen-APPL/AMR-FDM. The user can utilize any FDM kernel if at all possible. Finally, in the DDD part, all the computational costs on the computational domain are calculated, and then, the computational domain is redecomposed into new sub-domains corresponding to each process when the calculated computational costs exceed the load-balance criteria.

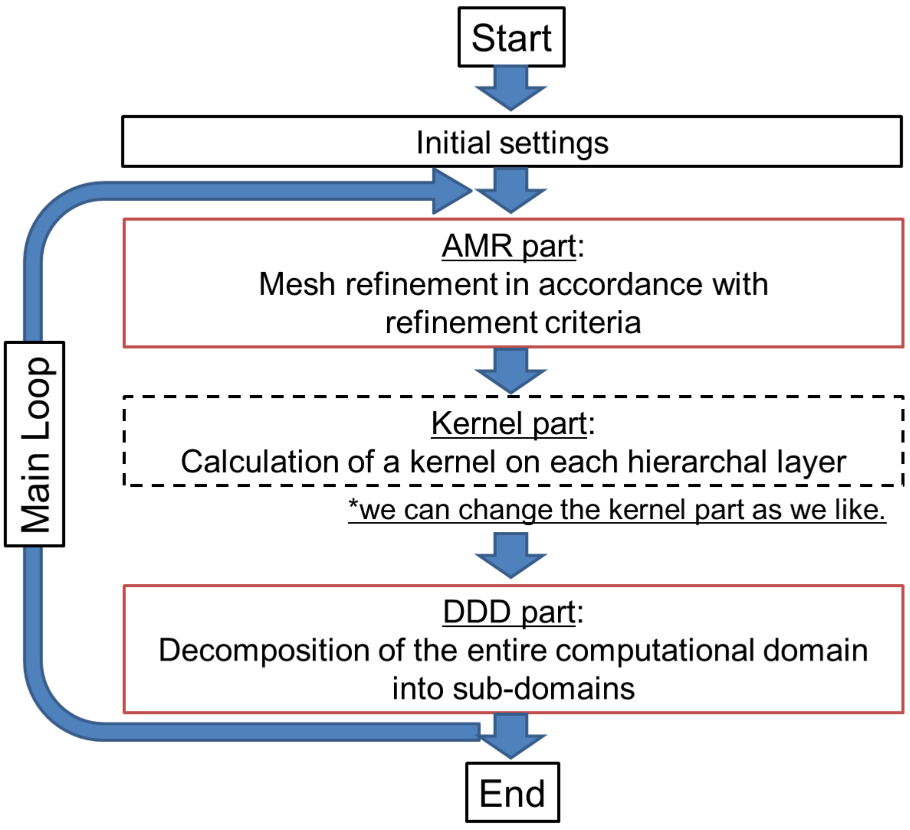

Start
Initial settings
AMR part:
Mesh refinement in accordance with refinement criteria
Kernel part:
Calculation of a kernel on each hierarchal layer
*we can change the kernel part as we like.
DDD part:
Decomposition of the entire computational domain into sub-domains
Main Loop
End

Figure 3 The Algorithm flowchart of the ppOpen-APPL/AMR-FDM

1.3 Data Structure and AMR Procedure

In the AMR technique used in ppOpen-APPL/AMR-FDM, in accordance with refinement criteria (thresholds), the spatial resolution is adjusted dynamically and locally to resolve complex physical phenomena. Although the AMR technique is a powerful method to reduce computational cost, a problem arises in that it causes the grid arrangement to become complex and non-uniform. Therefore, a fully threaded tree (FTT) data structure with multiple resolution levels is constructed in our AMR framework. FTT is one of the common approaches to deal with AMR grids efficiently. In general, there are two main approaches (namely, cell-based and block-based) for AMR simulations. In the ppOpen-APPL/AMR-FDM, a cell-based AMR technique is adopted and implemented. The basic concept of the FTT data structure used in the ppOpen-APPL/AMR-FDM is shown in Fig.4. In the region where high spatial resolution is required, an additional spatial grid system (Level N+1 shown in the figure) is locally created with half the size of a cell used in the upper level (Level N). When the higher resolution becomes unnecessary in the simulation run, the field information obtained in Level N+1 will be stored back to Level N, and the Level N+1 grid system will be automatically eliminated. Each cell consisting of one level of a spatial grid system has pointers that indicate neighbor, parent, and child cells. This subdivision of a grid system level takes place recursively until the spatial resolution meets the local refinement criteria. In addition, not only the grid interval Δx but also the time interval Δt is refined in the ppOpen-APPL/AMR-FDM.

In order to apply this framework to various FDM simulations, computations are organized not on interface-by-interface, but cell-by-cell basis. Figure 5 shows an example of the AMR procedure used in the ppOpen-APPL/AMR-FDM. When cells of Lv.N meet refinement criteria, the cells including the neighboring cells are refined, and new cells of Lv.N+1 are made (1 – 3 in Fig.5). After that, the interpolation of physical variables from Lv.N to Lv.N+1 is performed (4 in Fig.5). Finally, additional cells of Lv.N+1 are made in both side edges of the substantial cells, and the interpolation are also performed (5 – 6 in Fig.5). Here the purple cells indicate overlap cells. These cells exist in the background and are only referenced as boundary conditions on each hierarchical layer. Therefore the physical variables on the overlap cells are not updated in the kernel part shown in Fig.3. These AMR procedures are organized by some refinement flags that are stored in cell structures shown in Fig.4. Similarly, the unrefinement procedure are also performed by these flags. On the other hand, in the AMR framework, the time interval Δt is also refined as mentioned above. Therefore time integration of the framework is different from a time-stepping strategy used in conventional FDM applications. Figure 6 shows the schematic view of the computational sequence for time integration used in the ppOpen-APPL/AMR-FDM. The time integration is performed in serial order from fine to coarse grid layer, and physical variables are synchronized between each hierarchical layer at a specific time as shown in this figure. In the synchronization processing, interpolation from Lv.N to Lv.N+1 and average from Lv.N+1 to Lv.N are executed on each overlap cell shown in Fig.5.

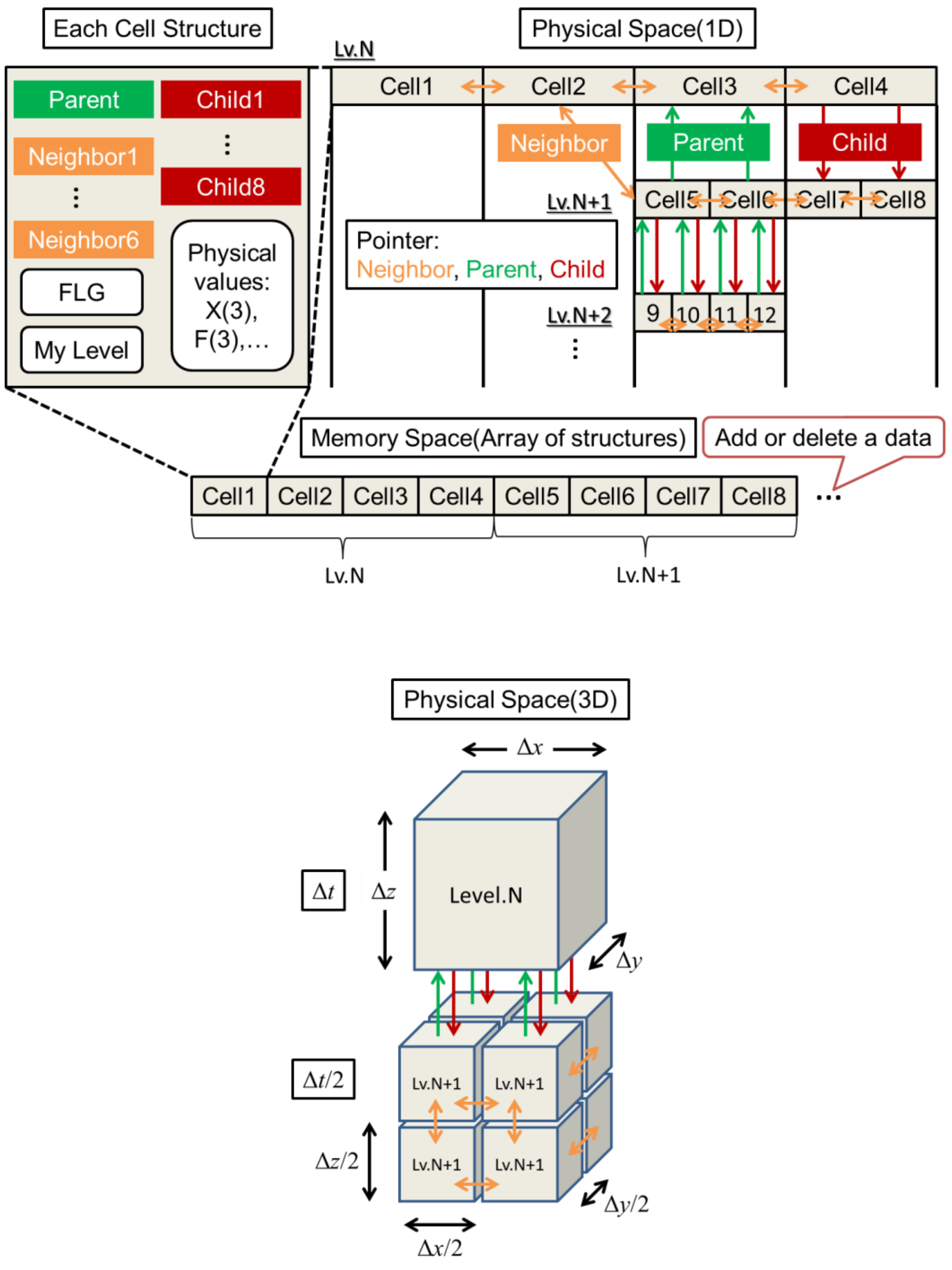


Figure 4 Fully threaded tree data structure used in the ppOpen-APPL/AMR-FDM. The computational domain is constructed by cell (grid) structures, and each cell is connected by pointers (neighbor, parent, and child) in a physical space. These structures are included in an array of structures in memory space. The time step interval of Lv.N+1 becomes half that of the parent level (Lv.N) according to the grid spacing, which also becomes half.

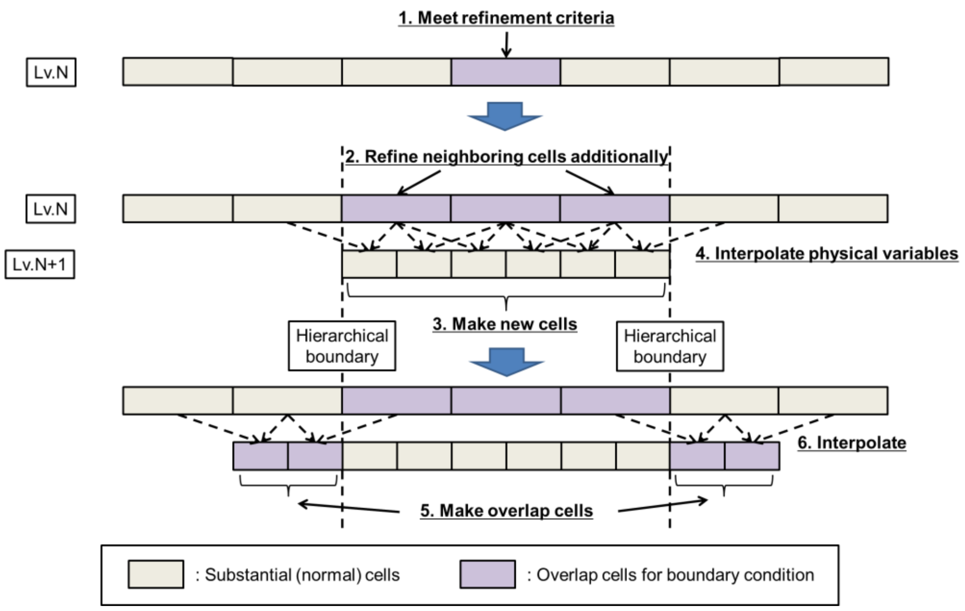

1. Meet refinement criteria
Lv.N
2. Refine neighboring cells additionally
Lv.N
Lv.N+1
4. Interpolate physical variables
3. Make new cells
Hierarchical boundary
Hierarchical boundary
6. Interpolate
5. Make overlap cells
: Substantial (normal) cells
: Overlap cells for boundary condition

Figure 5 An example of the AMR procedure used in the ppOpen-APPL/AMR-FDM. The kernel part is calculated on cell-by-cell basis. The AMR procedure is executed in serial order as shown in the figure. Here the purple cells indicate overlap cells that are only used for boundary conditions on each hierarchical layer.

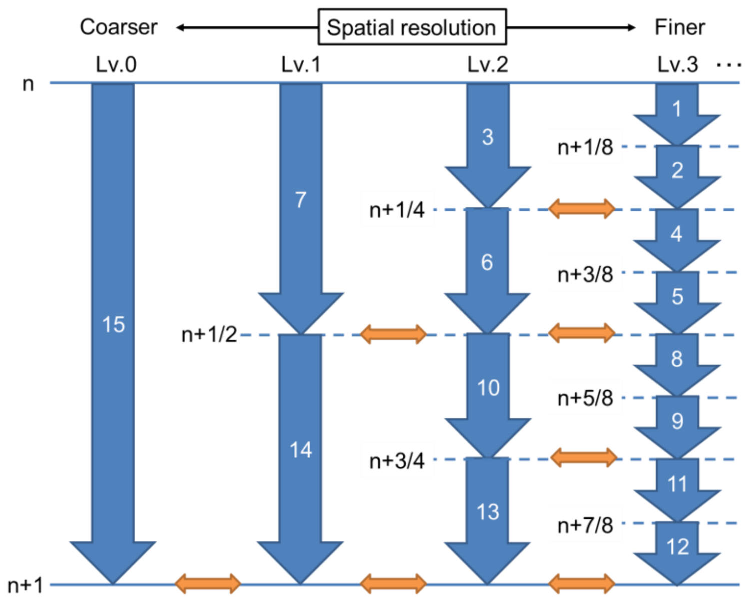


Figure 6 Schematic view of the computational sequence for time integration used in the ppOpen-APPL/AMR-FDM. The time integration on each hierarchical layer in the kernel part is performed in serial order as shown in the figure. Here n indicates time index (n corresponds to t, and n+1 corresponds to t + Δt), and the orange arrows indicate the timings of synchronization processing between each hierarchical layer.

1.4 Implementations of Dynamic Domain Decomposition

To decompose the entire computational domain into sub-domains so that the load balance becomes constant between processes, all the cells in the domain are numbered according to a space-filling curve in which neighboring cells are closely ranked. In the ppOpen-APPL/AMR-FDM, the Morton ordered curve is used as a space-filling curve, although any other curve can be used. In this method, from a three-dimensional grid (i, j, k), we extract each binary index with the order as (k, j, i, k, j, i, …), and a new binary number is generated. A locality of reference associated with a memory access is improved by using such a space-filling curve. In the ppOpen-APPL/AMR-FDM, the number of computational grid points and processes is restricted to 2^{3N} (N = 1,2,3,…) although there exists more flexible method for complicated geometries. Cells are numbered or ordered along one dimension according to the newly generated number. Then we decompose the cells by dividing the order by the number of processors. In numbering the curve, the computational cost per grid is considered.

In the hierarchical system of the AMR framework, the computational cost per grid increases to twice that of the parent level in association with a mesh refinement because the time step interval becomes half that of the parent level, $\Delta t \rightarrow \Delta t/2$, according to the grid spacing, which becomes $\Delta x \rightarrow \Delta x/2$. That is, the kernel part on a hierarchical layer of Lv.N+1 must be calculated twice per main loop (shown in Fig.6) compared with that of Lv.N owing to its half time step interval. Figure 7 shows an example of the DDD procedure. In the AMR framework, numbering by a space-filling curve is only performed on the Lv.0 layer to simplify the DDD procedure. The computational cost per grid on each layer is summed on the Lv.0 grids, and domain decomposition is performed based on the Lv.0 layer. This method has the advantage of a simple calculation associated with a DDD procedure, although the load imbalance is only partially corrected. If the refinement level increasingly becomes deeper, correction of the load imbalance may be difficult with this method. It has both merits and demerits.

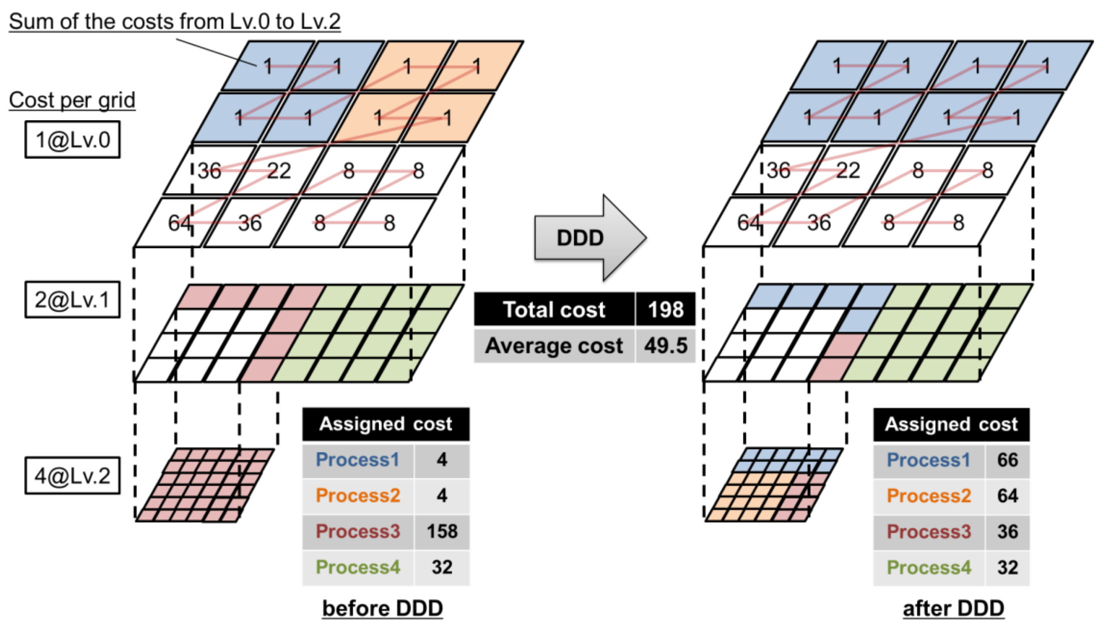


Figure 7 An example of the DDD procedure. The left figure indicates assigned costs and processes where the load balancing between each process is biased. The computational cost per grid on each hierarchical layer is summed on each Lv.0 grid, and then, the total cost and average cost per process of the whole domain are obtained. All the grids on the Lv.0 domain are numbered according to a space-filling curve, and the costs are summed until they exceed the average cost. Finally, a new sub-domain is determined by repeating this pattern, and the load imbalance is partially corrected with DDD.

2. Installation and Quick Start

2.1 ppohAMRFDM_0.3.0.tar

The “ppohAMRFDM_0.3.0.tar” archive includes the following:

- Source code files of “ppOpen-APPL/AMR-FDM ver. 0.3.0”
- Source code files of “advAMR3D”, which is a solver for 3D linear advection equations with explicit time-marching, developed on ppOpen-APPL/AMR-FDM
- Source code files of “post_vtk”, which is a visualization tool for advAMR3D
- Sample Makefiles
- Sample data files for advAMR3D

2.2 Structure of Directories

The "ppohAMRFDM_0.3.0.tar" archive includes the following directories. `$(CUR)` denotes the directory where the "ppohAMRFDM_0.3.0.tar" archive is unpacked.

Name of Directory	**Contents**
`$(CUR)/src`	source code files of ppOpen-APPL/AMR-FDM
`$(CUR)/examples/advAMR3D/src`	source code files of advAMR3D
`$(CUR)/examples/advAMR3D/run`	sample data set of advAMR3D (control data)
`$(CUR)/examples/advAMR3D/run/results`	sample results of advAMR3D
`$(CUR)/examples/advAMR3D/post_vtk`	source code of post_vtk, which is code for visualization of the results by advAMR3D
`$(CUR)/include`	directory that stores created module files
`$(CUR)/lib`	directory that stores created libraries
`$(CUR)/bin`	directory that stores created exec. files
`$(CUR)/doc`	documents
`$(CUR)/etc`	examples of 'Makefile.in'

2.3 Quick Start

(1) Preparation

- Fortran90 compilers (Operations have been confirmed with Intel and Fujitsu compilers)
- MPI Library
- OpenMP must be supported if you want to develop OpenMP/MPI hybrid code

(2) Modify 'Makefile.in'

Samples of 'Makefile.in' are found in `$(CUR)/etc`

Examples of 'Makefile.in'	Descriptions

`$(CUR)/etc/Makefile.in.fx10.flatmpi`
`$(CUR)/etc/Makefile.in.intel.flatmpi`
	Flat MPI for Fujitsu compiler
	Flat MPI for Intel compiler

`$(CUR)/etc/Makefile.in.fx10.hybrid`
`$(CUR)/etc/Makefile.in.intel.hybrid`
	OpenMP/MPI Hybrid for Fujitsu compiler
	OpenMP/MPI Hybrid for Intel compiler

Options in 'Makefile.in'	Descriptions
`$(MPIf90)`	Fortran 90 with MPI
`$(f90)`	Fortran 90 for single core
`$(pFFLAGS)`	Compiler options for Optimizations for ppOpen-APPL/AMR-FDM and advAMR3D
`$(sFFLAGS)`	Compiler options for Optimizations for post_vtk

`$(PREFIX)/include`
`$(PREFIX)/lib`
`$(PREFIX)/lib`
	directory that holds installed module files
	directory that holds installed libraries
	directory that holds installed exec. files

***** NOTICE ***: $(PREFIX) directory must be specified as ABSOLUTE/FULL path.**

(3) Compile/Install ppOpen-APPL/AMR-FDM

Operations	File created (library, module files, exec. file)

`$> cd $(CUR)/`
`$> make clean`	

`$> make`	`$(CUR)/lib/ppohAMRFDM.a`
	`$(CUR)/include/m_ppohamrfdm_util.mod`

`$> make install`	`$(PREFIX)/lib/ppohAMRFDM.a`
	`$(PREFIX)/include/m_ppohamrfdm_util.mod`

(4) Compile/Install advAMR3D and post_vtk

Operations	File created (library, module files, exec. file)

`$> cd $(CUR)/`
`$> make advAMR3D_clean`	

`$> make advAMR3D`	`$(CUR)/bin/advAMR3D`
	`$(CUR)/bin/post_vtk`

`$> make bin_install`	`$(PREFIX)/bin/advAMR3D`
	`$(PREFIX)/bin/post_vtk`

***** NOTICE ***: Processes (4) must be done after (3).**

(5) Running the code

(advAMR3D)

```
$> cd $(CUR)/examples/advAMR3D/run
$> mpirun -np 8 <$PREFIX>/bin/hybNS
```

with appropriate thread number for OpenMP (or corresponding operations)

(post_vtk)

```
$> cd $(CUR)/examples/advAMR3D/post_vtk
$> ./post_vtk
```

with single core operation only

(6) Clean/Uninstall

```
$> cd $(CUR)/
$> make clean       Clean files
$> make uninstall   Delete all installed files and Directories
```

3. ppOpen-APPL/AMR-FDM

3.1 Modules

m_ppohAMRFDM_util

This module contains information on variables for control, meshes and communications.

Parameters

`ppohAMRFDM_kint`	I	= 4 (in 'ppohAMRFDM_precision.inc')
`ppohAMRFDM_kdbl`	I	= 8 (in 'ppohAMRFDM_precision.inc')
`ppohAMRFDM_nlen`	I	= 80 (in 'ppohAMRFDM_precision.inc')

st_ppohAMRFDM_param

`ixmax`	I	Number of grid points in x direction @Lv.0 layer
`iymax`	I	Number of grid points in y direction @Lv.0 layer
`izmax`	I	Number of grid points in z direction @Lv.0 layer
`pxmax`	I	Number of process in x direction @initial Lv.0 layer
`pymax`	I	Number of process in y direction @initial Lv.0 layer
`pzmax`	I	Number of process in z direction @initial Lv.0 layer
`iomp0`	I	Number of OpenMP Thread
`LvMax`	I	Maximum Level
`initLv`	I	Initial Lv
`DDD, DDDflag`	I	Flag for function of dynamic domain decomposition
`npy`	I	Number of physical variables for used in the simulation
`nbf`	I	Size of overlap region between sub-domain corresponding to MPI parallelization
`nlv`	I	Size of overlap region between Lv layer
`it`	I	Index of main loop iteration
`itmax`	I	Maximum index value of main loop iteration
`dx`	RA	Spatial interval in each direction @Lv.0
`dt`	R	Time interval @Lv.0
`crtr_r0`	R	Criterion for mesh refinement
`crtr_d0`	R	Criterion for mesh unrefinement
`DDD_crtr`	R	Criterion for dynamic domain decomposition
`crtr_r`	R	Criteria for mesh refinement in each Lv layer
`crtr_d`	R	Criteria for mesh unrefinement in each Lv layer

nfg	I	Maximum value of flag using the AMR procedure
nint2	I	Conversion value of spatial index
itorder, lvorder	IA	Iteration order of AMR and kernel part in main loop
nID	I	Number of active grid in each Lv layer
nloop	I	Computational cost in each sub-domain corresponding to MPI parallelization

st_ppohAMRFDM_octset

octN	I	Index number of 'Mesh' array
octLv	I	Lv of an oct
Csort	I	Index of child oct with respect to parent oct
iFLG	IA	Flags for mesh refinement of an oct
iPOS	IA	Spatial index of an oct
MrtN	I	Morton number of an oct
bndry	I	Flag for boundary of a computational domain
F	RA	Physical variables of an oct
C	RA	Temporal variables of an oct
load	I	Computational cost of an oct
octPrt	TP	Pointer indicating a parent oct
octNb1, octNb2, octNb3, octNb4, octNb5, octNb6	TP	Pointer indicating neighbor octs (-x, +x, -y, +y, -z, and +z)
octCh1, octCh2, octCh3, octCh4, octCh5, octCh6, octCh7, octCh8	TP	Pointer indicating child octs
Psort	TP	Working pointer

st_ppohAMRFDM_comm_info

comm	I	MPI communicator
nprocs	I	Total number of MPI process
rank	I	Rank of MPI process
Mn2CPU	IA	Conversion array from Morton number to index number of 'Mesh' array
nrproc	IA	Number of oct associated with MPI communication
ngproc	IA	Number of overlap oct associated with MPI communication
iMesh	IA	Index array of 'Mesh' associated with MPI communication
MnDisp	I	Conversion value from index of 'Mesh' array to Morton number of an oct
MinMn	I	Minimum Morton number in a sub-domain corresponding to

		MPI process
MaxMn	I	Maximum Morton number in a sub-domain corresponding to MPI process
snum	IA	Number of oct sending to other processes in DDD procedure
rnum	IA	Number of oct receiving from other process in DDD procedure

st_ppohAMRFDM_sendbuf

bufi	IA	Send buffer array for MPI communication
bufr	RA	Send buffer array for MPI communication

st_ppohAMRFDM_recvbuf

bufi	IA	Receive buffer array for MPI communication
bufr	RA	Receive buffer array for MPI communication

st_ppohAMRFDM_calc_sequence

IDsq	IA	Active index array of 'Mesh' array

st_ppohAMRFDM_meshset

Mesh	TP	Substance of mesh (oct) information
GMesh	TP	Overlap mesh information between each sub-domain
Mesh2, GMesh2	TP	Working array for sort of 'Mesh' and 'GMesh'
sBuf	TA	Array for sending buffer in each process
rBuf	TA	Array for receiving buffer in each process
getID	TA	Array for Active index of 'Mesh' array
MinID, MaxID	IA	Maximum and minimum index number of 'Mesh' array
rfnoct1, rfnoct2, rfnoct3, rfnoct4	IA	Working array for OpenMP parallelization
rfncnt	I	Working variable for order of mesh refinement

3.2 Subroutines (API) called by user

ppohAMRFDM_init

This subroutine initializes MPI processes.

Parameters

`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`mpi_init, mpi_comm_dup, mpi_comm_size, mpi_comm_rank`

ppohAMRFDM_finalize

This subroutine terminates MPI processes.

Parameters

None

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`mpi_finalize`

ppohAMRFDM_barrier

This subroutine synchronizes MPI processes.

Parameters

`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

`ppohAMRFDM_pre`

Calls the following subroutines/functions

`mpi_barrier`

ppohAMRFDM_abort

This subroutine force-quit MPI processes.

Parameters

`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

`ppohAMRFDM_input_data`, `ppohAMRFDM_pre`, `ppohAMRFDM_addRoct`, `ppohAMRFDM_addGoct`

Calls the following subroutines/functions

`mpi_abort`

ppohAMRFDM_input_data

This subroutine loads ‘`input.dat`’ control file, and check the loaded parameters.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`ppohAMRFDM_abort`

ppohAMRFDM_pre

This subroutine allocates variables, makes initial mesh set according to '`input.dat`' control file.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`ppohAMRFDM_barrier, ppohAMRFDM_abort, ppohAMRFDM_setID, ppohAMRFDM_setev, ppohAMRFDM_make_RMesh, ppohAMRFDM_make_GMesh, ppohAMRFDM_refine_base`

ppohAMRFDM_refine_init

This subroutine refines mesh initially.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`ppohAMRFDM_average`, `ppohAMRFDM_setflag`, `ppohAMRFDM_addRoct`, `ppohAMRFDM_addGoct`, `ppohAMRFDM_connect_oct`, `ppohAMRFDM_set_buffer_ave`, `ppohAMRFDM_passing_iFLG`, `ppohAMRFDM_sortoct`

ppohAMRFDM_refine

This subroutine refines mesh in main loop iteration.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`
`iLv`	`I      in     Layer level`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`ppohAMRFDM_average`, `ppohAMRFDM_setflag`, `ppohAMRFDM_addRoct`,
`ppohAMRFDM_addGoct`, `ppohAMRFDM_connect_oct`,
`ppohAMRFDM_set_buffer_ave`, `ppohAMRFDM_passing_iFLG`,
`ppohAMRFDM_sortoct`

ppohAMRFDM_DDD

This subroutine re-decompose the whole computational domain into new sub-domains so that the computational load on each MPI process becomes nearly the same.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`ppohAMRFDM_DDD_check`, `ppohAMRFDM_loadbalance`, `ppohAMRFDM_passing_alldomain`, `ppohAMRFDM_connect_newocts`, `ppohAMRFDM_make_GMesh`, `ppohAMRFDM_refine_DDD`, `ppohAMRFDM_sortoct`, `ppohAMRFDM_passing_fields`

ppohAMRFDM_passings

This subroutine updates the information on physical variables '`F`' array at the domain boundaries by using MPI

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`		
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`		
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`		
`iLv`	`I`	`in`	`Layer level`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

`ppohAMRFDM_passing_field, ppohAMRFDM_average, ppohAMRFDM_interpolate`

ppohAMRFDM_load_parameters

This subroutine loads parameters required for calculation of kernel part.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`		
`iLv`	`I`	`in`	Layer level
`nID`	`I`	`out`	Number of active oct at the level
`dt`	`R`	`out`	Time interval @Lv.0 layer
`dx`	`RA`	`out`	Spatial interval @Lv.0 layer

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

None

ppohAMRFDM_load_mesh

This subroutine loads the structure variables of mesh information from ‘`Mesh`’ array.

Parameters

`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`		
`p0`	Derived type: `st_ppohAMRFDM_octset`		
`iLv`	`I`	`in`	Layer level
`index`	`I`	`in`	Index of ‘`Mesh`’ array

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

None

ppohAMRFDM_store_mesh

This subroutine stores the structure variables of mesh information to ‘`Mesh`’ array.

Parameters

`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`		
`p0`	Derived type: `st_ppohAMRFDM_octset`		
`iLv`	`I`	`in`	Layer level
`index`	`I`	`in`	Index of ‘`Mesh`’ array

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

None

ppohAMRFDM_load_index

This subroutine loads the mesh (grid) index of intended Lv layer from ‘`Mesh`’ array.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`		
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`		
`iLv`	`I`	`in`	Layer level
`index`	`I`	`in`	Index of ‘`Mesh`’ array
`ix`	`I`	`out`	Grid index in x direction @iLv
`iy`	`I`	`out`	Grid index in x direction @iLv
`iz`	`I`	`out`	Grid index in x direction @iLv

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in ppOpen-APPL/AMR-FDM

None

Calls the following subroutines/functions

None

4. advAMR3D

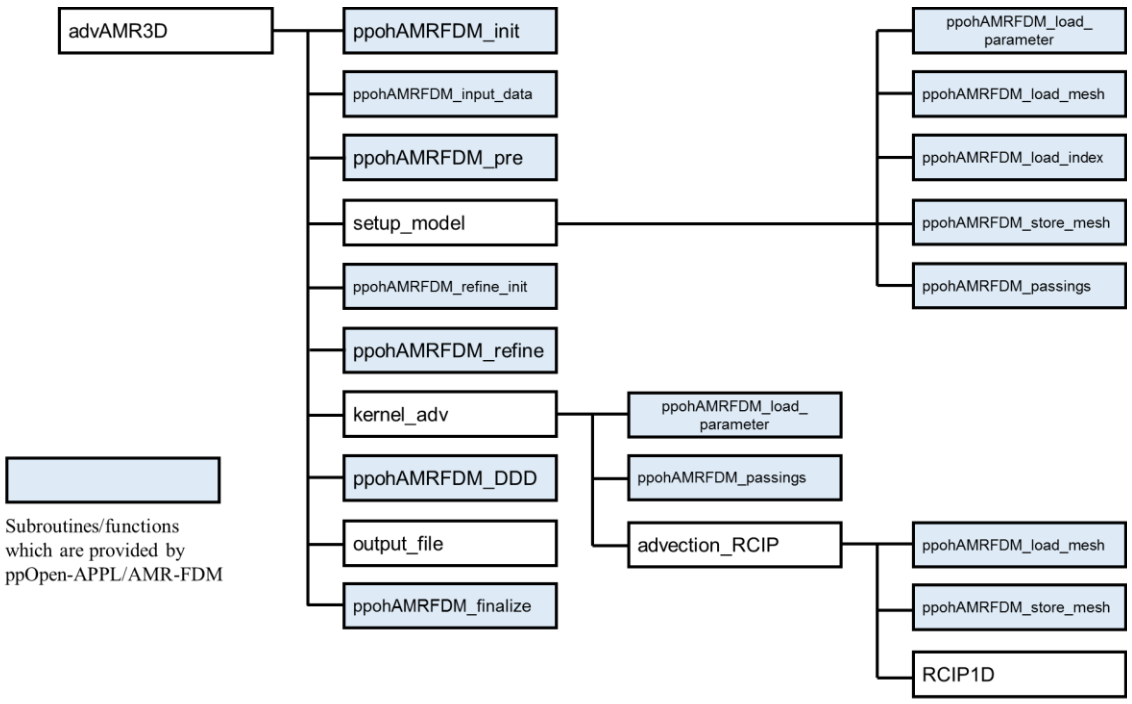
4.1 Overview of advAMR3D

For the demonstration of the application using the ppOpen-APPL/AMR-FDM, the advAMR3D for linear advection equation solver is developed on the ppOpen-APPL/AMR-FDM. The constrained interpolation profile (CIP) scheme is implemented as an explicit time-marching scheme in the advAMR3D. The advection equation is as follows:

$$\frac{\partial f}{\partial t} + c_x \frac{\partial f}{\partial x} + c_y \frac{\partial f}{\partial y} + c_z \frac{\partial f}{\partial y} = 0$$

where, f, c_x, c_y, and c_z indicate the waveform and the characteristic velocities in the x, y, and z directions, respectively.

4.2 Configuration



4.3 Modules

None

advAMR3D

Main Program of advAMR3D

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in advAMR3D

None

Calls the following subroutines/functions

`ppohAMRFDM_init`, `ppohAMRFDM_input_data`, `ppohAMRFDM_pre`, `setup_model`, `ppohAMRFDM_refine_init`, `output_data`, `ppohAMRFDM_refine`, `kernel_adv`, `ppohAMRFDM_DDD`, `ppohAMRFDM_finalize`

setup_model

This subroutine sets up the initial condition.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in advAMR3D

`advAMR3D`

Calls the following subroutines/functions

`ppohAMRFDM_load_parameter, ppohAMRFDM_load_mesh,`
`ppohAMRFDM_load_index, ppohAMRFDM_store_mesh, ppohAMRFDM_passings`

kernel_adv

This subroutine configures the kernel part of the advAMR3D.

Parameters

`st_param`	Derived type: `st_ppohAMRFDM_param`		
`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`		
`st_comm_info`	Derived type: `st_ppohAMRFDM_comm_info`		
`iLv`	`I`	`in`	Layer Lv

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in advAMR3D

`advAMR3D`

Calls the following subroutines/functions

`ppohAMRFDM_load_parameter, advection_RCIP, ppohAMRFDM_passings`

advection_RCIP

This subroutine updates the waveform function of an advection equation in each direction.

Parameters

`st_meshset`	Derived type: `st_ppohAMRFDM_meshset`		
`ch`	`C`	`in`	Direction
`iLv`	`I`	`in`	Layer level
`dt`	`I`	`in`	Time interval
`dx`	`IA`	`in`	Spatial interval
`nID`	`I`	`in`	Number of active oct at the level

Uses the following Modules

`m_ppohAMRFDM_util`

Called by the following subroutines/functions in advAMR3D

`kernel_adv`

Calls the following subroutines/functions

`ppohAMRFDM_load_mesh, RCIP1D, ppohAMRFDM_store_mesh`

RCIP1D

This subroutine updates the waveform function of the advection equation by rational CIP method.

Parameters

`PV`	R	in	Primitive value
`PVup`	R	in	Primitive value at upstream part of `PV`
`VIA`	R	in	Moment value
`VIAup`	R	in	Moment value at upstream part of `VIA`
`us`	R	in	Characteristic speed of waveform in a direction
`nPV`	R	out	Updated primitive value
`nVIA`	R	out	Updated moment value
`dts`	R	in	Time interval
`dtx`	R	in	Spatial interval

Uses the following Modules

none

Called by the following subroutines/functions in advAMR3D

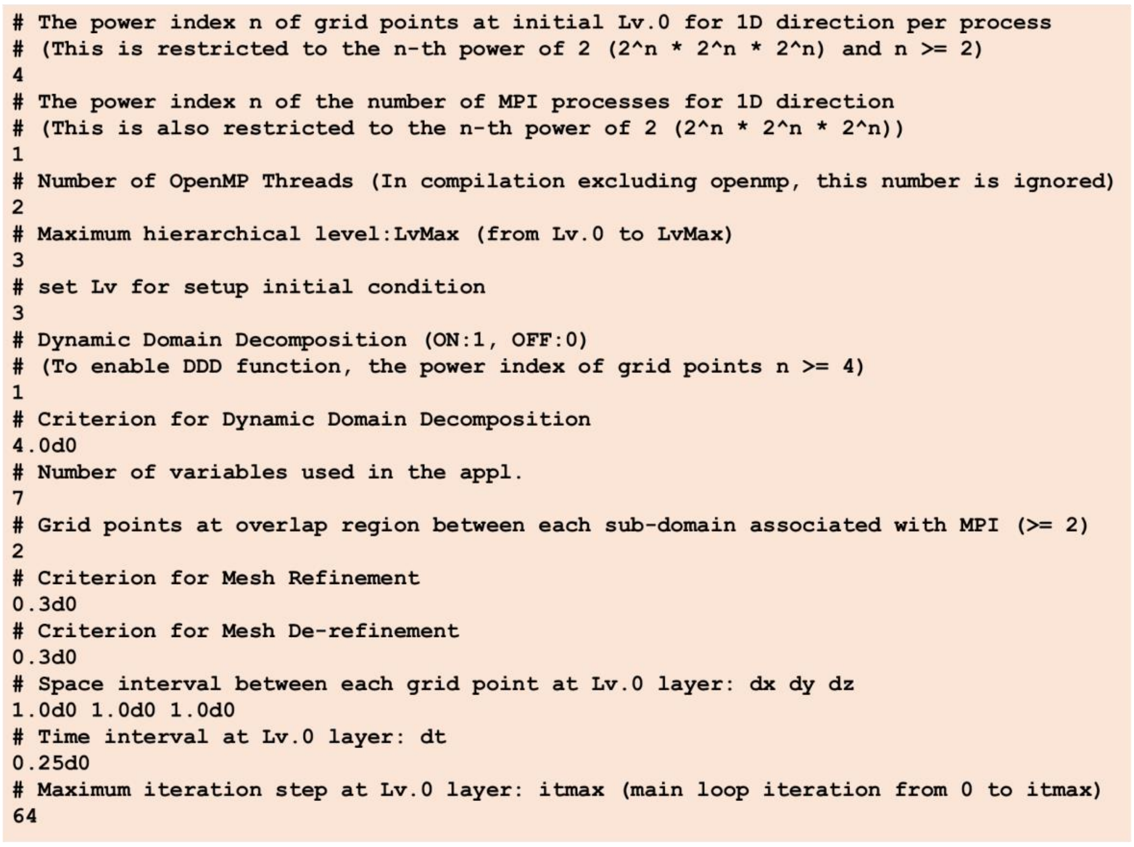
`advection_RCIP`

Calls the following subroutines/functions

none

4.5 Procedure of advAMR3D

To run the advAMR3D, control file (`input.dat`) used by the ppOpen-APPL/AMR-FDM is required. This control file consists information, as follows.



```
# The power index n of grid points at initial Lv.0 for 1D direction per process
# (This is restricted to the n-th power of 2 (2^n * 2^n * 2^n) and n >= 2)
4
# The power index n of the number of MPI processes for 1D direction
# (This is also restricted to the n-th power of 2 (2^n * 2^n * 2^n))
1
# Number of OpenMP Threads (In compilation excluding openmp, this number is ignored)
2
# Maximum hierarchical level:LvMax (from Lv.0 to LvMax)
3
# set Lv for setup initial condition
3
# Dynamic Domain Decomposition (ON:1, OFF:0)
# (To enable DDD function, the power index of grid points n >= 4)
1
# Criterion for Dynamic Domain Decomposition
4.0d0
# Number of variables used in the appl.
7
# Grid points at overlap region between each sub-domain associated with MPI (>= 2)
2
# Criterion for Mesh Refinement
0.3d0
# Criterion for Mesh De-refinement
0.3d0
# Space interval between each grid point at Lv.0 layer: dx dy dz
1.0d0 1.0d0 1.0d0
# Time interval at Lv.0 layer: dt
0.25d0
# Maximum iteration step at Lv.0 layer: itmax (main loop iteration from 0 to itmax)
64
```

Figure 8 Example of a control file (`input.dat`) for advAMR3D using the ppOpen-APPL/AMR-FDM

Actually, there are some restrictions to run the application in the current version. For example, if users set the maximum hierarchical layer or the grid points at Lv.0 too high, memory may be exhausted. The management of memory in the ppOpen-APPL/AMR-FDM is insufficient in the current version 0.3.0. This problem will be fixed in future. Furthermore the grid points at initial Lv.0 layer is restricted to the n-th power of 2, and the total grid poins in each direction have to be the same because the Morton ordered space filling curve is used in the ppOpen-APPL/AMR-FDM.
The advAMR3D creates result files (prop*****rank****.dat) in every time step. The result files are translated by ‘post_vtk’, and vtk files are created in `output` folder. the simulation results can be browsed by ParaView. The post_vtk is single code for visualization and have control file (`out_range.dat`) to specify the time range of the result files for the visualization.

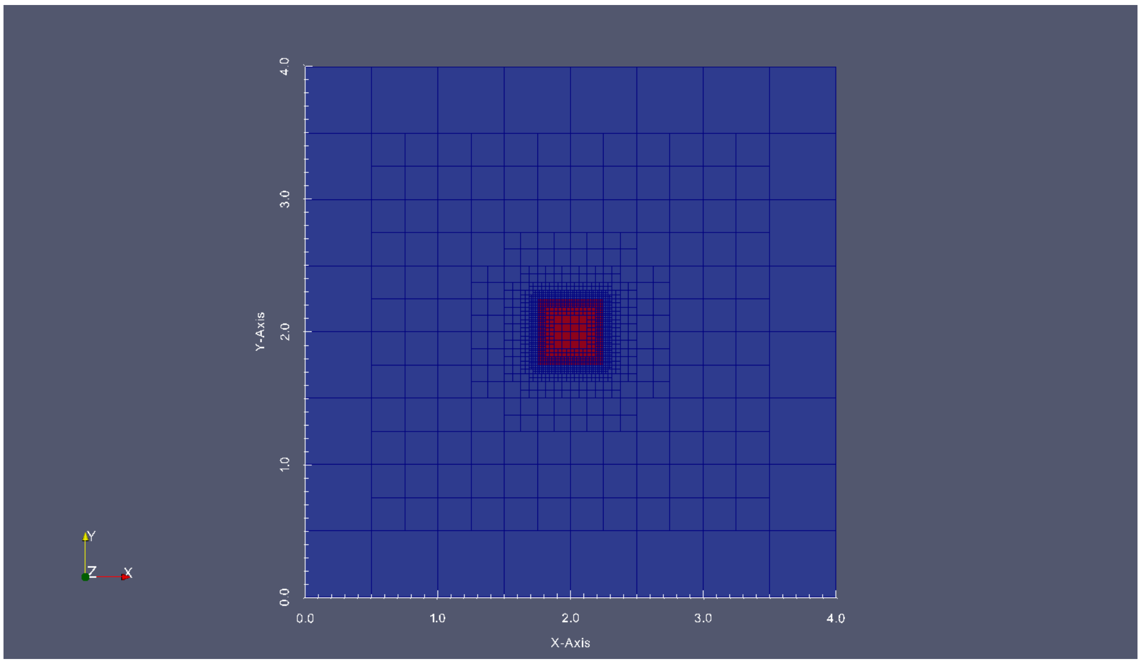


Figure 9 An example of simulation result by advAMR3D. The distribution of rectangular waveform and refinement mesh are shown.