Development of a Computational Framework for Block-Based AMR Simulations

Hideyuki Usui¹, Akihide Nagara¹, Masanori Nunami² and Masaharu Matsumoto³

¹The Graduate School of System Informatics, Kobe University, Kobe, Japan
²National Institute for Fusion Science, Toki, Japan
³Information Technology Center, The University of Tokyo, Tokyo, Japan

h-usui@port.kobe-u.ac.jp

Abstract

AMR technique can provide efficient numerical calculation by adapting fine cells to regions where higher numerical resolution is required. However, it is generally difficult for users to implement the AMR technique in their generic simulation programs which use uniform cells. For the purpose of carrying out numerical simulations including the AMR technique, we developed a framework for blocked-structured AMR simulation by which we can easily convert a generic uniform-cell simulation program to the one with the AMR treatment. In this paper we describe the developed framework and show the implementation of a simulation program into the framework by taking a two-dimensional advection simulation as an example.

Keywords: Adaptive Mesh Refinement, block-based AMR, AMR simulation, framework

1 Introduction

Large-scale parallel computation has been widely performed in various fields of engineering as well as science and its importance is rapidly growing in each field. The amount of available computational resources, however, is finite and sometime insufficient to realize a large-scale simulation model required to solve a specific problem. Despite the increasing computational power, lack of numerical resolution is a persistent problem in simulation studies. To efficiently use the available computational resources with keeping the required high resolution in computation, adaptive mesh refinement (AMR) technique [Zeeuw Powell, 1993] is one of the promising methods, which has been mostly used in fluid-type simulations. If we need a non-adaptive multi-scale simulation, it is can be easily realized with combination of uniform-cell simulation space with different cell size. However, in initial-value problems in which we need to solve various phenomena from the transient state to the steady state, it is generally difficult to predict a region where we need high resolution. In such a case,
AMR becomes a powerful method because simulation regions with high resolution are automatically created during a simulation run. The AMR technique can provide efficient numerical calculation by adapting fine cells to regions where higher numerical resolution is required. As an example of AMR framework, PARAMESH [MacNeice, Olson, Mobarry, deFainshtein, Packer, 2000] is used in several fields.

In our research group, we developed an innovative plasma particle code using AMR called PARMER [Usui, Nunami, Moritaka, Matsui, Yagi, 2011]. PARMER incorporates so called cell-based AMR technique with which we adaptively generate hierarchical layers with different cell size at the cells where a certain physical quantity exceeds a threshold for refinement. We would like to apply the AMR technique used in PARMER to generic simulation programs used in various research fields. It is, however, generally difficult to implement the AMR treatment in conventional simulation codes which are programmed based on uniform cell size. Particularly it is complex to numerically handle hierarchical layers with different cell size in a simulation run. Therefore, in this study, for those who are interested in adopting the AMR technique to their own simulation programs, we developed an advanced AMR framework with which the AMR technique is relatively easily introduced in a generic simulation program. To remove the difficulty of porting AMR to generic codes, we set that a region which needs to be refined is limited to a block-structured one consisting of the fixed number of cells. In the present study, our method can be applied to a simulation program in which differential equations are locally solved with finite difference method. We do not consider a solver which uses global operations with spatial synchronization such as Poisson’s equation as an application to the developed framework.

In this paper, our newly developed AMR framework is described in Sec. 2. In Sec. 3, we show an example of the AMR framework application to a simple simulation in which a two-dimensional advection equation is solved.

## 2 Block-Based AMR Framework

### 2.1 Block-Structured Domain

In the framework development, we decided to adopt the block-structured AMR [Berger Oliger, 1984] [Norman Bryan, 1999] because of better portability than other AMR-structures. In the block-structured AMR, regions required for the AMR treatment in the simulation domain have a self-similar structure. The self-similar block-structured domains for AMR are managed in a fully threaded tree (FTT) data structure which allows recursive refinement on a block-by-block basis [Khokhlov, 1998]. Each block consists of a domain formed with the fixed number of cells with uniform cell size. A block in a different level of refinement in the FTT structure has different cell size keeping the

![Figure 1: Example of block configuration in two-dimensional domain. In this case, a block has 4×4 cells.](image)
same number of cells. For instance, in one level higher, the cell size becomes half. In each block, we can incorporate the same uniform-cell simulation program of our interest and independently perform the simulation. Since each block has a common domain with the same number of cells because of self-similarity, what we need to consider is the cell size in each block depending on the refinement level which is given in the FTT structure.

A simple example is shown in Figure 1 in which each block has 4×4 cells and the simulation domain consists of 4×4 base-blocks. We call a block with the coarsest cells the base-block. It should be noted that the number of cells in the refined block is the same as that in base-block because of self-similarity although the cell size becomes half. The number of cells in each block and the number of base-block consisting of the whole simulation domain can be initially set as input parameters.

Figure 2 shows the block connections in the one-dimensional FTT structure. Each block has information about the level of refinement given in the FTT structure and the physical quantities defined at each cell in the block. Every block has three kinds of pointer which refer to neighbor, child, and parent blocks determined in the FTT structure. The neighbor pointer configures connections to adjacent blocks in the same hierarchical level. The child pointers and the parent pointer configure connections to blocks with fine and coarse cells located in the same region in different hierarchical levels, respectively. The parent block remains when child blocks are generated. In three-dimensional model, the physical quantities in each cell in the child blocks are obtained by the linear interpolation of those of eight parent-level cells which are most closely located around the corresponding child cell.

Numerical connection to neighboring blocks is configured by a buffer region. As shown in Figure 3, each block has a buffer region at the edges of the physical domain of the block. The buffer region consists of additional cells which are necessary in updating the physical quantities in the physical domain by the finite difference method. In the base-block level, as shown in (a) of Figure 3, the physical values at the boundary of the adjacent block are simply copied to the cells in the buffer region. Although one cell is assigned for the buffer region in Figure 3, the number of the buffer cells can be arbitrarily set by the users depending on the requirement of the finite difference method used in the corresponding simulation solver. In the child block, as shown in (b) of Figure 3, a buffer region is also attached at the edge of the physical region and the physical quantities in the buffer cell are inherited from the base-block by the linear interpolation as explained above. By making use of the buffer region, we can treat each block as an independent unit even if blocks in different level of hierarchy face to each other at the block boundary. As stated earlier, the hierarchical relation of all blocks is managed in the FFT structure, which allows us to incorporate the AMR treatment in simulation with a high degree of flexibility. In block calculation routine, the user can easily insert a generic numerical code, which is described in the following section.
2.2 Procedures of the AMR Framework Usage

We can easily modify a generic uniform-cell simulation program into an AMR one by inserting it into our developed AMR framework. Since the framework can handle the hierarchical relation among the blocks with the FTT structure, what the users basically have to prepare is the outer boundary condition for the entire simulation space, the main routine for calculation in each block, and a criterion for the cell refinement. We need to prepare these three parts in the Fortran language. In the following we explain the AMR framework usage by describing a program implementation to the framework.

The basic structure of the framework is shown in Figure 4. We use block arrays which are defined as structure data to keep the data such as physical quantities in blocks. In the framework, there are two major loops; the outer loop works for the temporal evolution with the number of time steps $N_{\text{step}}$ and the inner one is for the block calculation with the number of blocks $N_{\text{all}}$. Before the main $do$ loops, we need to provide parameters regarding blocks such as the number of cells per block and the total number of base-blocks consisting of a whole simulation domain as well as initial conditions for physical quantities which are to be solved in the simulation.

In the outer loop corresponding to the time increment, the framework has important routines regarding the cell refinement which has to be done before the block loop. The functions of the routines are the followings. (1) check if the physical quantities in a cell exceed the criterion for the cell refinement which is initially set, (2) generate child blocks if necessary, (3) connect blocks with pointers and construct a FTT, (4) exchange the boundary data between neighbor blocks for buffer region in the same hierarchical level, and (5) copy the data from the parent to the child blocks by interpolation when refinement occurs. These functions are included in the framework and the users do not have to modify except for the setup of the refinement criteria for a certain physical quantity.

Figure 3: Buffer region at the edges of block. (a) data exchange between buffer cells in the base-block level, (b) data copy to buffer cell from the parent block by interpolation.
In the inner loop, user’s solver is inserted and the physical quantities in all the blocks are updated. First, the contents of the block arrays containing the physical quantities in the simulation are copied to temporary arrays defined in the framework such as \( A, B, \) and \( C \) (Procedure (1) in Figure 4). Second, we update the physical quantities by performing a simulation using the arrays \( A, B, \) and \( C \) in each block (Procedure (2) in Figure 4). Since the cell size is uniform in each block, the inserted generic program can be used without any consideration of the cell refinement, which is the advantage of using the current AMR framework. Finally, we copy the updated values in the temporary arrays to the internal block arrays defined in each block (Procedure (3) in Figure 4). The treatment of the outer boundary condition for the entire simulation is done after the main calculation at each time step.

Before the actual simulation performance, we need to determine several parameters for the framework use. For simplicity, we assume a two-dimensional simulation space. First, we set the number of cells contained in one block. The block shape can be configured into a rectangle, not limited to a square. Since we adopt self-similar block system in the current framework, the number of cells in a block is maintained even if the cell size changes in the different hierarchical level. The cell size of the base-block, \( dx \) and \( dy \) in the \( x \) and \( y \) directions respectively, also have to be given. Then we need to determine how many base-blocks are necessary to realize the whole simulation domain.

Once the base-block setup is done, we need to set some parameters for the cell refinement. Prior to the AMR simulation, the user has to determine the maximum number of the hierarchical level \( n_{\text{max}} \) in the FTT structure by considering how high the spatial resolution is needed in the corresponding simulation. The cell sizes in a block with one-level higher refinement become \( dx/2 \) and \( dy/2 \) in the \( x \) and \( y \) directions respectively. In the same manner, the cell size for the \( n \)-th level block becomes \( dx/2^n \) where \( n \) denotes the hierarchical level with respect to the base-block level. Although the cell size

```
program
  type(oct_Block) :: block
  real :: A(nx,ny,nz), B(nx,ny,nz), C(nx,ny,nz)

Initial parameter setup
  do istep=1,Nstep (Time loop)
    Treatment of cell refinement
      do index=1,Nall (Block loop)
        Main calculation
          Data copy
            A(:,:,i) = block(index)%F(1,:,i,:)
            B(:,:,i) = block(index)%F(2,:,i,:)
            C(:,:,i) = block(index)%F(3,:,i,:)
          Insert uniform-cell program
            subroutine advance_field
              real :: A(nx,ny,nz), B(nx,ny,nz), C(nx,ny,nz)
              do iz=1,nz
                do iy=1,ny
                  do ix=1,nx
                    A(ix,iy,iz) = ***
                    B(ix,iy,iz) = ***
                    C(ix,iy,iz) = ***
                  enddo
                enddo
              enddo
            end subroutine advance_field
          Data copy to block arrays
            block(index)%F(1,:,i,:) = A(:,:,i)
            block(index)%F(2,:,i,:) = B(:,:,i)
            block(index)%F(3,:,i,:) = C(:,:,i)
        enddo
        Outer Boundary treatment
      enddo
  enddo
end program
```

Figure 4: Incorporation of user’s program to the AMR framework
differs in each hierarchical level, we use a common temporal interval $dt$ which is set for the $n_{\text{max}}$ level in other different levels. The user also has to set some criteria for the physical quantities for the cell refinement. The criteria are arbitrary such as some local values or gradients of any physical quantities solved in the corresponding simulation. When the corresponding physical values exceed the criteria the cell refinement will be conducted on the block basis in the simulation.

For the purpose of high performance computation, parallel calculation is inevitable by using multiple processes. The current AMR framework is applicable to parallel calculation. In parallelization, the whole simulation domain consisting of base-blocks at the initial state is uniformly divided into the number of process used in the parallel calculation and each subdomain which is composed of a group of base-blocks is assigned to each process. As explained earlier, data in buffer cells located at the edge of the block boundary has to be provided from the neighbor block. When the block boundary in a subdomain coincides with the process boundary, the data copy to buffer cells between neighbor blocks has to be done through MPI because the corresponding two neighbor blocks are contained in different processes. This function is also incorporated to the current framework. From this point of view, this framework is also useful in parallelizing a user’s program with the domain decomposition method even if there is no need to use the AMR function.

3 Example of Block-Based AMR Simulation

3.1 Simulation Model

To demonstrate the effectiveness of the developed framework, we converted a simple simulation program on the two-dimensional advection equation into an AMR simulation program by using the framework. In solving the advection equation, we simply adopted the upwind differencing scheme (UDS).

$$f^{n+1}_{i,j} = f^n_{i,j} - c_x \frac{\Delta t}{\Delta x} (f^n_{i,j} - f^n_{i-1,j}) - c_y \frac{\Delta t}{\Delta y} (f^n_{i,j} - f^n_{i,j-1})$$

In the above equation, $f^n_{i,j}$ denotes a field value at the time step $n$ and cell point $(i, j)$. $c_x, c_y, \Delta x, \Delta y,$ and $\Delta t$ are the phase velocities and the cell sizes along the $x$ and $y$ directions, and the time step interval, respectively. In the simulation, we set the local field gradient as a criterion for the block-based cell refinement. The following shows the criterion used in the current test simulation.

$$\max \left[ \left| f_{i,j} - f_{i,j,3} \right|, \left| f_{i,j} - f_{i,j,2} \right| \right] > \kappa$$

When the local field gradient exceeds a fixed value $\kappa$ which is initially set, a new block with fine cells is locally created in the corresponding region. The maximum refinement level $n_{\text{max}}$ is 3 for this simulation. The initial configuration of blocks is shown in Figure 5. A block consists of 10×10 cells in the $x$-$y$ plane. We used four processes for the MPI parallel calculation with the domain decomposition scheme. Each process handles 5×5 base-blocks.

As an initial value of $f$, we set a square-shaped waveform at the center with the size of 40×40 cells of a base-block shown in Figure 4. Since the gradient of $f$ is the largest at the waveform boundary, blocks of the level 3 are initially created there. We provide an initial velocity to the waveform as $c_x=c_y$ and let the waveform propagate along the diagonal direction of the simulation plane. The outer boundary condition of the simulation domain is periodic.

3.2 Evaluation of the Test Simulation

Figure 6 shows the profile of the square-shaped waveform and its corresponding block configuration in the $x$-$y$ plane around the final stage of the simulation. Although not shown in the
figure, the waveform propagated across the simulation system multiple times. Because of numerical diffusion by UDS in solving the advection equation in the waveform propagation, the edge of the waveform diffuses and the region with the finite gradient of the field spreads. Then the area of the AMR treatment eventually increases around the waveform boundary as shown in the lower panel of Figure 6.

For comparison, we performed another simulation using a uniform cell system in which the smallest cell size, $dx/2^3$, used in the AMR simulation is adopted. We compared the waveforms obtained in two different simulations at the final time step. Although not displayed here, we could confirm that there is little difference found between the two waveforms. The temporal variation of the gradient of the waveform is the same in both simulations because the spatial resolution at the field gradient is supposed to be the same in the two cases. This implies that the AMR treatment sufficiently works in the modified simulation program by using the current framework.

Next we examined the computational efficiency in terms of calculation time and memory usage in the simulation. The calculation time in the block-AMR simulation is reduced approximately half in comparison with that in the uniform-cell simulation. In the current case, the total calculation time is about 400 seconds in the block-AMR simulation while approximately 200 seconds in the uniform-cell simulation. Even though there are some overheads for the AMR treatment, we confirmed the calculation time is much reduced in the block-AMR simulation.
In the simulation, we adopted the domain decomposition method for parallelization. Each subdomain is assigned to each process and the amount of memory consumed in each process varies in time because the AMR region moves in the simulation domain. To compare the memory consumption between the block-AMR simulation and the uniform-cell simulation, we take a specific subdomain where an AMR region is created and the maximum amount of memory is required. In the current simulation, the process which requires the maximum amount of memory in the AMR simulation consumes 72MB while 160MB is consumed in the uniform-cell simulation.

As long as the current simulation model is concerned it turned out that the total calculation time and the amount of the memory usage are both reduced almost by half when the AMR framework is introduced. This implies that a simulation with AMR treatment has an advantage in the reduction of the usage of computational resources.

4 Conclusion

To minimize the effort to introduce the AMR technique into generic simulation programs using uniform cells, we developed a blocked-structured AMR framework by which we can easily separate the calculation part from the AMR treatment in a simulation. A whole simulation domain consists of a number of base-blocks. The base-block is formed with cells and users can arbitrarily set the number of cells in a block prior to the simulation. If necessary, we can apply AMR to each block domain, not to a single cell, by setting a criterion of cell refinement for physical quantities such as density and field gradient. The newly created refined block has the half cell size but has the same number of cells as that of the parent block. Namely block domains are self-similar structures. All the block domains are organized in a FTT data structure so that the hierarchical relation among the blocks is well maintained. Users’ generic program is ported into each block domain and solved independently. Each block domain has a buffer region which surrounds the physical domain of the block. Through the buffer region, data of adjacent blocks are exchanged for the boundary calculation with finite difference methods. By these systematic connections among blocks, a simulation in a whole domain is efficiently performed.

In parallelizing a simulation program with multiple processes we generally use the domain decomposition method with which we uniformly separate the whole simulation region into subdomains with the number of processes available for the parallel calculation. In parallelizing the block-structured simulation, each divided subdomain consists either of a single block or a group of blocks depending on the memory size of each process. In the domain decomposition method we need to exchange the boundary data of each subdomain between adjacent processes. The data exchange between processes is handled with MPI and this treatment is also supported in the current block-AMR framework. From this point of view, the developed framework is also useful for users to parallelize a code with the domain decomposition method using multiple processes even if there is no need to use the AMR function in the simulation.

One of the issues in parallelizing a simulation program is the load balance among processes assigned to a simulation run. In a simulation using this block-AMR framework the load of each process changes in time because AMR regions can move from a subdomain to another. A subdomain which contains child domains has a load larger than the one which consists of base blocks only. This load imbalance among subdomains has to be solved for efficient parallel calculation. To maintain the load balance, reorganization of subdomains should be done during a simulation run. At this moment, however, we implemented no specific treatment to change the subdomains in the current framework, which should be somehow added to the framework in the near future.

In this paper, we described the basic concept and function of the developed framework as well as the procedure in porting a generic simulation code into this framework. By taking a two-dimensional advection equation as an example, we performed a test simulation of a square-shaped waveform.
propagation by using the developed AMR framework. In the waveform propagation, we could confirm
that high resolution is achieved adaptively and locally at the steep gradient of the waveform by the
creation of new blocks with fine cells. We also confirmed that the computation resources used for this
test simulation are reduced almost by half in comparison with those used in a uniform cell simulation
for the present case.

Currently this framework has been preliminarily applied to a Magneto-Hydrodynamics (MHD)
plasma simulation for the analysis of multi-scale and non-uniform plasma phenomena such as Kelvin-
Helmholtz instability. In addition to MHD model simulation, we plan to apply the framework to
plasma particle simulation. Since particles move randomly in a simulation domain, they do not remain
in a specific block. To be able to handle these particles in AMR simulation by using the framework,
we need to develop an additional function for the particle treatment and attach the function to the
current framework, which is left as a future work.

References

Berger, J., & Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations.
J. Comput. Phys.

Carlisle, D. (2010, April). graphicx: Enhanced support for graphics. Retrieved from
http://www.ctan.org/tex-archive/help/Catalogue/entries/graphicx.html

Khokhlov, A. (1998). Fully Threaded Tree Algorithms for Adaptive Refinement Fluid Dynamics
Simulations. J. Comput. Phys.

MacNeice, P., Olson, K., Mobarry, C., deFainshtein, R., & Packer, C. (2000). PARAMESH: A parallel
adaptive mesh refinement community toolkit. Computer Physics Communications.

Norman, M., & Bryan, G. (1999). Cosmological Adaptive Mesh Refinement. Numerical Astrophysics.

Usui, H., Nunami, M., Moritaka, T., Matsui, T., & Yagi, Y. (2011). A Multi-Scale Electromagnetic
Particle Code with Adaptive Mesh Refinement and Its Parallelization. Procedia Computer
Science.

Zeeuw, D., & Powell, K. (1993). An adaptively refined cartesian mesh solver for the euler equations.
J. Comput.Phys.