PARAMESH : A Toolkit for Parallel Adaptive Models

P. MacNeice
Drexel University
K. Olson
University of Maryland, Baltimore County
J. Merritt, M. Bhat and M. Rilee
L3 Communications Analytics Corp.

Abstract— **We have developed a software package, called PARAMESH, which offers developers of appropriate scientific numerical models the ability to adaptively refine their computational mesh. Written in Fortran 90, PARAMESH supports models which use structured meshes. We describe the PARAMESH package, and illustrate its use for a range of scientific applications.**

Keywords— **AMR, Parallel Computing.**

I. Introduction

MANY scientific modeling challenges today attempt to simulate processes which span very large ranges in spatial scale. For example, models of gravitational radiation produced by interacting neutron stars must resolve the neutron stars (10km scales) and also reproduce the asymptotic wave structure (at distances greater than 10000km from the neutron stars.)

These models compute the state of the system at a set of discrete points which form a numerical mesh. The spacing of these points establishes the spatial resolution of the model. They have reached a point where the finest uniform meshes which can be run on the largest computers do not provide sufficient resolution. Larger dynamic ranges in spatial resolution are required, which must also be capable, and for this researchers are looking to adaptive mesh refinement(AMR) techniques.

At the same time the largest computers are now highly parallel distributed memory machines which provide a challenging programming environment. Few genuinely shared memory machines exist, and those which do, with few exceptions, perform inefficiently unless the programmer takes aggressive control of decomposing their computational domain. The principal reason is that most shared memory machines are actually distributed memory machines with globally addressable memory. Data locality is often critical for good performance, because memory access times are not uniform, and fetching data from more remote memory can be relatively expensive.

Ideally it should not be necessary for the developers of these models to have to become experts in AMR techniques and parallel computing. It should be possible to make these techniques available and competitive by providing an appropriate toolkit which can be used to extend their existing codes.

In this paper we describe just such a portable community toolkit which we have been developing at NASA Goddard Space Flight Center. Called PARAMESH, our toolkit is designed to provide parallel support with adaptive mesh capability for a large class of models on distributed memory machines.

Our package of Fortran 90 subroutines, called PARAMESH is designed to provide an application developer with an easy route to extend an existing serial code which uses a logically cartesian structured mesh into a parallel code with AMR.

Alternatively, in its simplest use, and with minimal effort, it can operate as a domain decomposition tool for users who want to parallelize their serial codes, but who do not wish to use adaptivity. The package can provide them with an incremental evolutionary path for their code, converting it first to uniformly refined parallel code, and then later if they so desire, adding adaptivity.

The package is distributed as source code which will enable users to extend it to cover any unusual requirements.

In this paper we will briefly describe how PARAMESH works, and illustrate the range of applications which have been developed to use it. Reference [1] provides a more complete description.

II. Basic Package Design and Application

The PARAMESH package builds a hierarchy of subgrids to cover the computational domain, with spatial resolution varying to satisfy the demands of the application. These sub-grid blocks form the nodes of a tree data-structure (quad-tree in 2D or oct-tree in 3D).

All the grid blocks have an identical logical structure. Thus, in 2D, if we begin, for example, with a 6 x 4 grid on one block covering the entire domain, the first refinement step would produce 4 child blocks, each with its own 6 x 4 mesh, but now with mesh spacing one-half that of its parent. Any or all of these children can themselves be

refined, in the same manner. This process continues, until the domain is covered with a quilt-like pattern of blocks with the desired spatial resolution everywhere.

The grid blocks are assumed to be logically cartesian (or structured). By this we mean that within a block the grid cells can be indexed as though they were cartesian. If a cell's first dimension index is i, then it lies between cells i-1 and i+1. The actual physical grid geometry can be cartesian, cylindrical, spherical, polar(in 2D), or any other metric which enables the physical grid to be mapped to a cartesian grid. The metric coefficients which define quantities such as cell volumes are assumed to be built into the user's algorithm.

Each grid block has a user prescribed number of guard cell layers at each of its boundaries. These guard cells are filled with data from the appropriate neighbor blocks, or by evaluating user prescribed boundary conditions, if the block boundary is part of a boundary to the computational domain.

The package supports 1D, 2D, 2.5D (such as is used frequently in Magneto-Hydrodynamics applications where a magnetic field pointing out of the 2-D plane is kept), and 3D models.

Requiring that all grid blocks have identical logical structure, may, at first sight seem inflexible and therefore inefficient. In terms of memory use this is certainly true, although, even in extreme cases the associated memory overhead is rarely more than about 30%. However this has two significant advantages. The first and most important is that the logical structure of the package is considerably simplified, which is a major advantage in developing robust parallel software. The second is that the data-structures are defined at compile time which gives modern optimizing compilers a better opportunity to manage cache use and extract superior performance.

A simple example is shown in Figure 1 in which a 6 x 4 grid is created on each block. The numbers assigned to each block designate the block's location in the quad-tree below. The different shapes assigned to the nodes of the tree indicate one possible distribution of the blocks during a 4 processor calculation. The leaves of the tree are the active sub-grid blocks.

For accuracy and for simplicity, we insist that during the refinement process the refinement level is not allowed to jump by more than 1 refinement level at any location in the spatial domain.

The package manages the creation and removal of the grid blocks, builds and maintains the tree-structure which tracks the spatial relationships between blocks, distributes the blocks amongst the available processors and handles all inter-block and inter-processor communication. It can distribute the blocks in ways which maximize block locality and so minimize inter-processor communications. It also keeps track of physical boundaries on which particu-

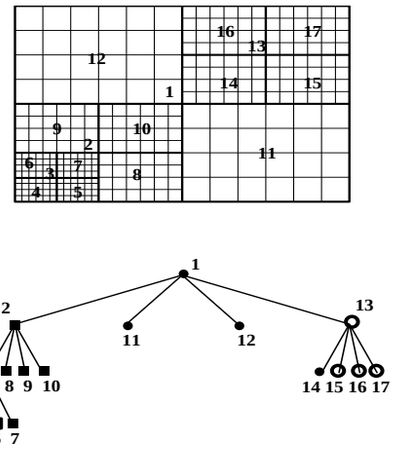

Fig. 1. A simple 2D example of a grid block tree covering a rectangular domain. The tree node shapes indicate how this tree might be distributed on a 4 processor machine to balance the workload. The heavy lines in the grid diagram indicate the boundaries of sub-grid blocks, and the lighter lines indicate individual grid cells.

lar boundary conditions are to be enforced, ensuring that child blocks inherit this information when appropriate.

The approach we have taken is to implement a simplified version of the AMR design of Berger and Oliger [2].

The philosophy we have adopted in constructing PARAMESH, is to remove from the application developer as much of the burden of inter-block and inter-processor communication as we possibly can. Hopefully, the result is that the application developer can focus on writing code to advance their solution on one generic structured grid-block which is not split across processors.

The parallel structure which PARAMESH assumes is a SPMD (Single Program Multiple Data) approach. In other words the same code executes on all the processors being used, but the local data content modifies the program flow on each processor. It is the message-passing paradigm, but with the burden of message-passing removed from the application developer by the package.

The programming task facing the user can be broken down into a series of straightforward steps.

1. Edit a few lines in a header files provided with the package which define the model's spatial dimensionality, the properties of a typical grid block, the storage limits of the block-tree, and the number of data words required in each grid cell for each of the packages data-structures. The header file is extensively commented to make this step easy.

2. Construct a main program. Most time-dependent fluid models will be able to use an example provided as a template for their main program. This can be easily modified to suit the user's requirements. The sequence of calls to the 'upper level' routines in the PARAMESH package should not need to be altered. The user will need to customize the construction of an initial grid, establish a valid

initial solution on this grid, set the number of timesteps and limits on the allowed range of refinement, and add any I/O required. Sample code for all these tasks is provided.
3. Provide a routine which advances the model solution on all the grid blocks through a timestep (or iteration). This step is much simpler than it appears. The routine can be constructed by taking the equivalent code from the user's existing application which advances the solution on a single grid, and inserting it inside a loop over the leaf blocks on the local processor. Inside this loop, the solution data for the current grid block must be copied from the package's data-structures into the equivalent local variables in the user's code segment. Then the user's code segment executes to update the solution on that block. Finally, the solution is copied back from the user variables to the package's data-structures, before the loop moves on to repeat the same sequence for the next leaf block. If conservation constraints must be satisfied, a few extra lines must be added inside this routine to capture fluxes and/or cell edge data at block boundaries.
4. Provide a routine to compute the model's timestep. Again this can be straightforwardly constructed from the existing template by inserting the appropriate code segment from the user's existing application, in the manner described in step 3. The existing timestep routine template has all the control and inter-processor communications required to properly compute the global minimum of the maximum timesteps calculated for each block, or to enable longer timesteps on coarser blocks if the user chooses that option.
5. Provide a routine to establish the initial state on the initial grid. A template has been provided which can be tailored to suit the user's model.
6. Provide a routine to set data values in guard cells at physical boundaries in order to implement the user's choices of boundary conditions. Once again a template routine has been provided which can be very easily modified.
7. Provide a function to test a single block to determine if any refinement or de-refinement is appropriate. This function is called during the refinement testing operation. Again, a template exists which can be modified by the user.

Detailed 'How To' instruction and illustration is provided in the User's manual which comes bundled with the software distribution, in the form of an HTML document.

Templates and worked examples are provided with the package for all of these tasks.

Our design philosophy while developing this package has been to present the user with a clean well commented Fortran 90 source code, sufficiently simple in structure that the user would not be afraid to customize routines for their own particular use. We also strove for efficiency on cache-based multiprocessors.

We have designed a number of tutorials which enable the user to familiarize themselves with the packages design before they begin to modify their application to use it.

The package is portable, working with either the MPI library or the Cray/SGI SHMEM library.

The package supports solution data located at grid cell center, at the centers of grid cell faces, edges and at cell corners.

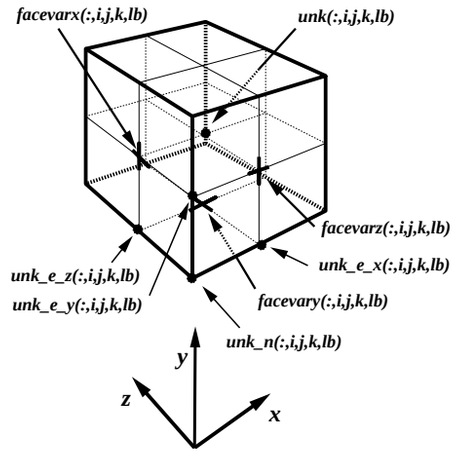

Fig. 2. Location of data with respect to mesh cell.

It includes some support for constraint equations, such as conservation control and to ensure interpolation which maintains divergence free properties.

We began development of PARAMESH five years ago, and have recently released version 3.0.

III. What Kind of Middleware is this?

PARAMESH does not fit the conventional description of a standard callable library. It must be open-source because there is no such thing as a standard scientific modeling application. Our experience has shown that each application developer wants to add their own feature(s) which extends the package, perhaps even breaking other features which they do not need. As a result it was essential that the code be amenable to user modification. This made the popular and highly abstract approach to object-oriented design unappealing, because it produces an unacceptably steep learning curve. We chose therefore to code directly in Fortran 90, and to focus on routine function rather than object properties.

IV. Selected Applications

PARAMESH has been distributed to more than one hundred researchers worldwide. Their applications span a wide range of scientific applications. Here we illustrate three which stress the AMR in different ways,

- modeling of coronal mass ejections
- nuclear detonation fronts on the surface of neutron stars
- interaction of compact gravitational objects.

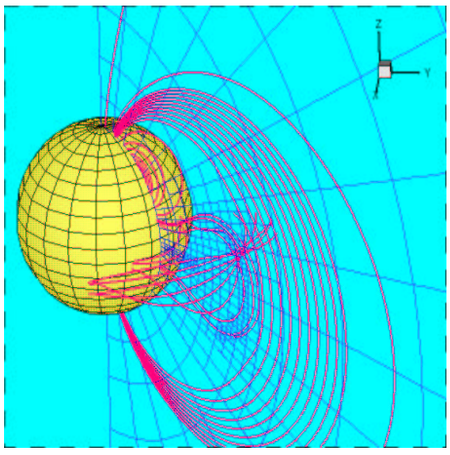

Fig. 3. Early stage in the development of flux rope in an idealized coronal mass ejection.

A. Coronal Mass Ejections

The first example is a 2.5D MHD model of a theory for the initiation of Solar coronal mass ejections(CMEs). Coronal mass ejections are the most significant drivers of Space Weather disturbances. The physical mechanisms responsible for CME initiation are not yet known. In this idealized simulation the initial magnetic field is a combination of dipolar and octupolar fields. The inner flux system about the Solar equator is sheared at the Solar surface causing it to expand upward. Magnetic reconnection begins where it pushes into overlying flux, and the mesh refines there to more accurately follow this process. Figure 3 shows the fieldlines as this begins to occur. The mesh block outlines are also shown.

The code used a Flux Corrected Transport algorithm[3], with magnetic field values specified at cell face centers, and requires that the magnetic field remain divergence free. It used a spherical coordinate system, with mesh cell spacing in the radial direction proportional to radius for a given refinement level.

B. General Relativity

The LISA project to detect gravitational radiation will need sophisticated numerical modeling support, both to predict and decipher complex signals which they hope to record. A group, under the direction of Dr. Joan Centrella at GSFC has begun development of the numerical models needed. These will need to resolve the interacting compact objects, on scales of 1-10 km, but also include the 'wave region', sufficiently far from the source that the wavefronts have achieved the wave pattern which will be detected near the Earth. This needs AMR [4]. In figure 4 they illustrate wave propagation in a test calculation using the code which produced the first AMR solution of wave propagation for the full Einstein equations.

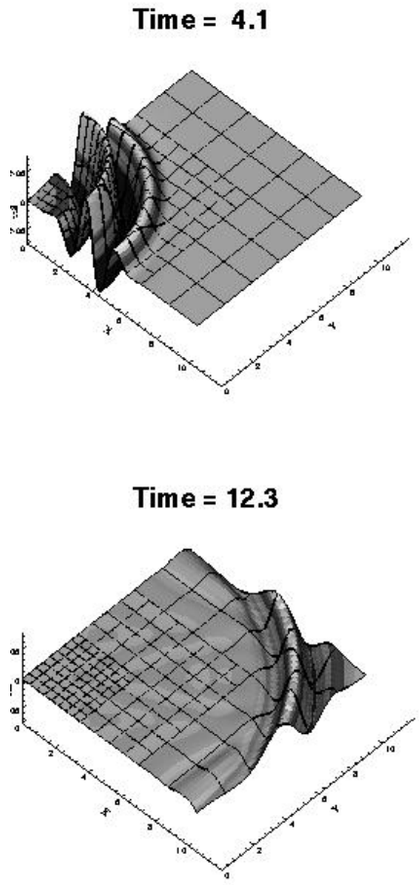

Fig. 4. A gravitational wave evolving as a solution of the full Einstein equations on a non-uniform grid. No significant wave noise is generated at the refinement discontinuity.

C. Astrophysics

The final example is a hydrodynamic simulation of the initial stages of a helium detonation propagating through the accreted envelope of a neutron star. Thermonuclear runaways on neutron stars are generally accepted to explain Type I X-ray bursts from neutron stars.

This simulation was developed by the FLASH Group at the University of Chicago [5,6]. It explores the evolution of a small 2 km wide section of the stellar surface near the origin of the detonation. Figure 5 illustrates the density distribution after 60 microseconds, as the detonation front propagates from its origin near the lower left corner. In the lower frame the outlines of the grid blocks are shown for the corresponding time. Each block is 8×8 mesh cells in size. The mesh included seven different refinement levels.

Their code uses a Piece-wise Parabolic algorithm, and their 2D computation was done in the (r, z) plane of a cylindrical coordinate system.

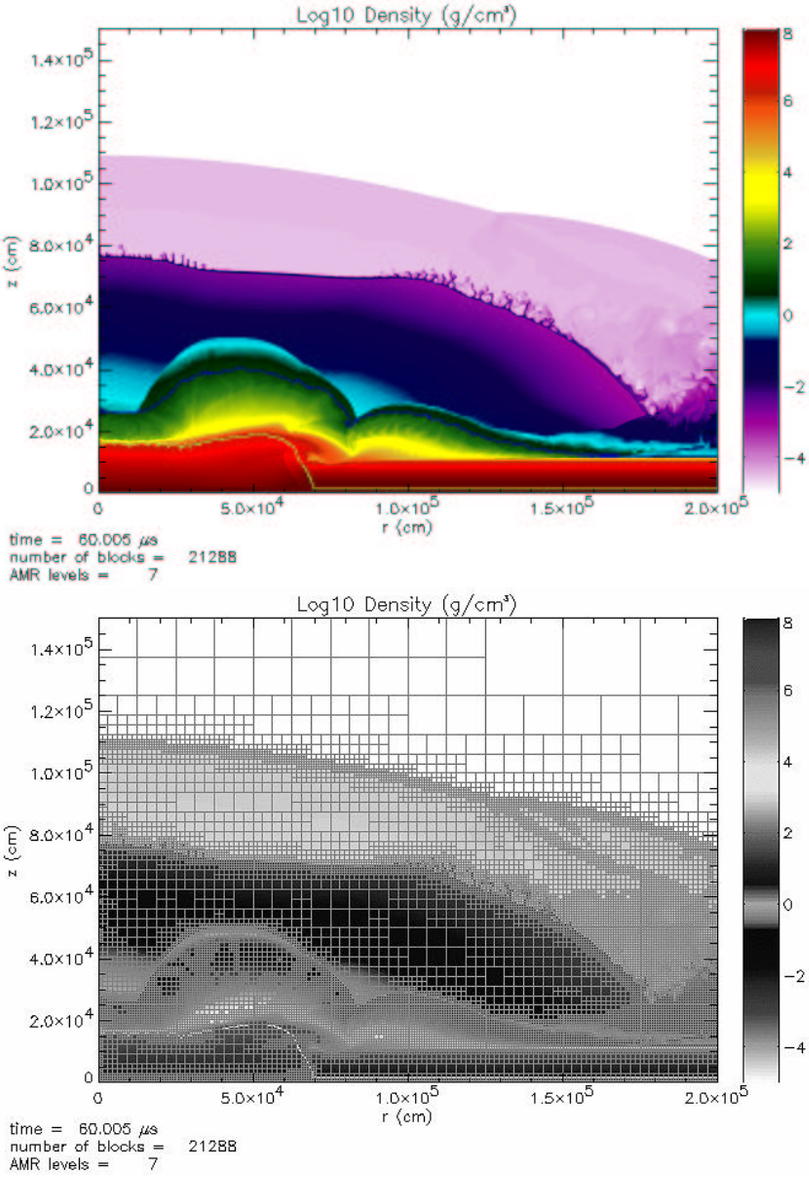

Fig. 5. Map of log of density map after 60 microseconds of a thermonuclear detonation in the accretion envelope of a neutron star, and the mesh structure for the same time.

Acknowledgments

The authors would like to acknowledge the assistance of Drs. D.S. Spicer, C.R. DeVore, the FLASH group at the University of Chicago and Mike Zingale, and Dr. Dae-Il Choi.

References

[1] MacNeice,P., Olson,K., Mobarry,C., deFainchtein,R. and Packer,C., *Computer Physics Communications*, vol. 126, p. 330, 2000.
[2] Berger,M.J., and Oliger,J., *J. Comp. Phys.*, vol. 53, p.484, 1984.
[3] DeVore, R., *NRL Memorandum Rep.*, No. 6544, 1989.
[4] New,C.B., Choi,D., Centrella,J.M., MacNeice,P. Huq,M.F., and Olson,K., *Phys. Rev. D*, vol. 62, p.84039, 2000.
[5] Fryxell, B. et al, *Ap.J.Suppl.*, vol. 131, p.273, 2000.
[6] Zingale,M. et al, *Ap.J.Suppl.*, vol. 133, p.195, 2001.