LA-UR 97-336

Approved for public release; distribution is unlimited

Title:

Parallel Object-Oriented Adaptive Mesh Refinement

Author(s): Dinshaw Balsara
Daniel J. Quinlan

Submitted to: Proceedings of Eighth SIAM Conference on Parallel Processing for Scientific Computing
March 14-17, 1997
Minneapolis, Minnesota

Los Alamos
National Laboratory

CONF-970342--11
LA-UR--97-336

Parallel Object-Oriented Adaptive Mesh Refinement *

Dinshaw Balsara † Daniel J. Quinlan ‡

Abstract

In this paper we study adaptive mesh refinement (AMR) for elliptic and hyperbolic systems. We use the Asynchronous Fast Adaptive Composite Grid Method (AFACx) [4], a parallel algorithm based upon the of Fast Adaptive Composite Grid Method (FAC) [5] as a test case of an adaptive elliptic solver. For our hyperbolic system example we use TVD and ENO schemes for solving the Euler and MHD equations [25, 26, 27]. We use the structured grid load balancer MLB [10] as a tool for obtaining a load balanced distribution in a parallel environment. Parallel adaptive mesh refinement poses difficulties in expressing both the basic single grid solver, whether elliptic or hyperbolic, in a fashion that parallelizes seamlessly. It also requires that these basic solvers work together within the adaptive mesh refinement algorithm which uses the single grid solvers as one part of its adaptive solution process. We show that use of AMR++ [3], an object-oriented library within the OVERTURE Framework [11], simplifies the development of AMR applications. Parallel support is provided and abstracted through the use of the P++ parallel array class [6, 7, 8].

1 Introduction

There are several areas of science and engineering where the numerical solution of physical problems on logically rectangular grids is highly desired. Such fields of study include physics, astrophysics, mechanical and aerospace engineering and geophysics to name but a few. In many of these fields of study useful algorithms have been evolved for quite some time. These algorithms have several requirements put on them, robustness, accuracy and stability being some of the many important criteria. While significant advances have been made in the design of unstructured mesh algorithms they do sometimes show deficiencies, especially in error analysis, speed, parallelism and ease of data management. For this reason we restrict our attention to structured mesh algorithms. While algorithmic excellence can often be ensured, these fields have another demanding requirement which consists of being able to solve problems where the interesting phenomena happen over a range of length scales. Should these phenomena happen all over the computational domain one has no option but to fully resolve the computational domain to the desired resolution. However, in several physical problems, phenomena requiring high spatial resolution occur only in small localized regions. It is in such situations that adaptive mesh refinement (AMR) techniques, originally pioneered by [29, 30, 12], to name but a few, come to the fore. Such techniques rely on detecting where extra resolution is needed and putting extra grid points in exactly those regions. Doing AMR in serial environments is, therefore, a mostly solved problem.

*This work supported by the U.S. Department of Energy through Contract W-7405-ENG-36.

†N.C.S.A. Univ. of Illinois

‡Computing, Information and Communications Division, Los Alamos National Laboratory, Los Alamos, NM. http://www.c3.lanl.gov/cic19/teams/napc/

However scientists and engineers have an almost insatiable need for higher resolution. The speed increases in superscalar and vector supercomputers has been linear as a function of time for some time. Parallel computers, on the other hand, seem to be growing in size, speed, memory and bandwidth at a much faster pace, hence our interest in parallel adaptive mesh refinement. It is in such situations that the capabilities developed here become useful.

Several of the problems in the above fields of study are of elliptic, parabolic or hyperbolic sort. We, therefore, focus on doing parallel AMR for elliptic and hyperbolic problems. General solution strategies often entail mixing multiple solvers in the same application. This produces yet another complication, especially in a parallel environment because multiple solvers have to work together and data associated with one type of solver has to be kept logically separate from data associated with another solver even as the two are required to share data. Furthermore, in a serial environment this data would be put in multidimensional arrays which occupy contiguous portions of memory, while in a parallel environment the data is distributed across several processors and has to be managed dynamicly across those processors. To add yet another layer of complexity, meshes of different topology might be required to handle boundaries. Thus mesh generation and mesh refinement are required to work together. A typical AMR application has a base grid and successive levels of refined grids nested one within the other. Each grid has a certain amount of computational work associated with it which may or may not be proportional to the number of grid points.

Optimally scalable performance in such situations places two further requirements, both having to do with load balancing. Firstly, we would like to be able to build grids on processors that are relatively idle. Moreover, as time evolves, the grid hierarchy changes making it necessary for us to be able to shuttle existing grids from one set of processors to another.

While the above paragraph is meant to lay out the problem, in this paragraph we present in a broad brush stroke fashion (and without any details) the solution strategies that have been evolved. Details will be relegated to the ensuing sections. This paragraph may, therefore, be taken as a lexical and intuitive introduction to these solution strategies. Having the capability to have multiple solvers work together even in a serial environment raises the need for some form of data abstraction and encapsulation. A distinction has to be drawn between private and public data to ensure that one solver does not access data that was not intended for it. Furthermore, with multiple meshes represented in the AMR mesh hierarchy, an operation performed on data associated with one mesh should not access data on another mesh. This directly points to the need for an object oriented language to be used even in a serial environment. We use C++ here since it is currently one of the most popular and well-developed object-oriented languages. In a parallel environment, multidimensional arrays will be distributed across many processors. While single mesh applications can keep data across boundary zones coherent through explicit message passing, this would prove too daunting a task for an application that had an unusually large number of meshes each with multiple multidimensional arrays associated with it. For that reason we use the P++ class library, [6, 8] to provide a logically contiguous index space for multidimensional arrays. This also allows data parallel concepts to be used in achieving load balancing on each single logically rectangular mesh. Including mesh generation in the refinement process necessitates having an object oriented strategy for encapsulating information associated with the problem topology. This is done via the Overture class library, [11]. The adaptive mesh refinement strategy, called AMR++ here, is a further class library which encapsulates strategies for handling multiple meshes on multiple levels of refinement. We instantiate the adaptivity for elliptic systems and hyperbolic systems. The elliptic system solver strategy consists of using the AFAC and AFACx algorithms developed by [4, 18]. The hyperbolic system solver, RIEMANN++, is based on TVD and ENO based higher order Godunov strategies based on the papers by [21, 22, 23, 24, 25, 26, 27].

The plan of the paper is as follows. We will not give extensive discussion of the class libraries mentioned above since those would each constitute several papers in and of themselves. But we will describe those facets of the libraries that have been useful for this work. In section 2 we describe OVERTURE and P++ and its role in AMR applications. In section 3 we describe the elliptic and hyperbolic solver parallelization strategy. In section 4 we describe AMR++. And in section 5 we discuss the availability of this software.

2 The OVERTURE Framework

The OVERTURE Framework [14] is an object-oriented environment for the development of numerical applications. Specifically it is a large collection of C++ libraries which work together and build upon each other. The OVERTURE Framework supports both serial and parallel architectures and generally abstracts the details of parallel execution. Overture contains many class libraries and will not be discussed in any great detail within this paper. OVERTURE is the subject of an other paper which is a part of this same conference [11]. Overture obtains its parallel support though the use of the A++/P++ class library internally. These class libraries are the principle mechanism by which the AMR++ applications we present in this paper are run in parallel.

2.1 The A++ and P++ array classes

A++ and P++ [19] are array class libraries for performing array operations in C++ in serial and parallel environments, respectively. P++ is the principle mechanism by which the OVERTURE Framework operates in parallel.

A++ is a *serial* array class library similar to FORTRAN 90 in syntax, but not requiring any modification to the C++ compiler or language. A++ provides an object-oriented array abstraction specifically well suited to large scale numerical computation. It provides efficient use of multidimensional array objects which serves to both simplify the development of numerical software and provide a basis for the development of parallel array abstractions. P++ is the *parallel* array class library and shares an identical interface to A++, effectively allowing A++ serial applications to be recompiled using P++ and thus run in parallel. This provides a simple and elegant mechanism that allows serial code to be reused in the parallel environment. With the improvements in C++ compiler technology, the A++/P++ classes are presently being converted to the use of templates which provides greater flexibility, though at the likely cost of working with fewer C++ compilers.

P++ provides a data parallel implementation of the array syntax represented by the A++ array class library. To this extent it shares a lot of commonality with FORTRAN 90 array syntax and the HPF programming model. However, in contrast to HPF, P++ provides a more general mechanism for the distribution of arrays and greater control as required for the multiple grid applications represented by both the Overlapping Grid model and the Adaptive Mesh Refinement (AMR) model. Additionally, current work is addressing the addition of task parallelism as required for parallel adaptive mesh refinement.

The use of P++ is a central part of the mechanism by which we develop the adaptive elliptic solvers. Though the actual multigrid solvers are more complex than the example code below, this example of Jacobi relaxation for Poisson's equation shows some of the general mechanism's used to execute the array statements used in the multigrid elliptic solvers for serial and parallel architectures. Notice how the Jacobi iteration for the entire array can be written in one statement.

```
  // Solve u_xx + u_yy = f by a Jacobi Iteration
Range R(0,n)                              // ... define a range of indices: 0,1,2,...,n
floatArray u(R,R), f(R,R)                 // ... declare two two-dimensional arrays
f = 1.; u = 0.; h = 1./n;                 // ... initialize arrays and parameters
Range I(1,n-1), J(1,n-1);                 // ... define ranges for the interior

for (int iteration=0; iteration<100; iteration++)
  u(I,J) = .25*(u(I+1,J)+u(I-1,J)+u(I,J+1)+u(I,J-1)-f(I,J)*(h*h)); // ... data parallel
```

More detail on the mechanism for the execution of the above code can be found in [11] within this conference proceedings.

2.2 Role of P++ within AMR

The P++ array class library is the principle mechanism by which the parallelism is exploited within our work. P++ provides several features that are focused on providing support for multiple grid applications generally and adaptive mesh refinement as a specific case.

- Object-Oriented Dynamic Distribution Support
  The object-oriented distribution support occurs in two parts, first in the processor subsetting mechanism to manipulate the distribution objects, and second in the management of multiple array objects though a single distribution object. This provides a dynamic mechanism by which the load balancing can interact with an AMR application independent of the number of array objects in use or the specifics of the application. This provides a valuable means of abstraction for the redistribution associated with load balancing, plus a simplified means of controlling large numbers of grids which occur within many applications using the OVERTURE Framework.

- Mixed Distribution Operations
  Operations between array objects with different distributions occur naturally within AMR applications. Such operations are a general part of the transfer of data between levels within the adaptively refined grid. P++ supports array operations regardless of the distribution of the operands in the array statement. This feature of P++ is fundamental in supporting complex data transfer between AMR levels in the parallel environment.

There are different ways of using P++ depending on the user's requirements and level of expertise with parallel programming. P++ can be used to represent all aspects of the numerical algorithms within an application or can be used in combination with FORTRAN or C code external to P++. This permits the use of existing single grid codes to simplify the move to related adaptive mesh refinement applications.

3 Applications

3.1 Elliptic Solver

For the solution of elliptic equations on adaptively refined grids we use the most efficient solution methods presently available. The work on the Asynchronous Fast Adaptive Composite Grid Method (AFAC) [1, 4] and the more efficient AFACx [2, 18] variation of it represent parallel adaptive multigrid like methods, the most efficient parallel methods for solution of large elliptic systems. There are many variations on use of multigrid techniques in the solution of adaptive refinement grid applications many are active research areas within both serial and parallel algorithms.

The fast adaptive composite grid method (FAC) is an algorithm that uses uniform grids, both global and local, to solve partial differential equations. This method is known to be highly efficient on scalar or single processor vector computers, due to its effective use of uniform grids and multiple levels of resolution of the solution. On distributed memory multiprocessors, such methods benefit from their tendency to create multiple isolated refinement regions, which may be effectively treated in parallel. However, they suffer from the way in which the levels of refinement are treated sequentially in each region. Specifically, the finer levels must wait to be processed until the coarse-level approximations have been computed and passed to them; conversely, the coarser levels must wait until the finer level approximations have been computed and used to correct their equations.

The asynchronous fast adaptive composite method (AFAC) eliminates this bottleneck of parallelism. Through a simple mechanism used to reduce inter-level dependence, individual refinement levels can be processed by AFAC in parallel. The result is that the convergence rate of AFAC is the square root of that for FAC. Therefore, since both AFAC and FAC have roughly the same number of floating point operations, AFAC requires twice the serial computational time as FAC, but AFAC may be *much* more efficiently parallelized.

3.2 Hyperbolic Solver

As explained in the introduction the hyperbolic system solvers that we focus on are of the TVD and ENO types. The ones implemented here are the algorithms in [25, 26, 27]. While multidimensional in their flux update their implementation has a strong dimension by dimension bias. Such solvers have been implemented in parallel using message passing strategies but such strategies are not of much use here. For programming languages such as CRAFT and HPF that offer a contiguous logical grid in a distributed memory environment [28] have shown how to do implicit domain decomposition to parallelize such solvers. Their paper offers an extensive collection of strategies and results. In that paper they present two different strategies for parallelization. The first one is based on an SPMD-like approach that is available in CRAFT. The other is based on being able to easily redistribute data – a capability available in HPF. In tests the CRAFT-based strategy outperformed the HPF strategy by a slim margin. For that reason we focus on that strategy here. P++ can support either strategy though. P++ offers logically contiguous grids as well as an ability to redistribute data. The paper by [28] gives several details. Here we explain the gist of the idea in the next paragraph and then show how it is implemented in P++.

Higher order Godunov schemes have a huge amount of work associated with updating a zone. This work consists of very computationally intensive interpolation and equally complex Riemann solvers. Typically this work is done on a dimension by dimension basis. Thus one dimensional data is extracted and passed on to a sequence of Fortran routines that do the interpolation and Riemann solver steps. After extensive processing the one dimensional data is handed back. The choice to use Fortran is one that is based on efficiency. In the code presented below we show how a y-directional sweep is performed in parallel in two dimensions. On processor base and bounds are extracted using P++ functionality. Then one escapes to an SPMD programming style where the data is locally extracted, processed by Fortran subroutines and then handed back. Other details such as timestepping and boundary updates are omitted.

```
main ()
{
int nx = 128, ny = 256, numberOfProcessors = 32, ghost_width = 2;

// A simple partitioning is specified here with ghost boundaries
Partition_Type Partn ( numberOfProcessors);
Partn.SpecifyDecompositionAxes ( 2);
Partn.SpecifyInternalGhostBoundaryWidths ( ghost_width, ghost_width);

int ixbegin, ixend, iybegin, iyend, ix, iy, timestep;

// Here we make and distribute "arr2d_1 and arr2d_2".
floatArray arr2d_1 ( nx, ny, Partn), arr2d_2 ( nx, ny, Partn);

// We want to have a copy of "my_copy_1d_arr" on all processors.
// Notice that it is small enough to just cover local portion of array
// with associated ghost cells.
floatSerialArray my_copy_1d_arr ( 36);

for ( timestep = 1; timestep <= 100; timestep++) {

// Each processor figures out its own local patch of data. It does
// so in the global index space of the 2d arrays.
ixbegin = arr2d_1.getLocalBase ( 0); ixend = arr2d_1.getLocalBound ( 0);
iybegin = arr2d_1.getLocalBase ( 1); iyend = arr2d_1.getLocalBound ( 1);

Optimization_Manager::setOptimizedScalarIndexing(On); // Turn off communication

  for ( ix = ixbegin; ix <= ixend; ix++) {

  // i.e. we want to load that processor's copy of "my_copy_1d_arr" with
  // one strip from the patch of 2d data it owns.

    for ( iy = iybegin - ghost_width; iy <= iyend + ghost_width; iy++)
    my_copy_1d_arr ( iy - iybegin + 1) = arr2d_1 ( ix, iy);

 // this function does lots of float point intensive work on "my_copy_1d_arr".
    Lots_Of_Work ( iybegin, iyend, ghost_width, &my_copy_1d_arr);

    for ( iy = iybegin; iy <= iyend; iy++)
       arr2d_2 ( ix, iy) = my_copy_1d_arr ( iy - iybegin + 1);

  }  // End "ix" loop.

Optimization_Manager::setOptimizedScalarIndexing(off); // Turn on communication

// Update ghost boundaries.
arr2d_2.updateGhostBoundaries ();
arr2d_1 = arr2d_2;

}  // End timestep loop.
}  // End main.
```

4 Adaptive Mesh Refinement - AMR++

Adaptive mesh refinement is the process of permitting local grids to be added to the computational domain and thus adaptively tailoring the resolution of the computational grid. This results is greater computational efficiency but is difficult to support. AMR++ is a library within the OVERTURE Framework [14] which builds on top of the previously mentioned components and provides support for OVERTURE applications requiring adaptive mesh refinement. AMR++ is current work being developed and supports the adaptive regridding, transfer of data between adaptive refinement levels, parent/child/sibling operations between local refinement levels, and includes parallel AMR support. AMR++ is a parallel adaptive mesh refinement library because it is uses OVERTURE classes which derive their parallel support from the A++/P++ array class library.

5 Software Availability

AMR++ is not yet available publicly, however both A++/P++ and the OVERTURE Framework have been publicly available for several years. Future releases of the OVERTURE Framework will include the AMR++ class library. These are available from http://www.c3.lanl.gov/cic19/teams/napc/.

References

[1] S. McCormick, *Multilevel adaptive methods for partial differential equations*, SIAM, Frontiers in Applied Mathematics, Vol. 6, Philadelphia 1989.

[2] S. McCormick, D. Quinlan, *Idealized Analysis of Asynchronous Multilevel Methods*, Preceedings of American Society of Mechanical Engineers, Winter Annual Meeting, Annahiem, CA, November 8-13, Adaptive, Multilevel and Hierarchical Computational Strategies, AMD-Vol. 157, pg. 1-8, 1992

[3] D. Balsara, M. Lemke, D. Quinlan, *AMR++, a C++ Object Oriented Class Library for Parallel Adaptive Refinement Fluid Dynamics Applications*, Proceedings of American Society of Mechanical Engineers, Winter Annual Meeting, Annahiem, CA, November 8-13, Adaptive, Multilevel and Hierarchical Computational Strategies, AMD-Vol. 157, pg. 413-433, 1992

[4] S. McCormick, D. Quinlan, *Asynchronous Multilevel Adaptive Methods for Solving Partial Differential Equations on Multiprocessors: Performance Results*, Parallel Computing, Vol. 12, 1989, pp. 145-156.

[5] Lemke, M., Quinlan, D., *Fast Adaptive Composite Grid Methods on Distributed Parallel Architectures*, Proceedings of the Fifth Copper Mountain Conference on Multigrid Methods, Copper Mountain, CO, April 1991. Also in *Communications in Applied Numerical Methods*, Vol. 8, No. 9, pg. 609-619, Sept 1992

[6] M. Lemke, D. Quinlan, *P++, a C++ Virtual Shared Grids Based Programming Environment for Architecture-Independent Development of Structured Grid Applications*, CONPAR/VAPP V, September 1992, Lyon, France; published in *Lecture Notes in Computer Science*, Springer Verlag, September 1992.

[7] R. Parsons, D. Quinlan, *Run-time Recognition of Task Parallelism within the P++ Parallel Array Class Library*, Proceedings of the Conference on Parallel Scalable Libraries, Mississippi State, 1993.

[8] R. Parsons, D. Quinlan, *A++/P++ Array Classes for Architecture Independent Finite Difference Computations*, Proceedings of the Second Annual Object-Oriented Numerics Conference, Sunriver, Oregon, April 1994.

[9] S. McCormick, D. Quinlan, *Multilevel Load Balancing*, Internal Report, Computational Mathematics Group, University of Colorado, Denver, 1987.

[10] D. Quinlan and M. Berndt, *MLB: Multilevel Load Balancing for Structured Grid Applications*, in this proceedings, SIAM, 1997.

[11] D. L. Brown, G. S. Chesshire, W. D. Henshaw, and D. Quinlan, *OVERTURE: An Object-Oriented Software System for Solving Partial Differential Equations in Serial and Parallel Environments*, in this proceedings, SIAM, 1997.

[12] M. J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comp. Phys., 82 (1989), pp. 64–84.

[13] K. D. Brislawn, D. L. Brown, G. Chesshire, and J. S. Saltzman, *Adaptive composite overlapping grids for hyperbolic conservation laws*, LANL unclassified report 95-257, Los Alamos National Laboratory, 1995.

[14] K. D. Brislawn, D. L. Brown, G. S. Chesshire, and D. J. Quinlan, *Overture code*, 1996. Los Alamos National Laboratory Computer Code LA-CC-96-04.

[15] G. S. Chesshire, *Overture: the grid classes*, LANL unclassified report 96-3708, Los Alamos National Laboratory, 1996.

[16] ——, *Grid, GridFunction and Interpolant classes for Overture, AMR++ and CMPGRD, user guide, version 1.00*, LANL unclassified report 96-3464, Los Alamos National Laboratory, 1996.

[17] ——, *Mappings for Overture: A description of the mapping class and documentation for many useful mappings*, LANL unclassified report 96-3469, Los Alamos National Laboratory, 1996.

[18] D. Quinlan, *Adaptive Mesh Refinement for Distributed Parallel Processors*, PhD thesis, University of Colorado, Denver, June 1993.

[19] ——, *A++/P++ manual*, LANL Unclassified Report 95-3273, Los Alamos National Laboratory, 1995.

[20] ——, *A++/P++ class libraries*, 1996. Los Alamos National Laboratory Computer Code LA-CC-96-01.

[21] B.vanLeer, *Towards the Ultimate Conservative Finite Difference Scheme IV*, J. Comput. Phys., 23 (1979), pp. 276-298.

[22] P. L.Roe, *Approximate Riemann Solvers, Parameter Vectors and Difference Schemes*, J. Comput. Phys., 43 (1981), pp. 357-372.

[23] A.Harten, S. Osher, B. Engquist and S. R. Chakravarthy, J. Comput. Phys., 71 (1987), pp. 231-303.

[24] C.-W. Shu and S. Osher, *Efficient Implementation of Finite Difference ENO Schemes*, J. Comput. Phys., 83, (1989), pp. 32-78

[25] D. S. Balsara, *Linearized Formulation of the Riemann Problem for Adiabatic and Isothermal MHD*, Astrophysical J., (1997), accepted.

[26] D. S. Balsara, *TVD Scheme for Adiabatic and Isothermal MHD*, Astrophysical J., (1997), accepted.

[27] D. S. Balsara, and C.-W. Shu *Evaluating Finite Volume ENO Schemes for MHD*, (1997), in preparation.

[28] D. S. Balsara, and R. Melhem *Implicit Domain Decomposition Strategy for Parallelizing Fluid Algorithms* , Computers in Physics, (1997), submitted.

[29] A. Brandt, *Multilevel Adaptive Solutions to Boundary Value Problems*, Mathematics of Computation, 31, (1977), pp. 333-390.

[30] M. J. Berger and J. Oliger, *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*, J. Comput. Phys., 53, (1984), pp. 482-512