

Non-Uniform Load Balancing of Adaptive Mesh Refinement Codes

Scott Morris

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112, USA.

Abstract

The traditional approach to dividing work among processors on a grid calculation is to use recursive bisection. This approach is useful when given a large, single domain. With adaptive mesh refinement, however, we are given the task of balancing multiple levels of refinement and irregular domains. Here, we propose to use a modified form of recursive bisection that can cut into any number of workload boxes of a specified size and produce a smaller leftover box. The result of such domain splits is computational domains that can be evenly distributed among the processors.

1 Introduction

In any large scale scientific computation massively parallel systems are the target goal. In order to achieve optimal usage of these parallel systems the performance of scientific codes must increase proportionally with the number of nodes in the system.

In order to take advantage of the parallel computing resource one must be able to divide pieces of the computation into semi-independent pieces. Communication can take place

This work was supported in part by DOE Grant W-7405-ENG-48

between pieces but should be kept to a minimum. If given such a problem, one of the most important components for the performance is balancing the computational workload among the processors. Thus, finding and maintaining an optimal load balance plays an integral role in obtaining scalable performance.

Typical large scale computations involve repeating the same computations across a large, uniform spatial domain. The exact same set of differential equations are being evaluated at each point on the 2D or 3D grid. In most scientific and engineering applications, however, a very small proportion of grid cells are of interest at any one time. The level of spatial and temporal resolution required at these points is far greater than the rest of the grid. Often much computational effort is wasted because of this. Examples of computations of this nature are propagating waves, particle simulations and gas dynamics problems.

Adaptive mesh refinement (AMR) is a technique whereby the degree of spatial and temporal refinement is variable and systematically controlled in software. With AMR, areas of interest such as the threshold of a shock wave and areas of large particle concentration are given a finer resolution and smaller time step in order to carry out the calculation with sufficient accuracy.

Using AMR creates a variable number of levels of refinement. Since these finer levels contain higher resolution they also are more computationally significant. It is therefore important to be able to balance the workload of all levels efficiently.

In addition to AMR issues, the workload in a particular grid cell is not always equivalent to all other grid cells. For instance, in particle simulations, the particles contain their own data and are not bound to the underlying grid structure. These particles interact with the grid and require a larger amount of memory and computational workload in areas of high

particle concentration. Thus, non-uniform workloads arise.

We present an algorithm that efficiently divides the workload of typical non-uniform, irregular AMR domains. Specifically, the work described here was done using SAMRAI (Structured Adaptive Mesh Refinement Applications Infrastructure) developed at Lawrence Livermore National Laboratory. The use of particles was facilitated by using Material Point Method [3] code developed at Utah as part of the C-SAFE (Center for the Simulation of Accidental Fires and Explosions) project [2].

2 Non-Uniform Load Balancing

When given a computational domain over which the computation cost is uniform, traditional strategies involve some type of domain decomposition and assignment in order to balance the workload among the processors. However, with the addition of Material Point Method to the SAMRAI infrastructure came the creation of a Particle Data type. As stated, these particles are free to move about the grid and thus irregular distributions of particles and thus computational workload are commonplace.

For such non-uniform distributions of workload the standard approach is to use recursive bisection to perform the domain decomposition rather than using a standard straight grid decomposition. This method is described by Berger and Bokhari [1] for numbers of processors that are powers of two and is known as binary decomposition.

Binary decomposition starts by computing the total workload along the longest spatial axis of the domain. It sums the amount of work at each point along this axis and creates a profile of the workload. We then start at either end of the domain moving one grid space at

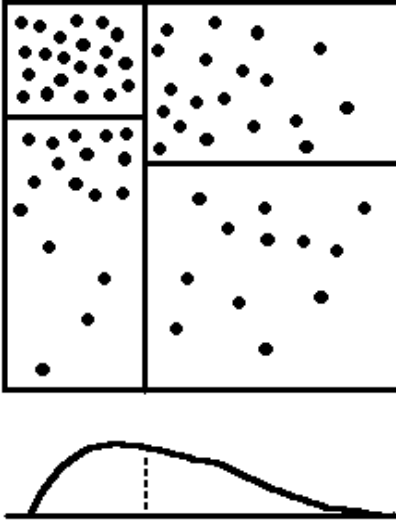


Figure 1: Typical binary decomposition of particle distribution with workload profile

a time and summing the workload. This process is continued until we reach the midpoint of the domain (there is approximately equal workload on each side). We mark this as a cut point and then proceed recursively among the two domains created by this cut point. An example binary decomposition for four processors with a profile curve along the x axis is shown in Figure 1.

Most of the computation (particles) in the figure is localized in the upper left corner which therefore has smaller boxes. It should be noted that although workload is approximately equal among the partitions, communication between the partitions is not. It should also be emphasized that this algorithm assumes a number of processors that is a power of two. In practice it is rare to be running a problem with processors not a power of two at the top level.

For AMR applications, however, at the lower levels of refinement you have a set of domains which cover a subset of the top level domain. These sub-domains are not spatially contiguous and may not be spatially continuous. It is here that the method of binary decomposition described above becomes inadequate. It was not designed to take into account that it may be called multiple times within a single load balance and across multiple sub-domains.

As an example of where traditional binary decomposition would fail, consider two non-connected leaf node boxes on the lowest level of the AMR hierarchy (highest level of refinement). One box contains twice as much workload as the other. Given four processors to load balance, we apply the algorithm to each of the two boxes (domains). Each box would be cut into four boxes which creates an excessive number of cuts and splits the data unnecessarily. The best assignment of boxes to processors will be to assign one sub-box from each original box to each processor. To take advantage of locality and to minimize overhead present in the box objects, this is not sufficient.

The application's total number of processors must be partitioned out to all of the sub-domains of the AMR hierarchy being load balanced. Our algorithm presents a way of efficiently partitioning the sub-domains using global workload information.

3 The Algorithm

Given the premise that we are distributing the workload resulting from multiple, non-contiguous and irregular domains we use a modified form of recursive bisection in order to get a reasonable load balance. An example domain (showing multiple levels of different

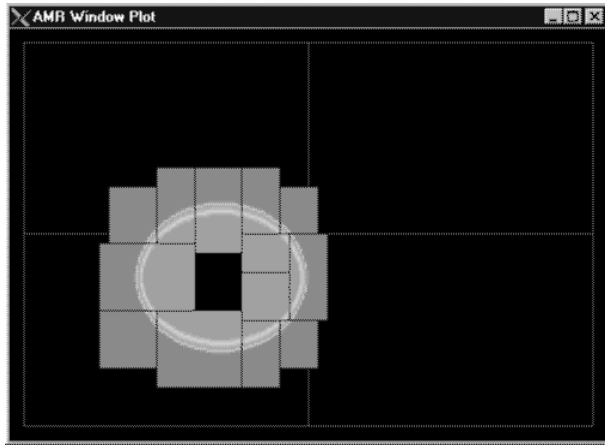


Figure 2: Linear advection plot, showing multiple AMR levels

refinement) is shown in Figure 2.

Black boxes represent the coarsest level of refinement, with the inner gray boxes representing the finest level. Each color of box in the example is represented by SAMRAI's box primitive. For the purposes of this algorithm we assume that the SAMRAI programmer has already provided a method for approximating the workload in a single box domain. In MPM applications, the workload can be approximated simply by returning the number of particles within one grid cell (box).

The algorithm proceeds by summing the total estimated workload in each box to compute the total (weighted) amount of work across the entire AMR framework. This number is then divided by the number of available processors to determine the ideal amount of work that should be assigned to each processor. A list of all boxes in descending order of estimated

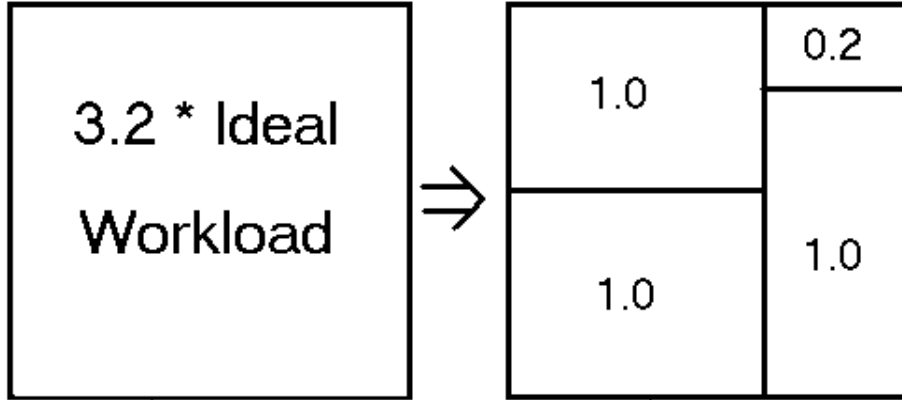


Figure 3: Sample modified recursive bisection decomposition

workload is also computed. This list is walked and modified recursive bisection is applied to each box whose estimated workload is greater than the ideal workload for a processor.

The form of recursive bisection used here functions similarly to binary decomposition which was described previously. Instead of looking for a cut point in the domain (a single box in our case) that creates a division of equal work on both sides, we use the pre-computed ideal workload. The algorithm attempts to find a cut point that will result in an integer multiple of the ideal workload on one side and an integer multiple of the ideal workload on the other side plus a leftover amount. Each cut is also subject to the qualification that these integer multiples be as close to each other as possible so that irregularly shaped domains are not created.

As an example, if we are given that a box contains 3.2 times the ideal workload for a processor (as in Figure 3), the algorithm will attempt to chop the box into 4 boxes, 3 of size ideal workload and one leftover box containing 0.2 times the ideal. In this situation,

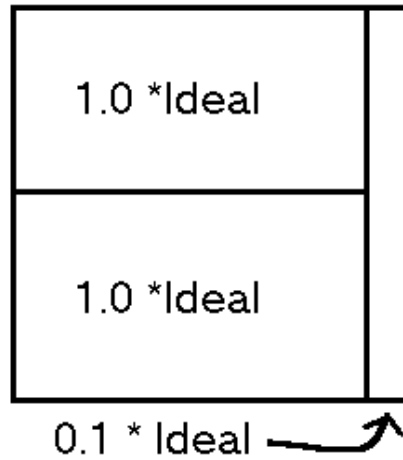


Figure 4: Example of irregular domain cut to be avoided

the first cut point would be chosen such that there is 2 times the ideal workload on one side and 1.2 times the ideal workload on the other. Therefore, the smallest cuts (into boxes of size ideal and the leftover boxes) are done at the lowest level of recursion. This is done so that irregular box formations such as that shown in Figure 4 do not arise and so that error can accumulate and be taken care of as the calculation proceeds. Since each cut point must be made on integer increments along an axis and because cut points are subject to various AMR constraints¹ they are rarely exactly a multiple of the ideal workload. So, as each cut is performed an error is accumulated and the size of the smaller, leftover box can be adjusted to compensate.

¹Example constraints are minimum and maximum box size, integer factorization of box dimensions and illegal cut points inherent to the particular AMR application.

4 Greedy Bin Packing

Once the algorithm has finished dividing any boxes larger than the best size to be assigned to one processor, the new list of boxes must be assigned to processors. This new list includes the chopped boxes which resulted from the modified recursive bisection as well as all original boxes that contained workloads less than the ideal amount for a single processor.

Our algorithm uses a simple greedy bin packing strategy to assign boxes to processors. This problem is the classic “knapsack” problem which is NP-complete and therefore an optimal solution is an infeasible goal. Furthermore, the load balancing routines are called every time step, making the speed of the load balancer a very important issue. It is most important to note that doing load balancing actually performs no calculation that is useful to the application—the load balancer is running only to speed up the calculation. The load balancer is also dynamic since it is called repeatedly throughout the calculation. Thus, it is imperative that the load balancer be fast. This is why we chose such a simple bin packing strategy. There are computationally more complex algorithms that give better solutions to the knapsack problem, but in practice, the greedy algorithm seems good enough.

5 Results

The algorithm has been extensively tested on the SAMRAI linear advection simulator. To generate nonuniform workload estimates, the sum of the variable “u” across each grid cell was used. This can be seen in Figure 2 where boxes that contain the interior and boundary of the advecting bubble contain more workload. The resulting estimated load balance for the Linear Advection code is shown in Table 1.

<i>Processors</i>	<i>Linear Advection 2D</i>	<i>Linear Advection 3D</i>
2	95.6449	93.2341
4	91.5862	90.3411
8	96.0993	92.0498
16	92.3900	89.4452
32	83.2528	72.9491

Table 1: Estimated Load Balance for Linear Advection Problem

The percentages listed in Table 1 are computed by dividing the ideal workload for one processor by the maximum amount of work assigned to any processor (after bin packing). A 100 percent estimated load balance represents a perfect balance (maximum work equals ideal work). This metric assumes that the workload estimate is accurate and second, that the performance lost by not assigning equal loads to each processor is proportional to the longest running (maximum estimated work) processor. The values in the tables are the average estimated load balance computed across the entire simulation run.

We can see that the algorithm performs very well for different numbers of processors as well as different levels of refinement. In practice, a 100 percent load balance is nearly impossible because there are a limited number of possible cut points and because of other AMR constraints. Any load balance of greater than 90 percent we consider to be acceptable performance.

Table 2 shows a comparison between our algorithm (with uniform load input) and SAMRAI's current uniform load balancer. It is important to note that our algorithm is also very useful for uniform AMR problems. Also, as the number of processors goes up the

<i>Processors</i>	<i>Modified Algorithm</i>	<i>SAMRAI Uniform</i>
2	97.5935	96.6953
4	95.0985	83.1212
8	91.3784	87.2528
16	85.9677	77.4738
32	77.5770	70.6321

Table 2: Comparison of uniform load balancing routines

estimated load balance numbers tend to decrease mainly because of the AMR constraints in the Linear Advection problem require a minimum box size. Thus, neither algorithm is able to balance efficiently at high numbers of processors.

6 Conclusion

In this paper we have provided an algorithm for the division of computational workload across multiple levels of an adaptive mesh refinement code. We introduced the concept of *modified recursive bisection* that divides work using the heuristic of cutting domains in multiples of the ideal workload for a processor. Results on sample calculations show that this technique for non-uniform load balancing has merit. In combination with a spatial bin packing routine, our algorithm will be integral for obtaining scalable performance on such applications.

A spatial bin packing algorithm would have to consider communication cost and attempt to assign neighboring domains to the same or close processors. This kind of routine would be

very application specific, or at least API specific. The algorithm presented here is application and API independent. It was developed within the SAMRAI infrastructure, but contains nothing SAMRAI specific.

Acknowledgment

We would like to thank Scott Kohn and Rich Hornung from Lawrence Livermore National Laboratory for the use of their SAMRAI software and for general ideas and support. An allocation of computer time from the Center for High Performance Computing at the University of Utah is also gratefully acknowledged. CHPC's SGI Origin 2000 system is funded in part by the SGI Supercomputing Visualization Center Grant. This work was part of the DOE's ASCI Alliance C-SAFE grant at the University of Utah.

References

- [1] Marsha J. Berger and Bokhari Shahid H. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, May 1987.
- [2] T. Henderson et al. Simulating accidental fires and explosions. *Computing in Science and Engineering*, 2(2):64–76, March/April 2000.
- [3] D. Sulsky, Z. Chen, and H.L. Schreyer. A particle method for history-dependent materials. *Computational Methods for Applied Mechanical Eng.*, 118:179–196, 1994.

List of Figures

1	Typical binary decomposition of particle distribution with workload profile	4
2	Linear advection plot, showing multiple AMR levels	6
3	Sample modified recursive bisection decomposition	7
4	Example of irregular domain cut to be avoided	8