15th US Army Symposium on Solid Mechanics
Myrtle Beach, South Carolina
April 12-14, 1999

ADAPTIVE MESH REFINEMENT IN CTH

David Crawford, Dept. 9232, P.O. Box 5800, Sandia National Laboratories,
Albuquerque, NM 87185-0820, dacrawf@sandia.gov.

Abstract

This paper reports progress on implementing a new capability of adaptive mesh refinement into the Eulerian multimaterial shock-physics code CTH. The adaptivity is block-based with refinement and unrefinement occurring in an isotropic 2:1 manner. The code is designed to run on serial, multiprocessor and massive parallel platforms. An approximate factor of three in memory and performance improvements over comparable resolution non-adaptive calculations has been demonstrated for a number of problems.

Introduction

Adaptive mesh refinement (AMR) has been widely hailed for improving computational resolution when resources are limited and has been used for hyperbolic problems on an experimental basis for years (Berger and Oliger, 1984; Berger and Colella, 1989, Tang and Scannapieco, 1995; Jones, 1997). For a mature Eulerian multi-material shock-physics code family like CTH and its predecessors, adaptivity is considered a natural next step in code development. It is a capability that allows full exploitation of today's limited computational resources and efficient use of tomorrow's substantial resources. Here we report on recent code development activity that is implementing an adaptive mesh refinement strategy in CTH otherwise known as AMRCTH. The final milestone of this work will be the distribution of AMRCTH to the CTH user community.

Algorithms

In order to achieve adaptivity yet retain the man-years of effort that have been expended on the development of physics routines, the refinement is block-based, where each block consists of a small patch of cells (typically 10x10x10 or 20x20x20). In order to facilitate its implementation on massively parallel platforms, each block communicates with its neighbors using a generalization of the message-passing paradigm developed for multiprocessor CTH (Fig. 1). This requires a more restrictive data structure than that used by Berger and Colella (1989). Like Berger and Colella, a maximum 2:1 resolution difference across block boundaries is strictly enforced and refinement and unrefinement occur isotropically. Unlike Berger and Colella,

DISCLAIMER

however, blocks are not allowed to overlap. In multiprocessor AMRCTH calculations, blocks are distributed across processors and CPU loading is optimized.

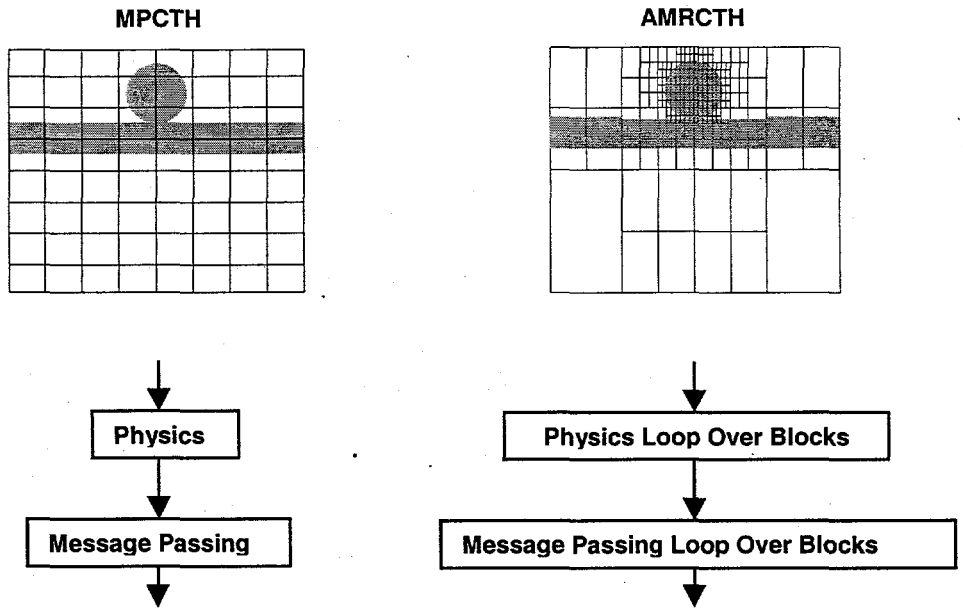

Fig. 1. In multiprocessor CTH (MPCTH as it is sometimes called) the computational domain is divided equally, one block per processor and messages are periodically passed to update the boundaries between domains. Cell spacing is required to match identically across processor boundaries. In AMRCTH, blocks are allowed to have a 2:1 mismatch in cell size across block boundaries (cell spacing must be aligned, however). Multiple blocks per processor are allowed. Messages passed between blocks are refined or unrefined to match the resolution of the receiving block.

Many times throughout the course of each explicit time step, the code must update the information at the boundary of every mesh block. In multiprocessor CTH, this takes the form of an explicit message being passed from one processor to another (Fig. 1). In AMRCTH, this has been generalized to allow inter-block message passing on *or* off processor. For communication between blocks of equal resolution, the form of the message is the same as in multiprocessor CTH. For communication between blocks of different resolution, a different approach is needed (Fig. 2). The data in the sending block are first transformed using second order unrefinement or refinement techniques (see Littlefield *et al.*, 1999) with the results placed in a temporary dummy block. The dummy block, acting as an alias for the sending block, sends the message that the receiving block expects to 'see'. This way, only the minimum required data are sent, message-passing bandwidth is reduced and memory requirements for the reception arrays are minimized.

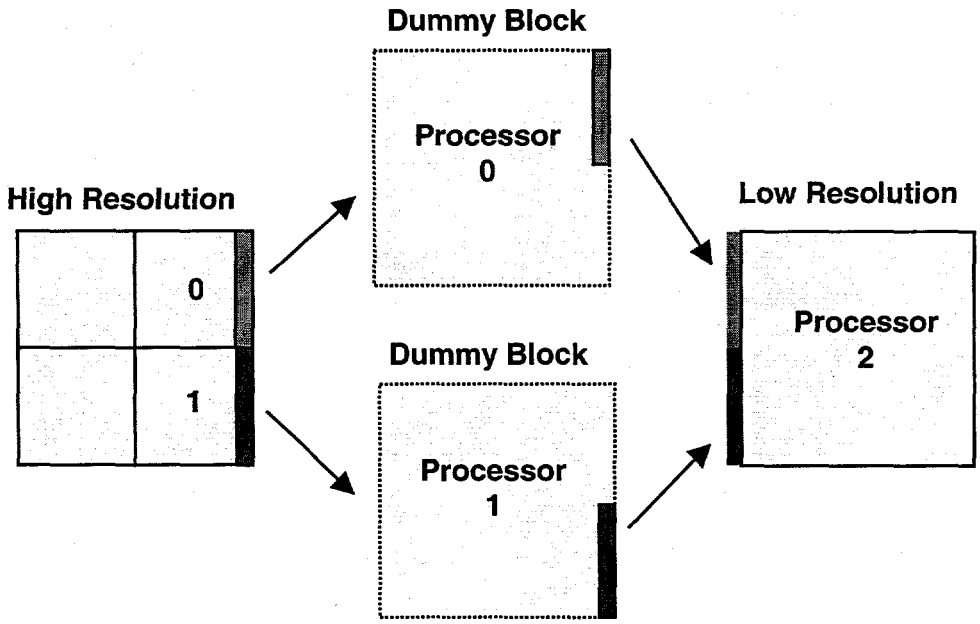

Fig. 2 Boundary updates between blocks of different resolutions use a dummy block during the creation of the 'send' message. The high resolution blocks on the left, located on processors 0 and 1, must send low resolution versions of their interior cells to the low resolution block on the right, located on processor 2. Processors 0 and 1 create dummy blocks with low-resolution versions of the contents of the high-resolution blocks. Half length messages (quarter length in 3D) are created and sent from the dummy blocks, received by the low resolution block in the space set aside for the reception of one full length message, and copied into the ghost cells.

Presently, a single time step, usually determined by the Courant condition in the highest resolution mesh, is used to advance the problem through time. At first glance, advancing the solution of the lower resolution mesh with the same time step as the highest resolution mesh may seem wasteful. A closer look, however, suggests that most computational cells are often at the highest resolution (Fig. 3) and the potential for improving performance with sub-cycling the highest resolution mesh must be offset by the additional algorithmic complexity. Additional research will be conducted to determine if the advantages of sub-cycling certain calculations can outweigh the costs.

Refinement Indicators

Research into refinement and unrefinement indicators is ongoing. The simplest approach is to use non-zero velocity as a refinement indicator and empty mesh as an unrefinement indicator. The 2D examples shown in Figs. 3-5 use this approach. By also allowing unrefinement of single material blocks (such as detonation products) we can effectively model explosions. The 3D examples shown in Figs. 6 and 7 use this approach. While this guarantees robust calculations, and is an improvement over non-adaptive calculations, it is far from achieving maximum efficiency. Littlefield *et al.* (1999) have developed a more aggressive refinement indicator that will be incorporated into AMRCTH. By incorporating user-definable refinement indicators such as his into a global error histogram, we hope to develop robust computational steering techniques.

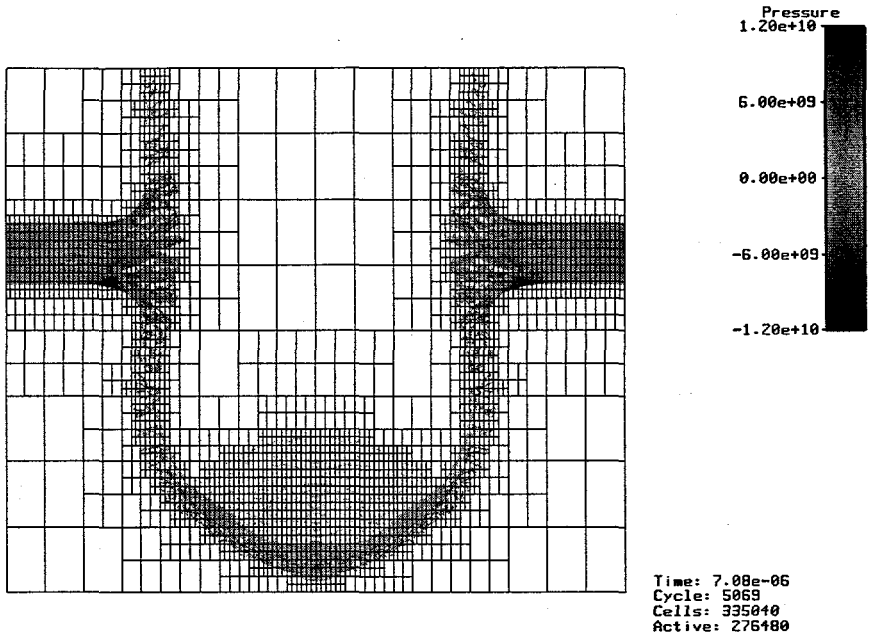

Fig. 3. In this 2D calculation of a copper ball striking a steel plate, 83% of the cells (276,480 out of 335,040) are at the highest resolution. The outline of each block (12x20 cells per block) is shown. To perform this calculation without AMR would require 987,140 cells. Including ghost cells, the AMR calculation requires 429,968 cells.

Memory utilization and performance

A moderate AMR calculation run on a workstation or a small number of processors can use several thousand blocks. A large problem run on a massively parallel computer such as ASCI Red can use tens of thousands of blocks. To measure the performance and memory utilization of some typical AMR calculations, two suites of calculations were performed. One suite consisted of a series of 2D bumper shield calculations performed on a large memory workstation. Here, message passing between blocks was implemented as memory-to-memory copies, representing relatively little overhead. Another suite, consisting of 3D spherical explosion calculations, was performed on the ASCI Red massively parallel computer where message passing was a real consideration.

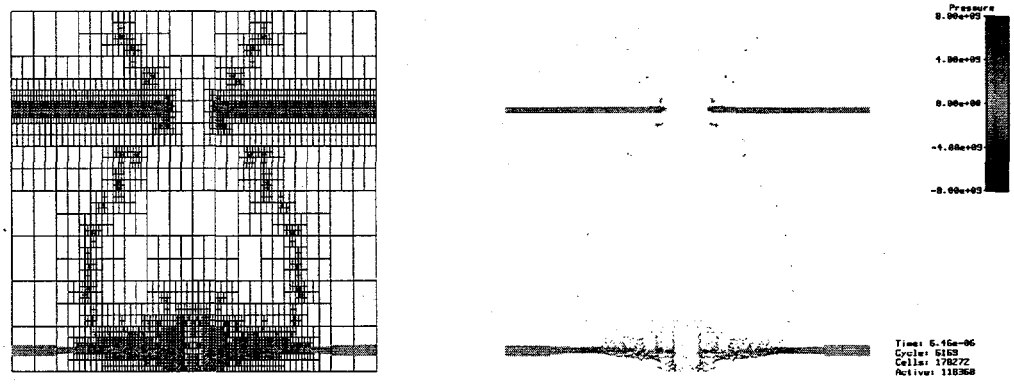

Fig. 4. 2D calculation of a copper ball striking a steel bumper shield. On the left, the outline of each block (6x12 cells per block) is shown. To perform this calculation without AMR would require 1.2 million cells. Including ghost cells, this seven level AMR calculation requires 277,312 cells.

Figure 4 shows a 2D calculation of a copper ball striking a steel bumper shield at 4.52 km/s. The calculation was performed for several resolutions denoted by the maximum allowed level of refinement. Each increase in refinement level represents a decrease of cell size by a factor of two. Equivalent non-adaptive calculations were performed with a 'flat' mesh of equal zone spacing. Each calculation was performed on a Sun Ultra 60 workstation with 1 Gbyte of memory. The memory requirements and CPU times for each of the adaptive and non-adaptive cases are plotted in Fig. 5. For every additional level of refinement, a non-adaptive calculation exhibits typical 2D scaling – increasing memory requirements by four and CPU time by eight (squares). For every additional refinement level, the AMR calculation (triangles) requires three times the memory and six times the CPU – effectively scaling as 1.5D.

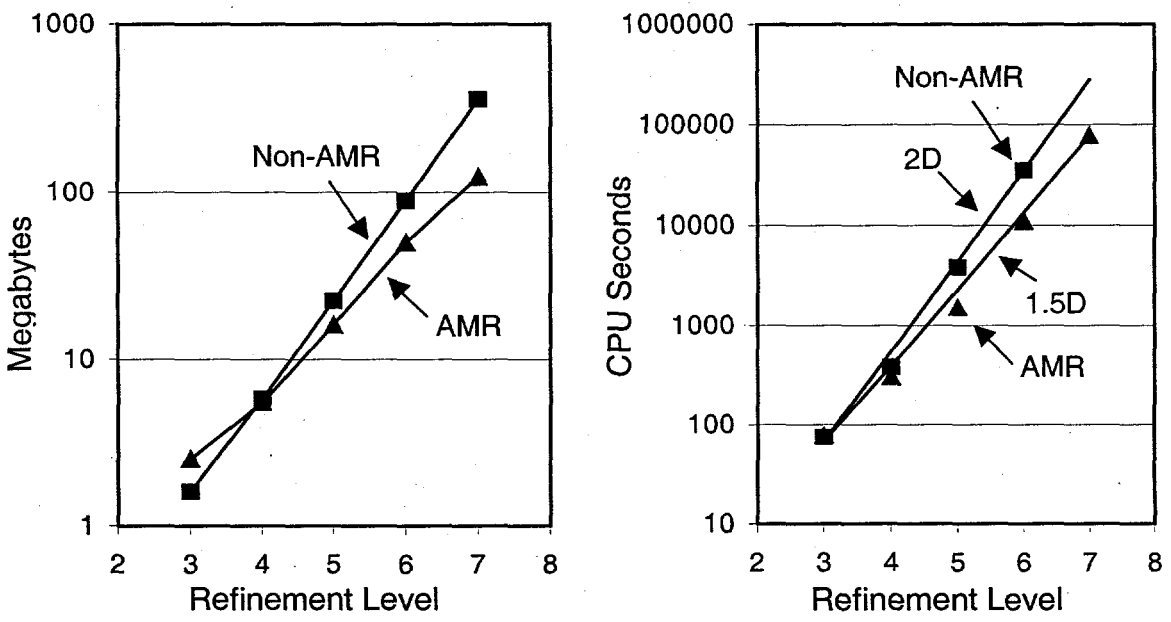

Fig. 5. Memory utilization and performance of the 2D bumper shield calculation shown in Fig. 4.

Figure 6 shows a 3D calculation of a spherical explosion of hot, dense air into air at STP. The calculation was performed for several resolutions denoted by the maximum allowed level of refinement. Each increase in refinement level represents a decrease of cell size by a factor of two. Depending on resolution, each calculation was performed on 16, 64, 384 or 952 processors of ASCI red. The memory requirements and CPU times for each of the adaptive cases are plotted in Fig. 7. For every additional level of refinement, a non-adaptive calculation will exhibit 3D scaling (extrapolated from a low-resolution 64-processor 'flat' calculation)–increasing memory requirements by eight and CPU time by sixteen (squares). For every additional refinement level, the AMR calculation (triangles) requires 5.5 times the memory–effectively scaling as ~2.4D. While the lower resolution test cases exhibit no significant performance improvements, the highest resolution case (level 7) exhibits an improvement by nearly a factor of two over the projected non-adaptive case.

The present version of the code exhibits substantial bookkeeping overhead in the algorithm that enforces the 2:1 rule. As much as 50-70% of CPU time is spent in just one routine. We see no reason why this cannot be substantially improved and consider reducing this to insignificance a major near term goal. The light gray curve in Fig. 7 reflects this ideal case.

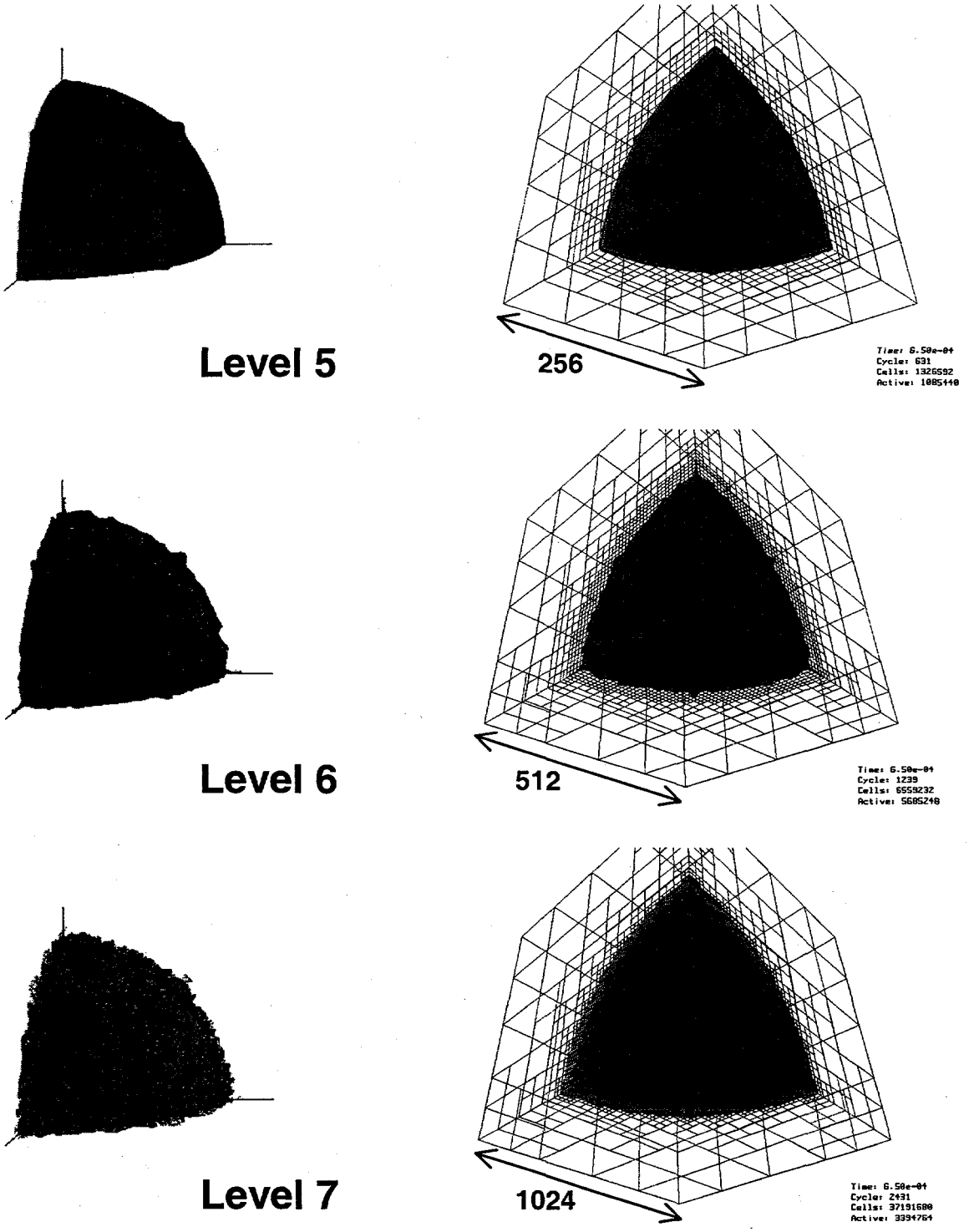

Fig. 6. 3D spherical explosion test problem. On the left, the interface between an octant of hot dense air is shown expanding into air at STP. Three reflecting boundary planes are used. On the right, the viewpoint is rotated and block outlines are shown. Calculations using three different maximum levels of refinement are shown. The Level 5 calculation is equivalent to a 256^3 cell non-AMR calculation, Level 6, 512^3 cells and Level 7, 1024^3 cells. These calculations were performed on ASCI Red with 64, 384 and 952 processors respectively.

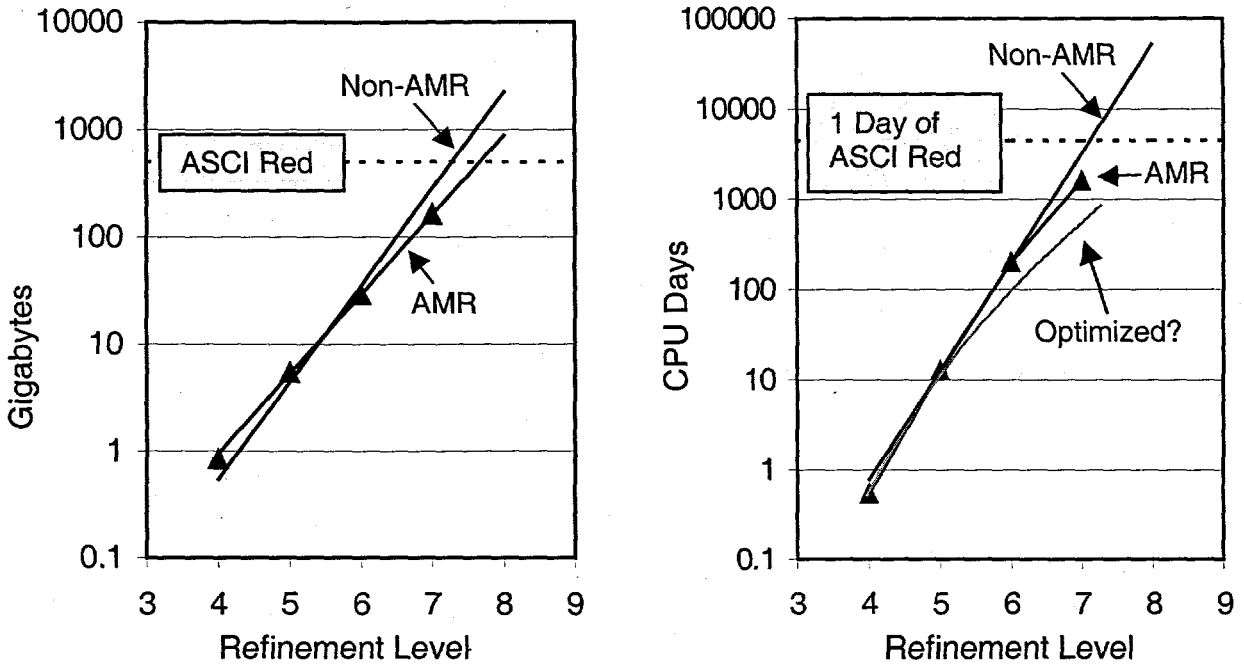

Fig. 7. Memory utilization and performance of the 3D spherical explosion calculation shown in Fig. 6. For every additional level of refinement, a non-AMR calculation exhibits typical 3D scaling – increasing memory requirements by eight and CPU time by sixteen. For every additional refinement level the AMR equivalent calculation (triangles) requires 5.5 times the memory– for nearly 2D (2.4D?) effective scaling. Substantial CPU is spent performing bookkeeping (see text). The gray curve at right indicates ideal performance assuming that bookkeeping CPU can be reduced to near zero.

Conclusions

With the preliminary version demonstrated here, block-based AMR appears to be a practical extension to CTH. At least a factor of three improvement in performance and memory utilization is achievable and, there is evidence of improved scaling for large problems. Two-dimensional problems begin to scale as if they were 1.5D and three-dimensional problems begin to scale as ~2.4D. While message passing is extensive in parallel AMR CTH, this cost appears to be overcome by the benefits that AMR provides; although, this may be true due to the high performance of the ASCI Red communications hardware. Further work is needed to evaluate performance on other parallel architectures. The present version of the code exhibits poor parallel bookkeeping performance, with as much as 50-70% of CPU spent on enforcing the 2:1 refinement rule; hence, there is substantial room for improvement in algorithm efficiency.

Acknowledgements: We would like to thank David Littlefield for his ongoing collaborative effort on second order refinement multi-material techniques and refinement indicators. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. Funding provided in part by the DOD/DOE Office of Munitions Memorandum of Understanding.

References

Berger, M. J. and J. Oliger (1984), Adaptive Mesh Refinement for Hyperbolic Partial-Differential Equations, *J. Comp. Phys*, **54**, 484-512.

Berger, M. J. and P. Colella (1989), Local Adaptive Mesh Refinement for Shock Hydrodynamics, *J. Comp. Phys*, **82**, 64-84.

Jones, B. (1997), SHAMROCK – an adaptive, multi-material hydrocode, in *International Workshop on new Models and Numerical Codes for Shock Wave Processes in Condensed Media, St. Catherines College, Oxford, UK, 15-19 September 1997.*

Littlefield, D. L. *et al.* (1999), this volume.

Tang, P.K and A. J. Scannapieco (1995), Modeling Cylinder Test, in *Shock Compression of Condensed Matter, Proceedings of the Conference of the American Physical Society Topical Group on Shock Compression of Condensed Matter held at Seattle, Washington, August 13-18, 1995,* 449-452.