LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-CONF-208312

Parallel Implementation and Scaling of an Adaptive Mesh Discrete Ordinates Algorithm for Transport

L. H. Howell

December 1, 2004

NECDC 2004
Livermore, CA, United States
October 4, 2004 through October 7, 2004

Parallel Implementation and Scaling of an Adaptive Mesh Discrete Ordinates Algorithm for Transport

Louis H. Howell*

*Lawrence Livermore National Laboratory

Block-structured adaptive mesh refinement (AMR) uses a mesh structure built up out of locally-uniform rectangular grids. In the BoxLib parallel framework used by the Raptor code, each processor operates on one or more of these grids at each refinement level. The decomposition of the mesh into grids and the distribution of these grids among processors may change every few timesteps as a calculation proceeds. Finer grids use smaller timesteps than coarser grids, requiring additional work to keep the system synchronized and ensure conservation between different refinement levels. In a paper for NECDC 2002 I presented preliminary results on implementation of parallel transport sweeps on the AMR mesh, conjugate gradient acceleration, accuracy of the AMR solution, and scalar speedup of the AMR algorithm compared to a uniform fully-refined mesh. This paper continues with a more in-depth examination of the parallel scaling properties of the scheme, both in single-level and multi-level calculations. Both sweeping and setup costs are considered. The algorithm scales with acceptable performance to several hundred processors. Trends suggest, however, that this is the limit for efficient calculations with traditional transport sweeps, and that modifications to the sweep algorithm will be increasingly needed as job sizes in the thousands of processors become common.

Introduction

At NECDC 2002 I presented a block-structured adaptive mesh refinement (AMR) implementation of the discrete ordinates algorithm for single-group radiative transport, designed to be coupled to multifluid Eulerian hydrodynamics in the Raptor code *(Howell, 2003).* That paper showed that the AMR scheme can produce solutions to the radiative transport equation as accurate as those computed on a uniform fine mesh, but using fewer computational cells and less CPU time. The coupling of the radiation field to the fluid energy is closely based on an AMR algorithm for gray radiation diffusion presented in *(Howell and Greenough, 1999 and 2003).* Both the diffusion and discrete ordinate algorithms require solution of

the radiation equations in two different contexts: on a single refinement level, repeatedly, as part of a nonlinear implicit update loop; and less often as a multilevel solution coupling different levels of mesh refinement in order to ensure conservation of energy. (Additional history, tracing the AMR algorithm back as far as *(Berger and Oliger, 1984),* is given in the references cited above.)

The equations resulting from the discrete ordinates discretization are traditionally solved by iterated transport sweeps. These sweeps may not converge rapidly by themselves, but various schemes have been developed to accelerate convergence, including diffusion synthetic acceleration, e.g. *(Alcouffe, 1977), (Larsen, 1982),* multigrid methods *(Morel and Manteuffel, 1991),* and transport synthetic acceleration *(Ramone et al., 1997).* Note that these are only a few examples taken from a large body of liturature. Even with acceleration, though, a large fraction of execution time is still spent doing transport sweeps. For the transport solver in the Raptor code I am using conjugate gradient acceleration, so sweeps and inner products are the main computationally-expensive parts of the algorithm.

In *(Howell, 2003)* I discussed a strategy for parallelizing AMR transport sweeps, but with only minimal results as the method was still under development. This paper will briefly recap the strategy and then focus on scaling performance for the resulting code, including some bottlenecks that were encountered and eliminated. Some earlier results along these lines were presented in *(Howell, 2004).* In any cases where the background material in the next section is not clear, see these two earlier papers for more detailed explanations.

Algorithm Description

Reduced to essentials, the discrete ordinate discretization of the radiative transport equation yields the following linear system:

$$(\Omega_m \cdot \nabla) I_m + \sigma_t I_m = \frac{1}{4\pi} \sigma_s \sum_{m'} w_{m'} I_{m'} + R_m. \tag{1}$$

Here the Ω_m are unit vectors distributed over the unit sphere. These discrete directions, or ordinates, along with their associated weights w_m, are collectively referred to as the ordinate set for the angular discretization. The unknowns are the intensities I_m in the directions Ω_m. The cross sections σ and source term R_m collect various effects associated with emission, absorption, isotropic scattering, temporal discretization, and coupling with matter.

The transport term $(\Omega_m \cdot \nabla) I_m$ can be discretized in many different ways. Discretizations most suitable for use in AMR can be expressed in finite volume form as a balance in each cell between fluxes through cell faces and net production in the cell interior. These include such popular choices as step, diamond difference, step characteristic *(Matthews, 1999),* and corner balance *(Adams, 1997).* For all of these discretizations, transport sweeps solve the left hand side of (Eq. 1) exactly, starting

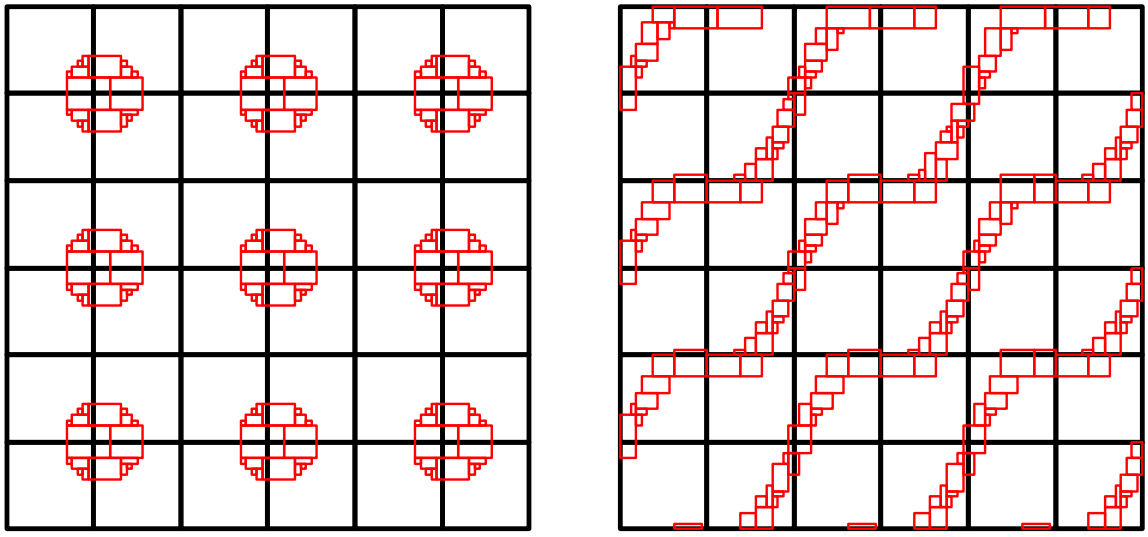

Figure 1: Two examples of AMR meshes, each with a single refined level. These are both artificial test problems designed for the parallel scaling study. The cases shown are for 36 processors—different sized problems are constructed using the same basic tiles. Each coarse grid is 256×256 cells, with fine grids refined by a factor of 2.

at the upsteam corner of the mesh for each ordinate m and proceeding downstream with respect to the direction Ω_m. The right hand side may then be updated, along with any reflecting boundary conditions, and the process iterated to convergence, or acceleration techniques may be applied.

The block-structured style of adaptive mesh refinement groups refined cells into rectangular grids, as shown in Fig. 1. These grids are the units of the parallel decomposition: each grid belongs to only one processor, though for load balancing a given processor is usually in charge of more than one grid. Multiple levels of refinement are possible so long as each is fully nested within the next coarser level. Refinement ratios are typically 2 or 4, and finer levels advance with proportionally smaller timesteps than coarser levels. As a result of this, there is usually only one level active at a time, so the grids of each level are distributed as uniformly as possible over the entire processor set. There is no guarantee that coarse and fine grids in the same part of the domain will be assigned to the same processors. For parallel infrastructure the code is based on BoxLib *(Rendleman et al., 2000).*

It is the directional, upstream-to-downstream nature of transport sweeps that makes the algorithm a puzzle to parallelize. A sweep for a single ordinate direction begins only in those grids that have no upstream neighbors, leaving most of the processors idle. Once those grids have been swept, the grids in the next layer immediately downstream may begin, and so on. Fortunately, there are many ordinate directions. All ordinates in the same quadrant in 2D (the same octant in 3D) generate the same dependency relations among the grids. Activity moves through the mesh in a pipeline; as each grid completes one ordinate, it begins to

work on the next in sequence. A similar scheme for regular grids appears in *(Koch et al., 1992).*

Starting all four quadrants (or eight octants) at once, the calculation begins at all corners of the domain and moves inwards. Conflicts occur when the wavefronts meet in the center; I discuss heuristics for resolving these conflicts in *(Howell, 2003).* Dorr, et al. *(1996),* working on uniform rather than adaptive grids, take the less deterministic approach of allowing execution order to be determined by the arrival times of messages carrying boundary data from other processors.

For adaptive meshes, the AMR timestep algorithm requires both single-level and multilevel solutions to the transport equation. Similar data structures are employed in both cases. First, for each quadrant (or octant), the code breaks each level of grids up into "waves". The first wave consists of grids with no upstream neighbors, the second wave has only first wave grids upstream, and so on. Note that this process does not depend on the ordinate set to be used.

In 3D AMR meshes it is possible to form dependency loops, where each grid in a loop depends on one of the others. It is always possible to break such loops by splitting some grids into smaller pieces, and indeed it is always possible to do so using cutting planes perpendicular to the z-axis. (To see this, consider that with enough such cuts, any 3D grid layout may be reduced to a series of 2D layers, and no dependency loops are possible in 2D.) I have taken this approach in the present code since splitting a BoxLib grid in the z direction leaves two contiguous chunks of data and is therefore more efficient for the underlying numerical routines.

Once the grids are arranged in waves, the code works out which waves may execute which ordinates at the same time without overlapping. This process does depend on the ordinate set, or at least on the total number of ordinates being used. The sweep process is organized into "stages", where at each stage a grid sweeps no more than one ordinate direction, and the objective is to minimize idle time for each grid by minimizing the number of stages for sweeping the entire ordinate set. More details of this optimization process are discussed in *(Howell, 2003),* but clearly the lower bound (best case) is for the number of stages to equal the number of ordinates since then every grid is busy at every stage of the computation.

So to summarize, there are two distinct setup phases of the algorithm. The first involves setting up the "wave" structures, including detection of individual dependencies between grids, determination of the distance of each grid from the upstream edge of the dependency graph, and splitting of 3D grids if necessary to break dependency loops. This phase is independent of the ordinate set. The second phase is setting up the "stage" structures. This means working out a reasonably-optimal ordering of ordinate sweeps for each grid using the dependency information contained in the waves, and also initializing the message-passing data classes that will control the parallel communication as the algorithm executes.

Neither setup phase actually touches any data, they just set up dependency and communication patterns. On a single refinement level, a sweep then consists of

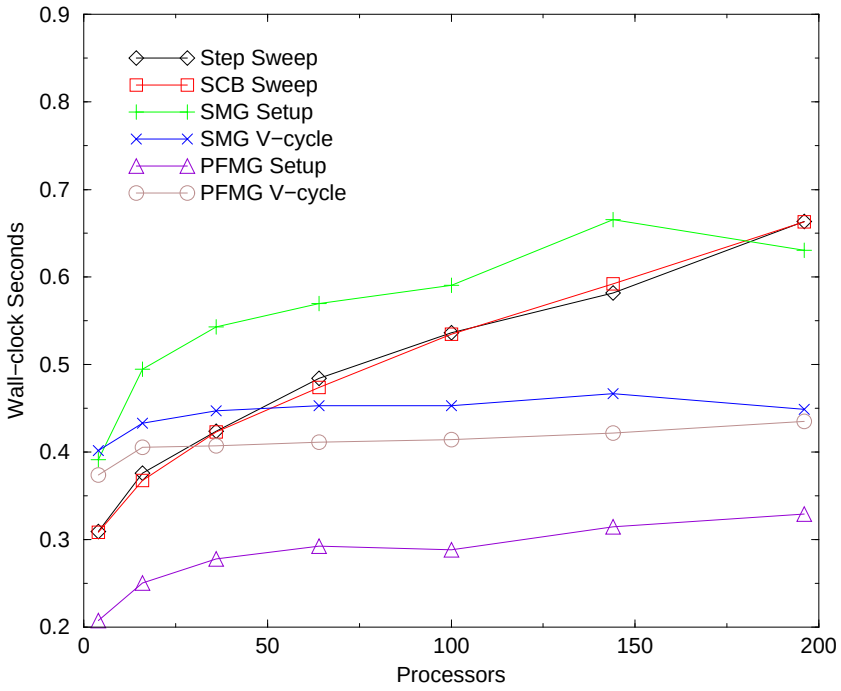

Figure 2: Timings for 2D grids arranged in a square array, one grid per processor, each grid is 400×400 cells. S_n transport sweeps (Step and SCB) are for all 40 ordinates of an S_8 ordinate set.

simply executing the preassigned stages in order, and accumulating scattering and reflection sources for use in the following sweep. For a multilevel problem, the algorithm first sweeps through the coarsest level, then continues to each of the finer levels using incoming fluxes interpolated from the coarser levels. Outgoing fluxes at the edges of fine levels are accumulated and passed back to the coarser levels for use as an "AMR correction source", or "refluxing source", on the next sweep. Each multilevel sweep therefore consists of a sweep on each active level, plus the necessary parallel communication between these levels. This multilevel sweep algorithm is described in more detail in *(Howell, 2004)*.

Parallel Scaling

All of the timings given in this paper were generated on the MCR machine at LLNL (2.4 GHz Xeon processors, 2 processors per node, Quadrics QsNet Elan3 interconnect). Older performance information on the IBM SP2 machine ASCI Blue Pacific was included in *(Howell, 2004)*.

Figures 2 and 3 show scaling results for simple domain decomposition with uniform grids. Varying numbers of grids are arranged in a square in 2D, or a cube in 3D, with one grid per processor in all cases. The transport sweep timings compare performance of the step approximation with the simple corner balance (SCB) algorithm *(Adams, 1997)*. The times given are for completion of sweeps in all

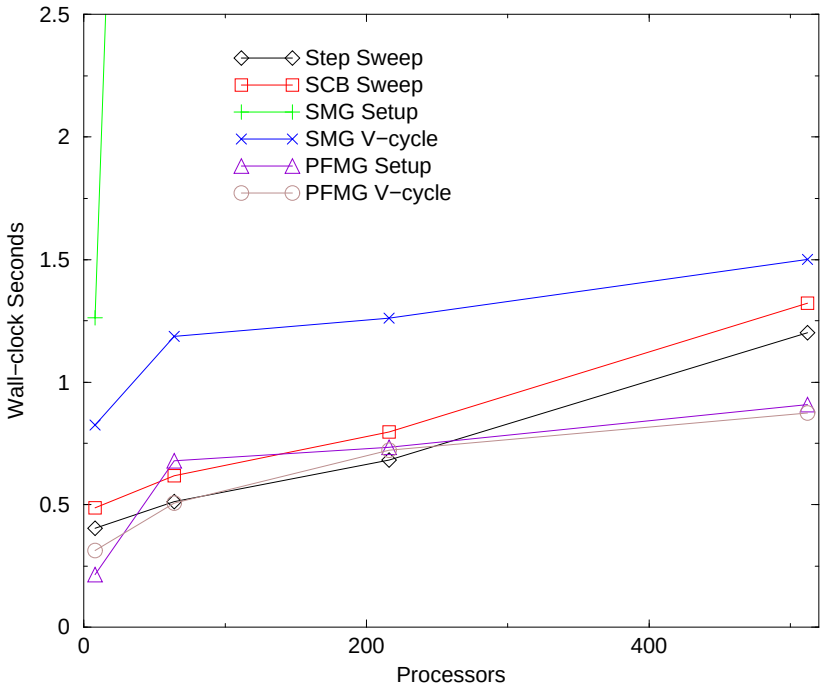

Figure 3: Timings for 3D grids arranged in a cubical array, one grid per processor, each grid is $40 \times 40 \times 40$ cells. S_n transport sweeps (Step and SCB) are for all 80 ordinates of an S_8 ordinate set.

of the ordinate directions in an S_8 ordinate set (40 ordinates in 2D, 80 in 3D). Step and SCB run at very similar rates, despite SCB being a more complicated and generally more accurate discretization. The implication is that memory access and parallel communication dominate the run times, and that once data has made it into local cache a few extra arithmetic operations add little incremental cost.

Also shown for comparison are the times for problem setup and a single V-cycle of two of the multigrid algorithms from the *hypre* library *(Chow et al., 1999),* which is used in the Raptor code for radiation diffusion calculations. Note that transport sweeps for a large number of unknowns (40 or 80) cost about the same as multigrid cycles for a diffusion equation in a single unknown. Most of the multigrid curves are flattening out, however, showing good scaling as both the size of the problem and the number of processors is increased. The transport curves on the other hand show a steady upward slope. For larger problems a larger number of stages is required for sweeping across the entire mesh, so scaling is not perfect even in theory. The results suggest that performance is acceptable for hundreds of processors but that some other form for the algorithm will be increasingly desirable for machines with thousands or tens of thousands of processors.

For both cases, transport and diffusion, the times shown are for the building blocks of iterative solvers, not the total time to obtain a solution. In general many transport sweeps or many V-cycles are required for convergence. The building blocks are deterministic, though, and independent of any problem details other than

the mesh layout. Timings for the complete solvers are more difficult to compare, as they depend on the coefficients of a particular system. See *(Howell, 2003)* for discussion of the largely orthogonal question of transport convergence rates and convergence acceleration.

Figures 4 and 5 show transport timings for the 2D AMR meshes from Fig. 1. Sweep times are given both for multilevel sweeps that couple both coarse and fine levels, and for fine level sweeps alone. Multigrid diffusion times are not included for these meshes since the focus of this paper is on transport, but the general patterns seen above for uniform grids are repeated for AMR grids: multigrid cycles for a single unknown cost about as much as transport sweeps for all ordinates in an S_8 ordinate set; the PFMG multigrid scheme scales well in both 2D and 3D; while the more robust SMG scheme scales poorly in 3D. *(Howell, 2004)* includes more scaling results for the multigrid algorithms on AMR meshes.

The main difference between the two tiling patterns shown in Fig. 1 is that in the pattern on the left the clusters of fine grids are all separate from each other, while in the pattern on the right they couple across the entire width of the domain. The effect of this can be seen in the fine sweep timings in Figs. 4 and 5. In Fig. 4 these curves flatten out since the number of stages remains constant, but in Fig. 5 the curves have an upward slope. The multilevel sweep timings have an upward slope in both figures since the coarse level is coupled in both cases.

Figures 6 and 7 extend these tests to 3D using similar patterns of grids (decoupled in one case, coupled in the other). The conclusions to be drawn are similar to those for the 2D tests. Runs for the coupled case on 512 processors could not be completed due to memory limitations.

The final two curves in each of the four AMR figures show the setup costs discussed in the previous section. These setup results are the unoptimized timings given in the oral presentation at NECDC 2004. In 2D these are obviously not scaling well, but given that each setup expense is amortized over many sweeps the cost is only becoming significant for the largest problems. In 3D, though, the setup costs are more serious, mainly due to the larger number of fine grids in the 3D AMR problems. In the following section I will tell what I have done since the conference to reduce these setup expenses to a more acceptable level.

Optimization of the Setup Phase

One of the key strengths of block-structured adaptive mesh refinement is the large amount of data associated with each regular grid. This means that the amount of information needed to describe the mesh is much smaller than the data, and mesh operations can usually be expressed in ways chosen for clarity and simplicity rather than for computational efficiency. It is only when a mesh expense begins to become an important component of the overall execution time, as in the 3D AMR timings for Figs. 6 and 7, that more attention to optimization becomes necessary.

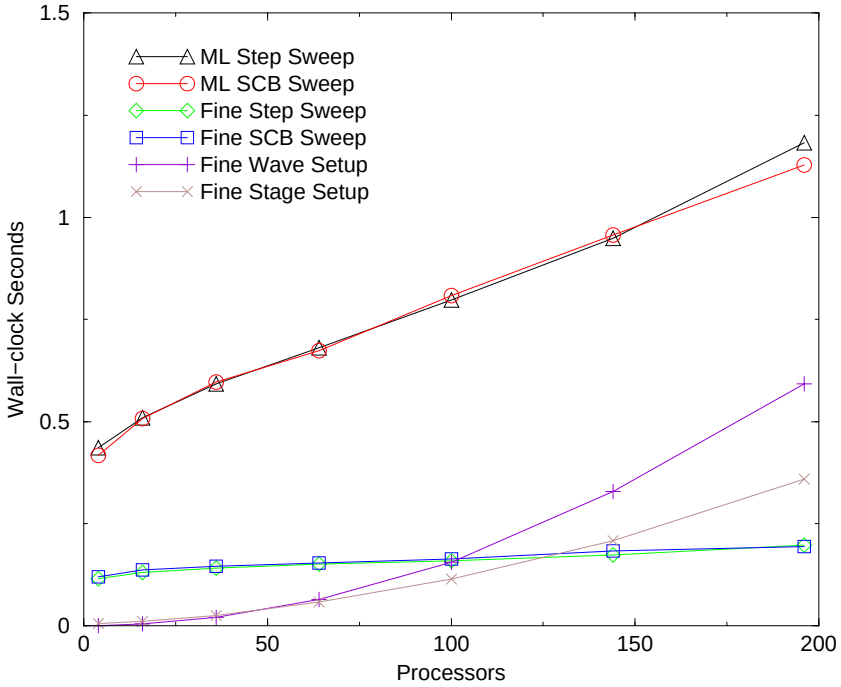

Figure 4: Multilevel sweep timings compared to sweeps on the fine level alone for grids arranged as shown on the left side of Fig. 1. In each run there are 4 coarse grids and 18 fine grids for every four processors. Each coarse grid is 256×256 cells. Fine level setup costs are included for comparison.

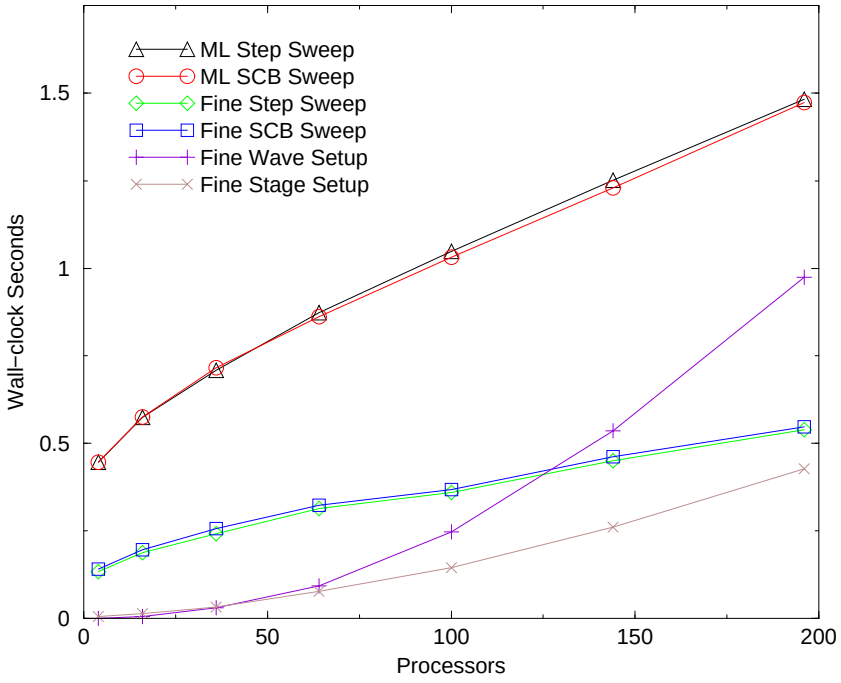

Figure 5: Multilevel sweep timings compared to sweeps on the fine level alone for grids arranged as shown on the right side of Fig. 1. In each run there are 4 coarse (256×256 cell) grids and a variable number of fine grids for every four processors. Fine level setup costs are included for comparison.

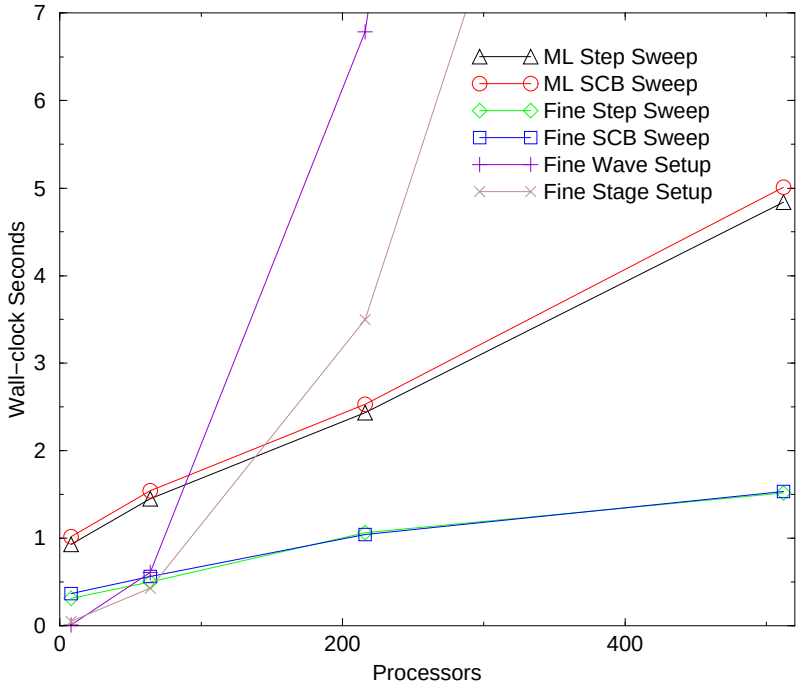

Figure 6: Multilevel sweep timings compared to sweeps on the fine level alone for 3D grids arranged in a manner similar to the left side of Fig. 1. In each run there are 8 coarse grids and 58 fine grids for every eight processors. Each coarse grid is $32 \times 32 \times 32$ cells. Fine wave and stage setup times for 512 procs are 37s and 18s.

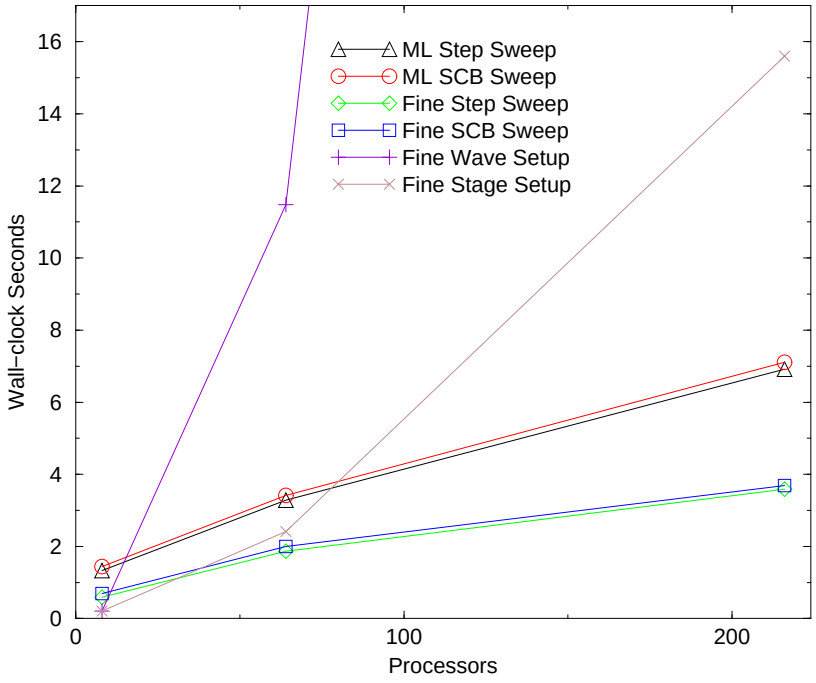

Figure 7: Multilevel sweep timings compared to sweeps on the fine level alone for 3D grids arranged in a manner similar to the right side of Fig. 1. In each run there are 8 coarse ($32 \times 32 \times 32$ cell) grids and a variable number of fine grids for every eight processors. Fine wave setup for 216 processors is off the scale at 131 seconds.

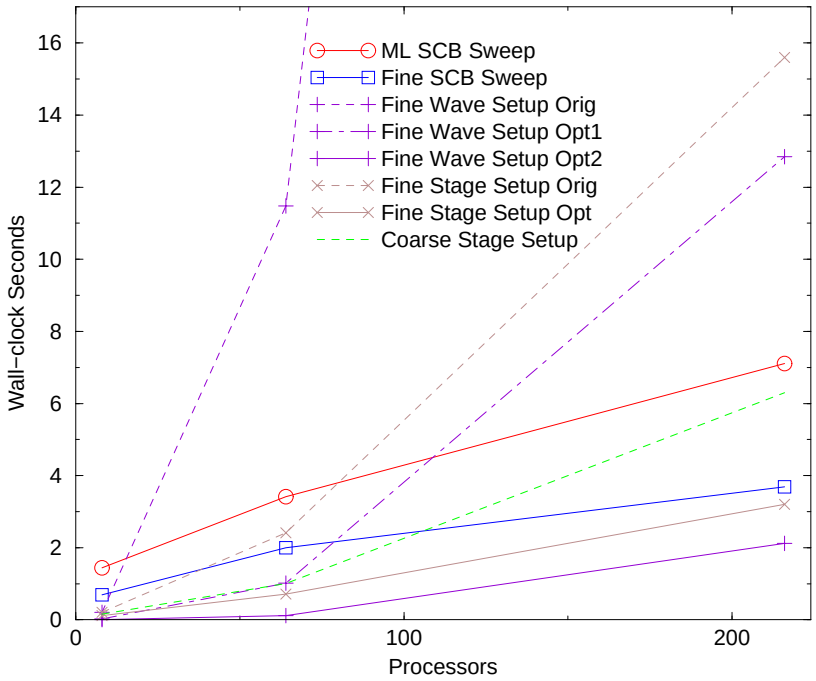

Figure 8: Sweep and setup timings for the 3D AMR test problem with coupled fine grids, Fig. 7. For the wave setup phase the original algorithm is shown along with two stages of optimization, while for stage setup the original algorithm is compared with one optimized version. Finally, the green line shows the time required for the coarse stage setup, which includes the arrangements for the parallel communication between coarse and fine levels. This step is not yet optimized, but the same technique used for the fine wave setup could be applied to it also.

Both components of the transport setup phase are primarily serial code: the wave setup is entirely serial, while in the stage setup the only parallel code is concerned with initializing the parallel data structures for the sweeps. There are ways that more of these steps could be parallelized, at least partially, but I have not found it necessary to do so. Only serial optimizations were required for the improvements presented in this section. Because Fig. 7 shows the worst setup performance, I reproduce it as Fig. 8 to show the effects of these changes.

The wave setup phase is concerned with dependency relationships between grids. It first generates for each grid a list of its neighbor grids in each direction. Then, for each direction, it determines how far each grid is from the upstream edge of the dependency graph (this distance is the "wavenumber" for each grid). The method for doing this is crude but simple to express: the code repeatedly loops over the grids; when a grid is found for which each upstream neighbor has a known wavenumber, the wavenumber for that grid is itself set accordingly. The number of passes required is not more than the maximum wavenumber for the entire mesh. In 3D there is additional coding added to detect and break dependency loops, but the basic algorithm is the same.

I first tried implementing a different scheme for the 2D case, that identified grids

at the upstream edge and then systematically traversed through the dependency graph, filling in wavenumbers as it went. Run times did not improve. The new algorithm was sometimes slightly faster than the old, brute force approach, sometimes slightly slower. I think the decreased number of passes over the grid list was offset by the added cost of the recursive graph traversal algorithm. I did not implement this variation in 3D, and so it does not appear in Fig. 8.

The true bottleneck in the wave setup turned out to be the first step, determining the lists of neighboring grids for each grid in each direction. This was coded in the simplest possible way, as an $O(N^2)$ comparison of every grid with every other grid, in each direction. I first recoded this more efficiently by reusing as much information as possible for the different directions, but keeping the basic $O(N^2)$ loop. The result was the line marked "Opt1" in Fig. 8, faster by a factor of 10 but still slower than the multilevel sweep.

For the line marked "Opt2" I eliminated the need for the $O(N^2)$ comparisons. I sorted the grids into bins, eliminating the need to compare to grids outside the neighboring bins, and reducing the cost to $O(N)$. The only drawback of this method is that it has a free parameter, the bin size, which must be chosen appropriately for a given grid layout. A bin size equal to the largest grid size in the problem ($32 \times 32 \times 32$) gave the result shown in the figure. The fine wave setup phase went from the most to the least expensive part of the calculation.

The stage setup determines a reasonably efficient allocation of ordinates to grids for each stage of the sweep process. The most expensive part of this involves comparing many "waves" of grids for different directions to see if they have any grids in common. The original method was to compare every grid in one wave to every grid in the other, an $O(M^2)$ process (but note that this M is the number of grids in a wave, not the total number of grids N in the problem). Optimizing this comparison process to $O(M)$, using the fact that each wave is sorted in increasing grid order, accounted for most of the improvement shown in Fig. 8. Storing the results of comparisons for reuse did not provide much additional benefit. Still, the fine stage setup time is now less than the time for a single fine sweep.

Finally, I include one more setup cost, that for the coarse stage setup. This curve is higher than the other setup costs in Fig. 8 since it includes setting up the parallel communications between coarse and fine levels. This step is not optimized at present, but could be improved with a bin sorting scheme similar to that used for the fine wave setup.

I conclude with Fig. 9, showing the effects of these optimizations on the 3D AMR test case with decoupled grids. There are no results for the intermediate level of wave setup optimization ("Opt1"), but otherwise the curves are the same as described for Fig. 8. In the 2D tests the optimized setup times dropped to the level of insignificance, so there seems no need to add additional plots for those cases.

I am not claiming that any of the optimizations presented in this section are particularly sophisticated. The point is rather to only do those optimizations that

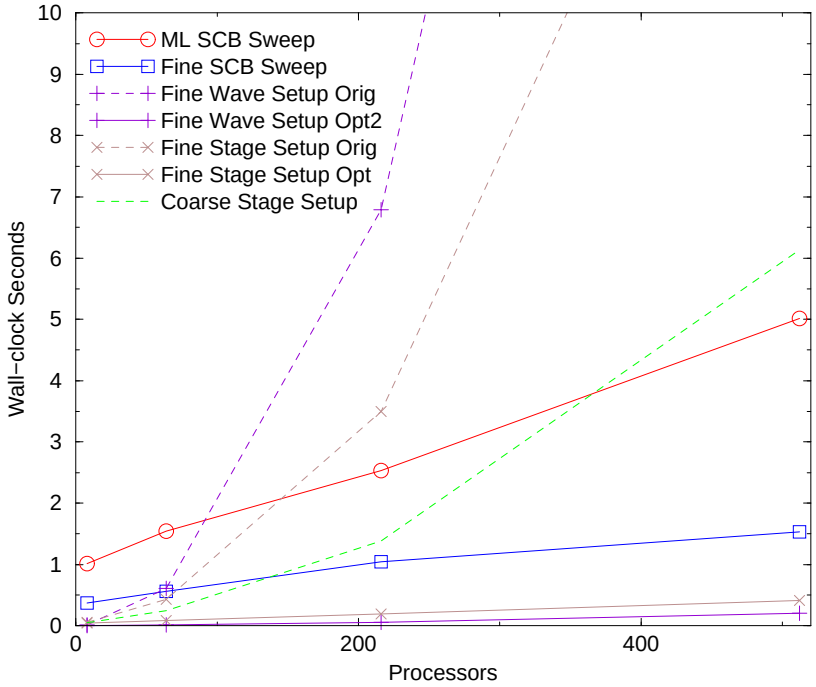

Figure 9: Sweep and setup timings for the 3D AMR test problem with decoupled fine grids, Fig.6. Effects of the setup optimizations are shown as in Fig. 8.

will significantly impact performance of the code as a whole. With block-structured AMR the data operations are fully parallelized, but the mesh operations generally are not. If it becomes necessary to parallelize the mesh setup operations, then it can be done, but so far it has not been necessary. Serial optimizations have sufficed for the fine wave and stage setup operations, and could also be applied to the coarse stage setup phase. The timing information suggests that the next step to improving performance will be in a different area entirely, exploring alternatives to the sweeping algorithm, which may in turn change the interactions with the convergence acceleration schemes.

Acknowledgements

This work was performed under the auspicies of the Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

References

Adams, M. L., "Subcell Balance Methods for Radiative Transfer on Arbitrary Grids,", *Transp. Theory Stat. Phys.*, **26**, 385–431 (1997).

Alcouffe, R. E., "Diffusion Synthetic Acceleration Methods for the Diamond-Differenced Discrete-Ordinates Equations," *Nucl. Sci. Eng.*, **64**, 344–355 (1977).

Berger, M. J., and Oliger, J., "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," *J. Comput. Phys.,* **53**, 484–512 (1984).

Chow, E., Cleary, A.J., and Falgout, R.D., "Design of the *hypre* Preconditioner Library," Proceedings of the SIAM Workshop on Object Oriented Methods for Interoperable Scientific and Engineering Computing, Yorktown Heights, NY, October 21–23, 1998 (1999).

Dorr, M. R., and Still, C. H., "Concurrent Source Iteration in the Solution of Three-Dimensional, Multigroup Discrete Ordinates Neutron Transport Equations," *Nuc. Sci. Eng.,* **122**, 287–308 (1996).

Howell L. H., "A Discrete Ordinates Algorithm for Radiation Transport Using Block-Structured Adaptive Mesh Refinement," Proceedings of the 12th Nuclear Explosives Code Development Conference, Monterey, CA, October 21–24, 2002 (2003).

Howell, L. H., "A Parallel AMR Implementation of the Discrete Ordinates Method for Radiation Transport," Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Chicago, IL, September 3–5, 2003 (2004).

Howell, L. H., and Greenough, J. A., "A Block-Structured Adaptive Mesh Refinement Algorithm for Diffusion Radiation," Proceedings of the 10th Nuclear Explosives Code Development Conference, Las Vegas, Nevada, October 25–30, 1998 (1999).

Howell, L. H., and Greenough, J. A., "Radiation Diffusion for Multi-Fluid Eulerian Hydrodynamics with Adaptive Mesh Refinement," *J. Comput. Phys.,* **184**, 53–78 (2003).

Koch, K. R., Baker, R. S., and Alcouffe, R. E., "Solution of the First-Order Form of Three-Dimensional Discrete Ordinates Equations on a Massively Parallel Machine," *Trans. Am. Nuc. Soc.,* **65**, 198–199 (1992).

Larsen, E. W., "Unconditionally Stable Diffusion-Synthetic Acceleration Methods for the Slab Geometry Discrete Ordinates Equations, Part 1: Theory," *Nucl. Sci. Eng.,* **82**, 47–63 (1982).

Mathews, K. A., "On the Propagation of Rays in Discrete Ordinates," *Nucl. Sci. Eng.,* **132**, 155–180 (1999).

Morel, J. E., and Manteuffel, T. A., "An Angular Multigrid Acceleration Technique for S_n Equations with Highly Forward-Peaked Scattering," *Nucl. Sci. Eng.,* **107**, 330–342 (1991).

Ramone, G. L., Adams, M. L., and Nowak, P. F., "A Transport Synthetic Acceleration Method for Transport Iterations," *Nucl. Sci. Eng.,* **125**, 257–283 (1997).

Rendleman, C. A., Beckner, V. E., Lijewski, M., Crutchfield, W., and Bell, J. B., "Parallelization of Structured, Hierarchical Adaptive Mesh Refinement Algorithms," *Comput. Visual. Sci.* **3**, 147–157 (2000).