LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-PROC-212154

Parallel Deterministic Neutron Transport with AMR

Christopher Clouse

May 11, 2005

Computational Method in Transport Workshop
Tahoe City, CA, United States
September 11, 2004 through September 16, 2004

This work was performed under the auspices of the U. S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng.48.

Parallel Deterministic Neutron Transport with AMR

C. J. Clouse[1]
[1]**Lawrence Livermore National Laboratory, Livermore CA clouse1@llnl.gov**

AMTRAN, a one, two and three dimensional Sn neutron transport code with adaptive mesh refinement (AMR) has been parallelized with MPI over spatial domains and energy groups and with threads over angles. Block refined AMR is used with linear finite element representations for the fluxes, which are node centered. AMR requirements are determined by minimum mean free path calculations throughout the problem and can provide an order of magnitude or more reduction in zoning requirements for the same level of accuracy, compared to a uniformly zoned problem.

1 Introduction

AMTRAN, a two and three dimensional Sn neutron transport code designed to run effectively on large parallel machines that have both distributed memory and shared memory parallelism, first began development in 1995 under an industrial partnership agreement with several oil well logging companies and LLNL. Due to shortened time lines and dwindling DOE funding in support of industrial partnership agreements, AMTRAN never became a major contributor in oil well logging calculations. However, it was recognized as an excellent R&D test bed for trying out new parallel algorithms and AMR techniques as applied to the Boltzmann equation. Development has continued over the years and, although the code and its algorithms have been presented at several conferences [1][2] this is the first publication of the basic code and it's techniques. At the time development began in 1995, the notion of applying AMR techniques to neutron transport problems was unexplored. Since that time, a number of other AMR neutron transport codes have been developed [3][4]. AMTRAN, however, remains unique in its combination of degrees of parallelism (MPI in space and energy and threading in angle) and its use of finite element node centered fluxes (the other referenced AMR codes use face or zone centered fluxes). As in the case of hydrodynamics, where spatial AMR was first developed, many transport applications have widely varying resolution needs within the same application. Stability and accuracy considerations require that spatial zoning be able to resolve length scales less than a neutron mean-free-path for most commonly used algorithms. This can be on the order of a millimeter in fissionable materials of nominal density to a meter or more in air. An example of particular interest in our work is the interrogation of cargo containers for fissionable material using a 14 MeV neutron source. The cargo container could be a semi-tractor trailer with the neutron source situated on one side of the trailer and a detector located on the opposite side. The goal is to be able to match the detector signal with known configurations of various types of fissionable materials. In this example, the distance between source and detector could be many meters; the material throughout most of which is probably air or some other material that is relatively transparent to neutrons, but the fissionable target would require good spatial

resolution. Spatial AMR allows us to get the needed resolution in the target without making the overall calculation unwieldy.

2 Code Overview

AMTRAN operates in 2D cylindrical and 3D cartesian geometries. Node centered fluxes are represented with continuous linear finite elements, similar to the methods employed by Greenbaum and Ferguson[5]. Angular discretization in 2D and 3D is with standard discrete ordinates and, therefore, requires half angle approximations to maintain acceptable conditions on the ordinates when finite differencing the angular derivative in cylindrical coordinates (see ref. 6 for a discussion of this topic). In 1D, a quadratic finite element approximation for the angular unknown has been implemented and provides significant computational savings over standard differencing (see ref. 7 for a detailed discussion). AMTRAN has several simple internal generators: nested spheres with point to point linearly interpolated densities, nested cylinders with constant densities and constant density cartesian blocks. It is also capable of reading COG[9] input and using the geometry generator routines in COG to construct a mesh.

2.1 Production of AMR blocks

The AMR algorithm in AMTRAN is block based and thus produces a hexahedral (quadrilaterals in 2D) block decomposition of the problem domain. Like previous AMR work in the field of hydrodynamics, e.g. Berger and Colella[8], the zone size in each direction is halved for each increase in level of refinement, unlike most hydrodynamic

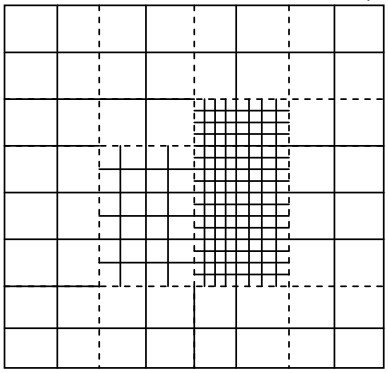

Fig. 1. 2D example where dashed lines indicate block boundaries.

AMR techniques, though, no level nesting is required, i.e. it is possible to have adjacent blocks differ by any power of 2 in zoning, as illustrated in fig. 1. Uniform zoning within a block allows fast and efficient computation of the transport equation.

The refinement criteria is based on neutron mean free path considerations,

$$h < \min\left(\sum_{g} a^{g} \lambda^{g} \right) \qquad (1)$$

where λ^g is the minimum neutron mean free path by energy group, h is the zonal width and a^g is a user defined multiplier, which can vary with energy group. Blocks are created by beginning at the finest level and tagging all zones that need to remain at that level. The tagged zones are boxed up using a "smart bisection" algorithm that can be briefly described as follows for 3D with the obvious extension to 2D.

Count up all the tagged zones in each 2D plane of the problem. Let y_i represent the number of tagged zones in plane *i*, then define f_i to be the discrete function

$f_i = y_i$

The splitting plan, i = isplit, is chosen such that

$$isplit = \max\left[\left|\frac{d^2}{dx^2} f(x_{i+1}) - \frac{d^2}{dx^2} f(x_i)\right|\right] \tag{2}$$

where the second derivatives are defined using a standard central difference formula,

$$\frac{d^2}{dx^2} f(x_i) \equiv f(x_{i+1}) - 2f(x_i) + f(x_{i-1}) \tag{3}$$

A block is subject to further splitting until it satisfies one of two conditions: 1) it no longer contains any tagged zones, in which case it is discarded, or 2) it satisfies the following condition,

$$\frac{number\ of\ tagged\ zones}{total\ number\ of\ zones} \geq irattag$$

where irattag is a user specified efficiency ratio. Generally, more and smaller blocks will be produced as the value of irattag is increased.

2.2 Sweeping the Mesh

The time-dependent transport equation can be written as follows,

$$\frac{1}{v_g}\frac{\partial}{\partial t}\Psi_m^g + \hat{\Omega} \bullet \vec{\nabla}\Psi_m^g + \sigma_t^g \Psi_m^g = \sum_l (2l+1) P_l \sum_{g'} \sigma_{lg\,g'} \phi_l^{g'} + \upsilon\sigma_f^g \phi + q \tag{4}$$

$$\equiv S$$

where

$\Psi_m^g =$ angular flux for energy group g and angle m,

$P_l =$ Legendre polynomial term *l*,

$\sigma_t^g =$ total cross section for energy group *g*,

$\sigma_{lg\,g'}$ = the l^{th} component of a Legendre polynomial expansion of the differential scattering cross section from group *g'* to group *g*,

v_g = the neutron velocity for group *g*.

$\phi_l^{g'} = \frac{1}{2}\sum_m P_l(\mu)\Psi_m^{g'}$ where m is the discrete angular index.

$\upsilon\sigma_f^g\varphi$ = represents the production of the scalar flux into group *g*, and q represents a possible, externally driven source term.

AMTRAN can solve eq. 1 as either a fixed source calculation (where q is non-zero) or the usual k eigenvalue calculation, where k is a multiplier on the fission source term, or α eigenvalue calculation where the time dependent term is included in eq. 1 and the time dependence is modeled as $e^{\alpha t}$. Eq. 1 is solved through standard source iteration and angular sweeps in which the source terms on the right hand side of eq. 1 are evaluated using the previous iterates values for the fluxes. Then, with the value of *S* determined,

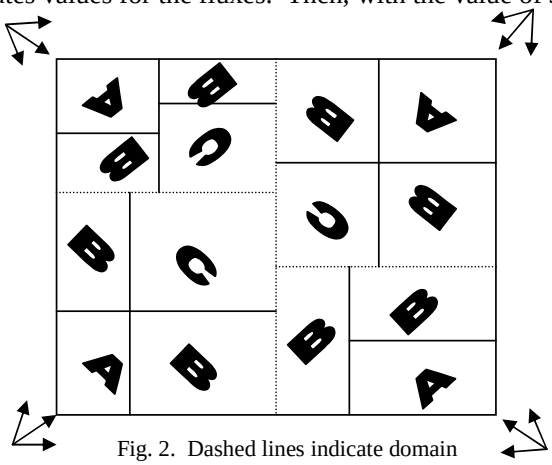

Fig. 2. Dashed lines indicate domain (and block) boundaries. Solid lines indicate block boundaries.

inversion of the sweeping term on the left hand side of eq. 1 is accomplished by sweeping through the mesh in the direction of neutron flow; one sweep for each unique combination of direction and energy group. This downwind sweeping is complicated by block decomposition on a domain decomposed mesh in which different domains reside on different processors. In order to avoid idling processors, AMTRAN's default domain decomposition is limited to 8 domains (4 domains in 2D). By ensuring that each domain includes one of the corners of the problem, all domains can immediately begin sweeping. Fig. 2 illustrates a simple 2D example assuming four spatial domains and 12 angles. Each block designated "A" can be swept immediately by any one of the three angles that originate from its corner. After any "A" block is swept, its neighboring "B" blocks would have sufficient boundary information to begin their sweep, followed by the "C" blocks, etc. A domain continues to sweep blocks until no more blocks can be swept without receiving information from neighboring domains, at which point it sends out all of its downwind boundary information to the necessary neighboring domains and waits to

receive upwind boundary information from any domains. AMTRAN assigns an estimated weight to each zone in the generator mesh based on the mean free path in that zone. This allows AMTRAN to estimate where to place domain boundaries such that each domain has roughly equal weight and, therefore, will have roughly equivalent zone counts after the AMR blocks are made. If each domain has roughly equivalent zone counts, then each will finish their sweeps at about the same time, pass information to and receive information from neighboring domains, and continue sweeping with little or no idle time. If there are reflecting boundaries in the problem, then the corners that lie on reflecting boundaries will begin their sweeps with "old" boundary information from the previous iteration. (Our definition of the term "iteration" is comparable to the standard textbook definition of an "outer iteration", which implies all angles have been swept through the entire mesh.) This causes some degradation in the rate of convergence, but the fractional increase in the number of iterations it takes to converge is usually substantially smaller than the relative speedup achieved by simultaneously beginning sweeps from all corners. For example, in 3D critical sphere calculations with one reflection plane, the number of iterations required to achieve convergence is about 20% more than the number of iterations required by ordering the sweeps such that reflecting boundaries are not swept until incoming sweep information is received, but the calculation will run roughly twice as slow by ordering the sweeps since half the processors will be idle at any given time. Because eight spatial domains (four in 2D) can be very limiting when attempting to scale up to thousands of processors, recent work in AMTRAN has focused on efficient use of processors with more than eight spatial domains. Basically, the idea for achieving high efficiency is through the use of domain overloading techniques. We have created the construct of a *domain master* which represents a unique collection of domains that may or may not be located contiguously in space. Domains are assigned to domain masters in such a way as to keep the domain master busy as much of the time as possible. Systematic algorithms have been worked out that asymptotically approach 100% theoretical efficiency as the number of domains per domain master is increased. The difference between theoretical and actual efficiency is dependent on how well the code is able to produce domains that are roughly equal in computational work, since the algorithm assumes equal weight domains. Details of the algorithm have been presented at an international conference[10], and will be outlined in a journal article in the near future.

2.4 Block Interfaces

As the sweeps proceed from block to block, three scenarios can occur at block boundaries: 1) no change in zoning, 2) go from a coarser mesh to a finer mesh, 3) go from a finer mesh to a coarser mesh. The first scenario obviously requires no special treatment. The second scenario can be dealt with in a straight forward fashion through bilinear interpolation of the coarse grid fluxes onto the fine mesh. This is consistent with the linear finite element representation of the fluxes at the nodes. Unfortunately, the finite element representation of the fluxes at the nodes does not provide an obvious

Comment [CC1]:

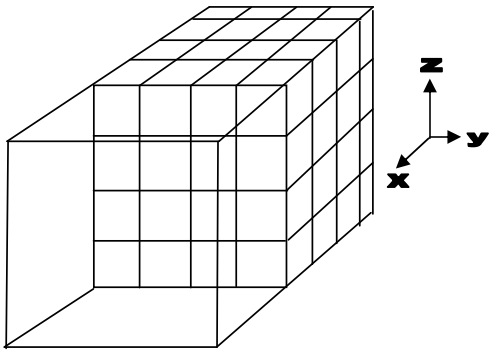

Fig. 3. A single zone from a coarse block that borders a finer block along it's x-face.

unique solution to the third scenario, which is illustrated in fig. 3. Over time, three different methods evolved in the code for treating scenario 3. The original method, referred to as the pseudo-source method, is constrained by two criteria: fluxes of nodes at the same physical location on two different mesh should have the same value and flux must be conserved across the interface. To satisfy the first criterion, we require that fluxes at nodes 1, 5, 21 and 25 in fig. 4 have the same value on the coarse and fine

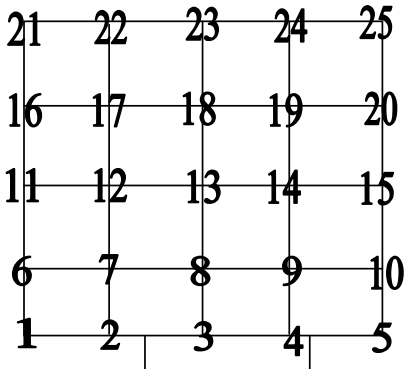

Fig. 4. Y-z interface of coarse to fine zone. Fine zone nodes are numbered 1 to 25. Coarse zone nodes are nodes 1, 5, 21 and 25.

blocks. If we integrate eq. 1 over a zone and focus on just the streaming term for the specific example shown in figs. 3 and 4 with neutrons traveling in the +x direction. The flux leaving the fine zones overlapping the coarse zone can be written as,

$$\sum_{i=1}^{25} \int_{y_0}^{y_0+\Delta y} \int_{z_0}^{z_0+\Delta z} w_{yi} w_{zi} \Psi_i \, dz dy$$

where, w_{yi} and w_{zi} are the y and z components of the linear finite element weight functions at node i, defined as

$$w_{yi} = 1 - \frac{(y_i - y)}{dyf} \quad \text{for } y_{i-1} \geq y \geq y_i, \qquad w_{zi} = 1 - \frac{(z_i - z)}{dzf} \quad \text{for } z_{i-1} \geq z \geq z_i$$

$$w_{yi} = 1 - \frac{(y - y_i)}{dyf} \quad \text{for } y_{i+1} \geq y \geq y_i, \qquad w_{zi} = 1 - \frac{(z - z_i)}{dzf} \quad \text{for } z_{i+1} \geq z \geq z_i$$

$$w_{yi} = 0 \quad \text{and,} \qquad w_{yi} = 0 \quad \text{elsewhere,}$$

and $dyf = dzf = ¼\ \Delta y$ where $\Delta y = \Delta z$ is the zone size on the coarse grid and the coordinates of node 1 are given by (y_0, z_0). Likewise, the flux entering the coarse zone can be expressed as

$$\sum_{i=1,5,21,25} \int_{y_0}^{y_0+\Delta y} \int_{z_0}^{z_0+\Delta z} w^c_{yi} w^c_{zi} \Psi_i \, dz dy \tag{5}$$

where, w^c_{yi} and w^c_{zi} are the y and z components of the linear finite element weight functions on the coarse zone at node i, defined as

$$w^c_{yi} = 1 - \frac{(y_i - y)}{\Delta y} \quad \text{for } y_i \geq y, \qquad w^c_{zi} = 1 - \frac{(z_i - z)}{\Delta z} \quad \text{for } z_i \geq z,$$

$$w^c_{yi} = 1 - \frac{(y - y_i)}{\Delta y} \quad \text{for } y \geq y_i, \qquad w^c_{zi} = 1 - \frac{(z - z_i)}{\Delta z} \quad \text{for } z \geq z_i.$$

At the time boundary information is received, the difference between eq. 2 and eq. 3 is

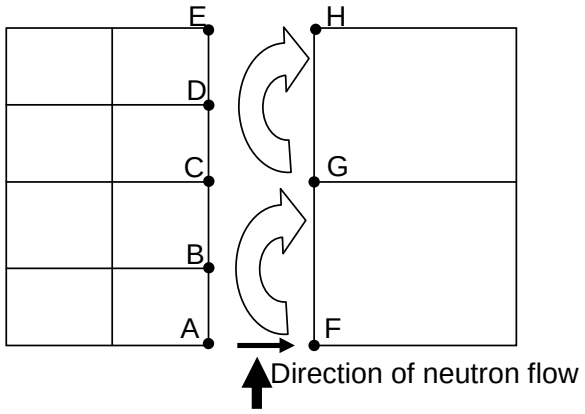

Fig. 5. Marching Method

calculated and stored as an additional zone centered source term that is included in the

solution of eq. 1 and, thus, from the perspective of nodes downwind from the boundary, the total flux crossing from the fine mesh to the coarse mesh has been accounted for through the inclusion of an additional source term, which we will refer to as the pseudo-source term. In a similar fashion, the difference between the coarse and fine mesh for the second term on the left hand side of equation 1, the absorption term, is also accumulated into the pseudo-source term. The second method is referred to as the *marching method* and is illustrated in fig. 5. In this method, the value of the most downwind node (node A in fig. 5) is copied to the corresponding physical node on the neighboring coarse grid (node F in fig. 5). The value of node G is then simply determined by flux conservation across the interface between nodes F and G. The value of node H is then determined by flux conservation across the interface between node G and H, etc.

The third method is referred to as the area weighting method and is illustrated with a 3D example in fig. 6. In our example, we have a coarse block (block I) partially overlapped by a fine block (block II) on the top face. For neutrons incident on the upper right front corner, node C would not only receive a flux contribution from block II (indicated by the shaded area) but also from two other blocks that overlap the two faces that are orthogonal

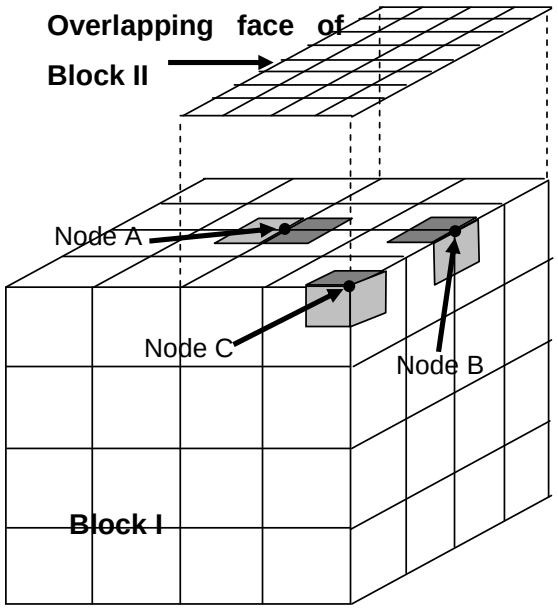

Fig. 6. Area Weighting Method

to the shaded face. The total contribution would be the area weighted sum of the three incident faces. A corner node, like node C, can be shared by up to 8 blocks (in 3D), each of which may have a different flux value for that node. Only in the case where all blocks sharing a node are at the same AMR level are we guaranteed that all blocks see the same physical value. In the case of node A, the lightly shaded area would contribute to the area weighted nodal flux, but if node A were located on the left edge of block I, then the

lightly shaded area would either be located on a separate block or, in the case of block I lying on the left edge of the problem, there would be no neighboring block. In either case, the lightly shaded area would not contribute to node A's value. Likewise, if the direction of neutron flow were from back to front, only the shaded face, representing block II's contribution, would contribute to node C's value. Thus, one can see that the area and number of faces contributing to a nodal flux value is dependent on the direction of neutron flow. Method 1 is difficult to implement in time-dependent problems and method 2 is prone to instabilities. Thus, the default method used in AMTRAN is method 3.

2.5 Energy Group Parallelism

Distributed memory parallelism (i.e. the number of MPI processes spawned) is equal to 8 (or 4 in 2D) x the number of energy groups per process. If the number of processes is not evenly divisible by 8 (or 4 in 2D) then the number of energy groups on a processor will vary by processor. In the case of maximum parallelization, the user runs with one energy group per process. Typically, in serial Sn codes, energy groups are swept sequentially from highest to lowest where the source term is updated after each energy group sweep so that effects from higher groups are immediately included in the lower group sweeps, thus providing Gauss-Seidel like convergence on the iteration. Therefore, in the case of no upscatter, the solution would converge after a single iteration of all groups. In the case of maximum parallelization, which is frequently the case in a typical AMTRAN calculation, all energy groups are being solved simultaneously, using source terms calculated from the previous iterate fluxes and, therefore, the iterative technique is Jacobi-like rather than Gauss-Seidel. One might expect a Jacobi solution to take more iterations to converge than Gauss-Seidel however, in practice, what we observed for problems dominated by fission, and thus have large upscatter components, there appears to be a break-even point at about 16 energy groups where, for problems with fewer than 16 energy groups, a Jacobi solution converges in fewer iterations and with more than 16 energy groups, a Gauss-Seidel solution converges in fewer iterations. The difference was not large, however, varying by about +/- 20% from 6 to 24 energy groups.

In AMTRAN, each process calculates it's energy group(s) contribution to the source term of each energy group in the problem. At this point, two different methods can be employed for communicating the results to the other processes. The first method is a tree-summing algorithm illustrated in fig. 5 for a 4 group calculation with one energy group per process.

Many vendor implementations of MPI_Allreduce implement essentially the same algorithm, however, we have seen MPI_Allreduce performance on some machines to be substantially worse than our implementation of the above algorithm and, therefore, we do not rely on the MPI_Allreduce call for the summing of the sources since it can be a

significant fraction of the run time of a calculation.

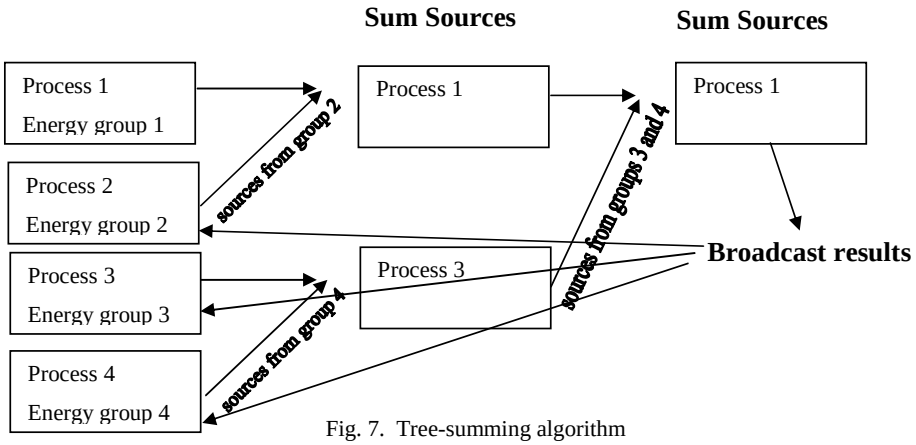

Fig. 7. Tree-summing algorithm

A second method takes advantage of the fact that a process only needs to know what the source contributions are to it's energy group(s). Thus, after a process computes the contribution of its energy group(s) to all the others, it sends individual messages to each process containing the contribution to that process' energy group(s). This is illustrated in fig. 8. The method illustrated in fig. 7 requires 3(N-1) communications while that illustrated in fig. 8 requires N(N-1) communications, where N is the number of MPI processes per domain. The messages in the second method, though, are much smaller and less synchronized than those in the first method and, as a result, provide about a factor of 2 reduction in wall clock time for a 16 energy group calculation on the IBM ASCI Pacific Blue SP-2 machine at LLNL.

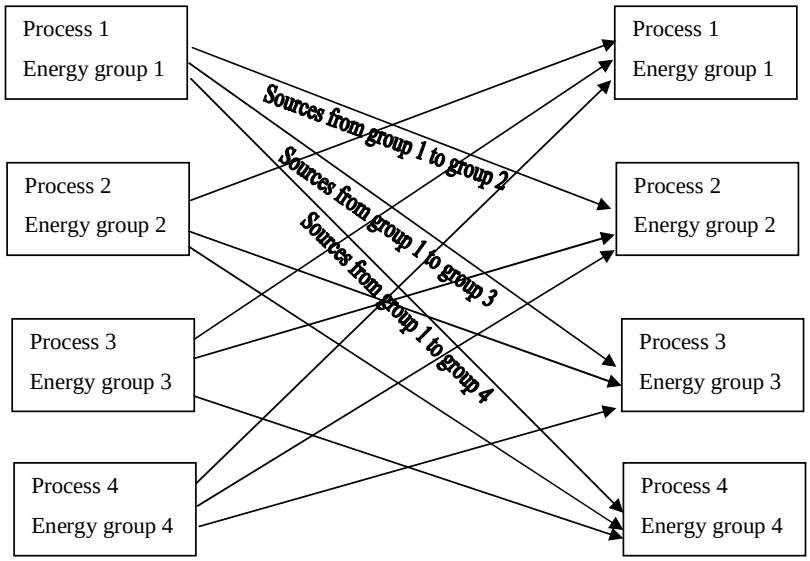

Fig. 8. Arrows for process 1 are labeled. Arrows for other processes would be labeled in an analogous fashion.

3 Numerical Results

As a simple numerical demonstration of the effectiveness of spatial AMR, the two-dimensional rod test case, as defined in ref. 4, will be used. This problem consists of a cylindrical rod of ^{235}U surrounded by vacuum with a density that decreases linearly from 66.71 g/cm^3 at the center plane to 20.09 g/cm^3 at the ends. All problems were run on LLNL's Thunder machine, which is a 1024 node (four processors per node), 1.4 GHz Itanium machine. Aussourd[4] specifies the finest level zoning to be 1 mm and gives results for up to 8 levels of AMR, but states that efficiency gains beyond 3 levels are negligible and tend to degrade accuracy. In fact, since the difference in density between the peak value and the ends is only a little more than a factor of three and the neutron mean free path varies linearly with the density, allowing more than three levels violates AMTRAN's default zoning criteria, since three levels of refinement already represents a factor of four difference in zone size for each direction. Fig. 9 shows the zoning used by AMTRAN with three levels of refinement. Table 1 shows the results of several serial variations of the calculations relative to a serial baseline calculation consisting of a single block, uniformly zoned with 1 mm zoning. The k_{eff} of the baseline calculation was 1.98480, which differs slightly from ref. 4. This isn't surprising since the nuclear database and energy group resolution were not specified, so a direct comparison could not be made. The relative error in Table 1 is defined as:

$$Error = abs\left(1 - \frac{1 - k_{eff}}{1 - k_{eff}^{baseline}}\right) \tag{6}$$

and the relative compute time is just the ratio of the time for the calculation relative to the baseline calculation:

$$Compute\ \ time = \left(\frac{time}{time_{baseline}}\right) \tag{7}$$

The major difference between Aussourd's[4] AMR method and our application is our block based approach versus his tree based hierarchy. As he points out, the advantage to a block based approach is it is more amenable to spatial parallelism, but is less efficient in reducing zone counts. A tree based algorithm is better able to capture irregularly shaped gradients. Our experience, however, has been that calculations generally run

Mesh	Relative Error	Relative compute time	Number of zones
Uniform, single block (baseline calculation)	0	1	20800
Uniform, multi-block	0	0.95	20800
3 level AMR	2.0e-5	0.20	5200
Ref. 4 with 3 level AMR	4.1e-5	0.30	5760

Table 1.

more efficiently with liberal settings for the boxing efficiency; i.e. it is better to minimize the number of blocks at the expense of running with more total zones. This assumes, of course, that the difference in zone count is not too large; generally no more than about 20%. If the difference is significantly more that 20%, it is probably worthwhile to increase the boxing efficiency.

Aussourd[4] reports roughly 40% overhead associated with the AMR logic for this test problem. As can be seen from table 1, we observe little, if any, overhead. In fact, the multi-block logic, which is the major cost associated with the AMR overhead of our block based approach, actually experiences a 5% reduction in run time for a uniform calculation relative to a single block uniform calculation. This is most likely do to improved cache performance of the multi-block approach since, if a block is small enough for all the unknowns to fit into cache, the sweeps can be performed without cache swapping. We have seen this effect in the past and, in fact, added an input variable which allows users control over the maximum size of a block so calculations can be tuned for different architectures. This super-linear speedup is also seen in the three level AMR calculation, which runs 5 times faster than the baseline calculation despite the fact that the zone count is only reduced by a factor of 4. It should be noted, though, that this particular test problem is ideally suited for a block based AMR approach, since the gradients are
planar.

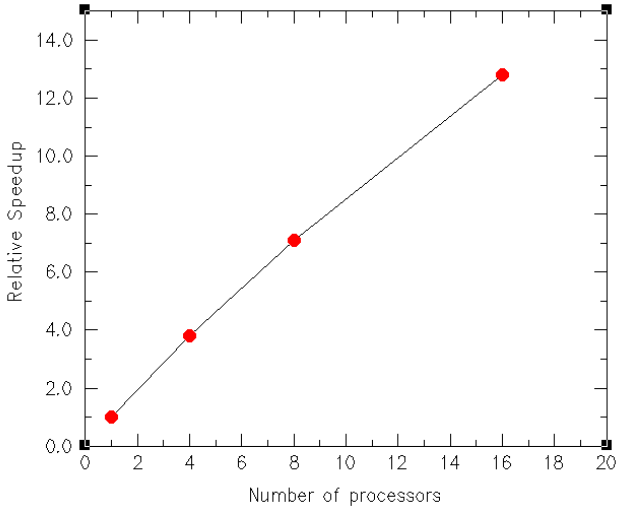

Fig. 9. Relative speedup as a function of the number of processors for the rod test problem (fixed problem size).

Fig. 9 shows the relative speed improvement for the rod test problem as a function of processor count. All points were run with the three level AMR version of the problem, giving a total of 5200 zones with 16 energy groups and S10 quadrature. The 4, 8 and 16 processor runs used 4 spatial domains. The 8 processor run has two processors assigned per domain, and, therefore, would be running with 8 energy groups per processor. The 16 processor run has 4 processors assigned per domain, thus giving 4 energy groups per processor. Since there are two reflecting planes in this problem (the axis and z=0), the

domain decomposed problems take more iterations (~20% increase) to converge than does the serial calculation, so the timings have been normalized to the iteration count of the serial calculation. As can be seen from the plot, we achieve about a 12.8X speedup with 16 processors, giving an overall parallel efficiency of 80%. The small problem size limits the degree of parallelism we can employ for this particular test case (the 16 processor run completed 116 iterations in about 7 seconds). Two dimensional calculations are commonly run that exceed 50,000 zones with 32 or more energy groups. A large three dimensional calculation can have several million zones. These kinds of calculations require hundreds to thousands of processors.

4 Future Work

Much of our recent effort has been focused on the ability to refine in direction. Problems such as the neutron interrogation of a cargo container, mentioned in sec. 1, not only require spatial AMR because of the large problem dimensions, but one is generally only interested in a narrow region of directional phase space; basically the cone of angles, originating from a 14 MeV source, that passes through the container and strikes a detector on the far side of the container. This is the classic source/detector problem of trying to get adequate resolution at a detector that is far from the source; ray effects can be severe. We have been working on techniques that allow local adaptive refinement of the directional set of angles and expect to publish our results in the near future. We also hope to extend the quadratic finite element in angle[7] work to multi-dimensions to see if the substantial improvements in convergence as a function of the number of angles holds for higher dimensions.

References

[1] Clouse, C.: "Parallel 3D Neutronics on an AMR Grid", Fifth Joint Russian American Conference on Computational Mathematics, Sept. 22, 1997.

[2] Clouse, C.: "Parallel Deterministic Neutron Transport with Adaptive Mesh Refinement", 16th International Conference on Transport Theory, Georgia Tech University, Atlanta, GA, May 14, 1999.

[3] Baker R.: A Block Adaptive Mesh Refinement Algorithm for the Neutral Particle Transport Equation. Nuc. Sci. & Eng., 141, 1-12 (2002)

[4] Aussourd, C.: Styx: A Multidimensional AMR S_N Scheme. Nuc. Sci. & Eng., 143, 281-290 (2003)

[5] Greenbaum, A. and J. Ferguson: A Petrov-Galerkin Finite Element Method for Solving the Neutron Transport Equation. Journal of Comp. Phys., 64, 97-111 (1986)

[6] Lewis, E. and W. Miller: Computational Methods of Neutron Transport, Wiley and Sons, New York, 137-140 (1984)

[7] Tolar, R. and J. Ferguson.: Quadratic Finite Element Method for 1D Deterministic Transport. Trans. Am. Nucl. Soc., 90 (2004)

[8] Berger, M. and P. Collela: Local Adaptive Mesh Refinement for Shock Hydrodynamics. Journal of Comp. Phys., 82, 64-84 (1989)

[9] Buck, R., E. Lent, T. Wilcox and S. Hadjimarkos: COG User's Manual, fifth edition. Lawrence Livermore National Laboratory. (2002)

[10] Compton, J. and C. Clouse: Domain Decomposition and Load Balancing in the AMTRAN Neutron Transport Code, 15th Annual Conference on Domain Decomposition Methods, Berlin, July (2003)