LOS ALAMOS NEUTRAL PARTICLE TRANSPORT CODES: NEW AND ENHANCED CAPABILITIES

R. E. Alcouffe, R. S. Baker, F. W. Brinkley, B. A. Clark, K. R. Koch, and D. R. Marr
Radiation Transport Group, X-6
Los Alamos National Laboratory

Abstract

We present new developments in Los Alamos discrete-ordinates transport codes and introduce THREEDANT, the latest in the series of Los Alamos discrete ordinates transport codes. THREEDANT solves the multigroup, neutral-particle transport equation in X-Y-Z and R-Θ-Z geometries. THREEDANT uses computationally efficient algorithms: Diffusion Synthetic Acceleration (DSA) is used to accelerate the convergence of transport iterations, the DSA solution is accelerated using the multigrid technique. THREEDANT runs on a wide range of computers, from scientific workstations to CRAY supercomputers. The algorithms are highly vectorized on CRAY computers. Recently, the THREEDANT transport algorithm was implemented on the massively parallel CM-2 computer, with performance that is comparable to a single-processor CRAY-YMP. We present the results of THREEDANT analysis of test problems.

Introduction to THREEDANT

THREEDANT is the latest transport code in the ONEDANT/TWODANT/TWOHEX series of codes from Los Alamos National Laboratory. THREEDANT is very similar to these codes in that it is portable, efficient, and utilizes the discrete-ordinates method. THREEDANT also relies on the CCCC reactor-physics interface files to facilitate communication with other reactor physics codes. THREEDANT can use all of the usual cross-section formats. The input is driven by the same input module used in ONEDANT and TWODANT. THREEDANT also follows in the ONEDANT and TWODANT tradition in that efficient numerical methods are used to solve the transport equation. This efficiency allows three dimensional calculations on scientific workstations and supercomputers as well. This efficiency is obtained with a combination of simple, accurate finite difference

approximations and effective acceleration techniques for iterative processes. THREEDANT has been applied to whole-core reactor calculations, shielding calculations and a host of test problems. THREEDANT is currently in friendly user status, with users at selected sites assisting in the final phases of the initial development.

THREEDANT Methods

The methods in the THREEDANT code are for the most part the same as those used in the ONEDANT and TWODANT [1] codes since the latter are tried and true for the designed applications. The philosophy in all these codes is to keep the solution algorithms as simple as possible for rapid calculation taking advantage of the vector processor capabilities for current 'super computers'. Thus in the spatial dimensions we use a rectangular mesh in the coordinate system chosen for the problem (X-Y-Z or R-Z-Θ for THREEDANT) that spans the entire spatial region. We use diamond differencing with set-to-zero negative flux fixup [1] in the spatial variables, discrete ordinates in the angular variables, and the multigroup approximation for the energy variable. This completely characterizes the discretization of the transport operator. In general, source iteration is used to solve within group scattering, upscattering and fission processes. This is a Neumann series type iteration where as shown explicitly below, the source is computed from an estimate for the angular flux. The angular flux is updated by inverting the transport operator upon the source which is then recalculated, and so on until convergence is obtained. Because of the properties of the transport operator for linear differencing methods, the convergence of this iteration is assured, however, the rate can be exceedingly slow for both the within-group scattering and the multigroup upscattering including fission. Thus we include an iteration convergence acceleration method: diffusion synthetic acceleration. Basically it involves obtaining a corrected diffusion equation with the solution that is the same as the scalar flux solution of the transport equation. For multigroup problems with either fission or upscatter, we use a multigroup form of the DSA equation so that both the inner and outer iterations will be accelerated.

In the following sections we review in a little more detail the algorithmic considerations in solving the three-dimensional transport equation along with the computational considerations for enhancing efficiency. These include taking advantage of vector processing and the relatively new considerations of parallel processing on massively parallel SIMD architectures. We emphasize that THREEDANT has a fairly extensive set of checks on the quality of the solution based upon the satisfying of the angularly integrated balance equation. The satisfying of this balance equation over any selected region of the spatial domain is a fundamental property of the method of discretization and so is an important diagnostic.

Transport Algorithms

In order to explain the solution process for the transport equation, we first write out the equation discretized as multigroup in energy and discrete ordinates in angle with isotropic scattering as:

$$\Omega_m \cdot \nabla \psi_g^{k+1/2}(r,\Omega_m) + \sigma_{tg}(r)\psi_g^{k+1/2}(r,\Omega_m) = \sigma_{g\to g}(r)\phi_g^{k+1/2} + \\ + \sum_{g'=1}^{g-1} \sigma_{g'\to g}(r)\phi_g^{k+1/2} + \sum_{g'=g+1}^{G} \sigma_{g'\to g}(r)\phi_{g'}^{k}(r) + \frac{\chi_g}{k_{eff}}\Phi^k(r) + \\ + Q_g(r,\Omega_m) \\ g = 1 \to G \\ m = 1 \to M \qquad (1)$$

where

$$\phi_g^k(r) = \sum_{m=1}^{M} w_m \psi_g^k(r,\Omega_m)$$

, and

$$\Phi^k(r) = \sum_{g=1}^{G} \nu\sigma_{fg}(r)\phi_g^k(r)$$

k	is an outer iteration index,
m	is the angular index,
g	is the energy group index,
w_m	is the angular weight,
ϕ_g	is the scalar flux,
$\psi_g(r,\Omega_m)$	is the angular flux for group g and angle m.

In Eq. 1, we show the outer iteration procedure by the index k where the half integer denotes the unknown to be solved for while the integral k indicates the known function from the previous iteration. The spatial discretization of this equation is by diamond differencing; i.e. assuming a linear variation in the angular flux in each of the transverse directions in the spatial coordinate system. The angular derivative terms for R-Z–Θ geometry are also diamond differenced along levels of constant ξ as is done in TWODANT. This procedure allows the discrete representation of the transport operator (the left hand side of Eq. 1) to be arranged as lower triangular. That is, the left hand side of Eq. 1 is inverted non-iteratively.

Eq. 1 is also a synopsis of the solution procedure for the transport equation including the inner and outer iterations. An inner iteration is accomplished by inverting the transport operator (the left hand side of Eq. 1) onto the source which is separated into the within group scatter (the first term on the left hand side) and the inscatter sources. The within group source is updated from the transport inversion and the process is repeated for group

g until the scalar flux is converged to the specified tolerance. The downscatter source (second term on the right hand side of Eq. 1) is then updated for the next group with the other terms held constant. Inner iterations are then performed for this group as before, and the process is repeated for all groups. The fission and upscatter terms are then updated; this constitutes one outer iteration. The outer iterations are repeated until the fission and upscatter terms are converged.

To assess the computational cost of this solution procedure, we measure the cost of inverting the transport operator and the total number of iterations involved. The latter consideration will be addressed in the next section when we treat iteration convergence acceleration. Here the cost of inverting the transport operator will be treated as a basic measure of efficiency. That is, since the inversion of the transport operator is non-iterative, we measure the efficiency of inversion by determining the average CPU time required for a phase space cell; the CPU time per angle per group per spatial mesh cell, which is called the **grind time** in this paper. The grind time is machine dependent and will also depend on how efficiently vectorization and paralellization have been accomplished.

The numerical inversion of the transport operator is recursive in the direction of flow of the particles. That is, one cannot compute the downstream portion until all communicating upstream cells have been computed. This seems to rule out vectorization. However, as shown in Fig. 1, the calculational procedure can be arranged so that in a 2D plane the flow proceeds along a logical diagonal in the spatial mesh. All cells along a diagonal can be solved simultaneously since the initial data has been computed from the adjacent upstream diagonal. This then allows vectorization and on a Cray YMP the speedup is about a factor of 5 for large problems (the grind time decreases from 2.1 μs to 0.4 μs). We incorporate this planewise solution method into THREEDANT.

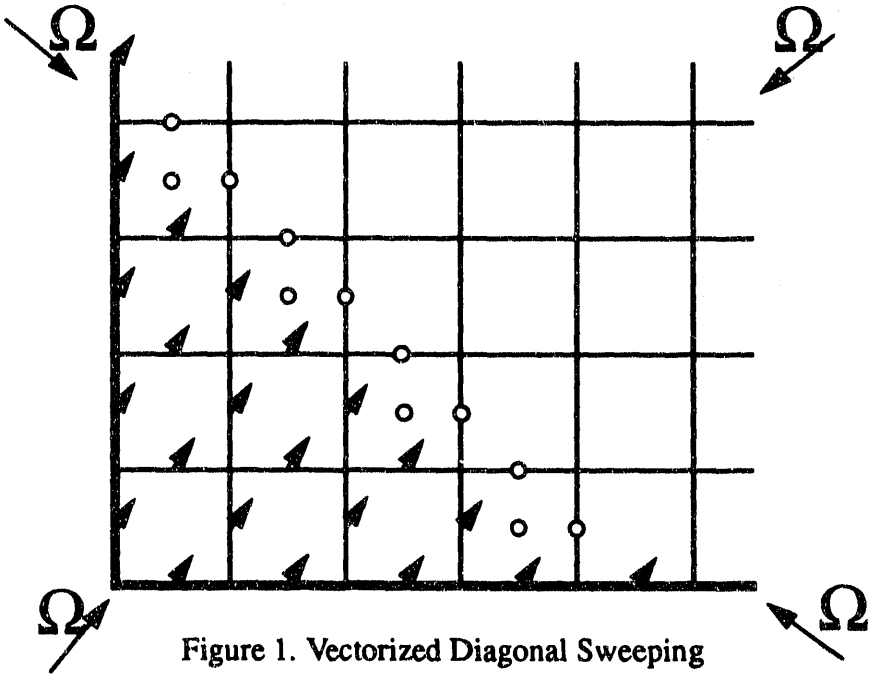

Figure 1. Vectorized Diagonal Sweeping

DSA Methods

As indicated above, the computational cost of solving the transport equation is directly proportional to the number of iterations needed to attain a specified convergence tolerance. The spectral radius of convergence of both the inner and outer source iterations can be arbitrarily close to unity depending upon the scattering ratio for the former and spectral or fission effects for the latter. This means that an arbitrarily large number of unaccelerated iterations would be necessary for solution. Thus it is imperative that an effective and efficient iteration convergence acceleration method be used in order to reduce the computational cost in the general case of solving the transport equation. For our codes we have implemented a diffusion synthetic acceleration (DSA) [2] method to help converge the solution procedure. In brief, we employ a corrected, multigroup diffusion equation converged whose solution is the same as the scalar flux solution of the transport equation. It has been shown many times in the literature that when the diffusion equation is properly discretized, the DSA procedure is effective in reducing the number of iterations to a predictable amount that is usually much less than is the case for the unaccelerated iteration procedure [2,3]. In the following we outline the procedure for the three-dimensional transport equation and point out the requirements for a successful DSA method in terms of computational efficiency. We also point out some of the limitations of the method as incorporated because of compromises that have been made and because of the limitations of the transport discretization method.

We begin by displaying the DSA equation analogous to Eq. 1:

$$-\nabla \cdot \underline{D}_g^{k+1/2} \cdot \nabla \phi_g^{k+1}(r) + \sigma_{Rg}(r)\,\phi_g^{k+1}(r) = \sum_{\substack{g'=1 \\ g' \neq g}}^{G} \sigma_{g' \to g} \phi_{g'}^{k+1}(r) + \frac{\chi_g}{k_{eff}} \Phi^{k+1}(r) + Q_{0g}(r) \tag{2}$$

where

$$[\underline{D}_g(r)]_{i,i}^{k+1/2} = -\frac{[J_g^{k+1/2}(r)]_i}{\nabla_i \phi_g^{k+1/2}(r)}, \quad i=1,2,3$$

$$J_g^{k+1/2}(r) = \sum_{m=1}^{M} w_m \Omega_m \Psi_g^{k+1/2}(r, \Omega_m), \tag{3}$$

$$\sigma_{Rg}(r) = \sigma_{tg}(r) - \sigma_{s0g}(r).$$

In Eq. 2 we display a particular form for the DSA method applicable to multigroup problems to be used in conjunction with Eq. 1 where the index k+1 indicates that the solution to Eq. 2 is to be used to update the source of Eq. 1. In this form the diffusion coefficient becomes a diagonal tensor whose elements are derived from the previous transport solution (at index k+1/2). Thus, the transport iteration procedure is to first estimate the scalar flux and compute the source for Eq. 1. As before, we then invert the transport oper-

ator in each group, starting in the highest energy group and updating the down-scattering source as we go along, to obtain the angular flux, $\psi_g^{k+1/2}$. This is then used to compute the diffusion coefficient tensor elements for Eq. 2 which is then solved to obtain the next estimate of the scalar flux, ϕ_g^{k+1}, which is in turn used to compute the source for Eq. 1, etc.

This addresses the outer iteration procedure but there still is the inner iteration to be considered. If there is no fission or upscatter, then all elements of the right hand side of Eq. 1 are known except the ingroup scattering source. The inner iteration procedure then makes use of the same equations (1 and 2) except that in this case the fission term is absent and all the within group scatter terms are known. Instead of doing a multigroup calculation, the within group scattering source is iterated group by group using the single group g equations 1 and 2 until the scalar flux is converged in group g to the specified precision.

Returning to the outer iteration, we make use of Eqs 1 and 2 as shown, but have the option of iterating the within group scattering source either assuming the inscatter and fission source are known as explained above, or to iterate this source partially. In the general case, we have found that for problems with fission and/or those with strong upscattering, the most efficient strategy is to take only one inner per group through Eq. 1 and use Eq. 2 to converge Φ and the upscatter source. The outers then consist in updating the diffusion tensor elements from the transport inversion and repeating the pass through Eq. 2. Once Φ and the upscatter are converged, then for each group the within group scattering is iterated until the scalar flux has converged as in the inner iteration procedure above. This usually does not change the fission distribution more than the convergence precision so that the solution will have been obtained. This is the 'one inner per outer until source convergence' procedure which is the default in all our transport codes using DSA.

We noted that the predicted effectiveness of the DSA method is based upon the diamond differenced form of the transport equation. A situation which interferes with the accelerator is when the diamond method is modified because of negative extrapolated fluxes. The negative flux fixup algorithm is non-linear in the sense that its use depends upon the solution being generated. In severe cases the use of fixup will cause the DSA procedure to be less effective and in fact worse than no acceleration. The code tries to detect such situations and simply turn off the accelerator. This is noted in the iteration synopsis of the code. To avoid this situation, one usually has to remesh the problem to yield less fixups. A fixup monitor to aid in this has been installed in TWODANT and is slated to be also included in THREEDANT in the near future.

Multigrid Method

We note that in the above described multigroup DSA iteration strategy in iterating the solution to the transport equation to convergence, the multigroup transport outers are replaced by the solution of the corrected transport equation, Eq. 3. This strategy will only be computationally efficient if the CPU time required to solve the diffusion equation is substantially less than that required to invert the transport operator over all the angles and groups. Since the diffusion operator is elliptic, in the multidimensional cases the inversion of the operator is done iteratively by some method such as SOR, Tchebychev, ICCG, or

multigrid. In Ref. 4 we show that for the three-dimensional, S-4 case, in order for the solution of the diffusion equation to be less costly than that of the transport operator, the spectral radius of the iteration procedure used to invert the diffusion operator must be less than 0.8. Of all the methods listed above, only the multigrid can guarantee a spectral radius less than unity and thus is the method chosen in THREEDANT to invert the diffusion operator.

Since the details are covered in Ref. 4, we only sketch the multigrid procedure used in the THREEDANT code. The multigrid method for inverting the diffusion operator in three dimensions consists in the following stages: An alternating line relaxation is performed on the original (finest) mesh in each of the three coordinate directions. A line relaxation is an inversion of the diffusion operator along a line of points in the mesh; and alternating implies that every alternate line is chosen so that the transverse leakage terms coming from the differenced form of the diffusion operator are held constant on each pass. A coarser mesh is chosen which is one half of the original mesh in each direction and the fine mesh diffusion operator and residual are transferred to that mesh. The problem on this mesh is another diffusion problem which yields a correction to the solution on the fine mesh. To invert the coarser mesh diffusion operator, an alternating line relaxation is then done in the same way as on the fine mesh. The results are then transferred to a still coarser mesh and the procedure is repeated until we come to a coarsest mesh. On this mesh the transferred diffusion operator is inverted directly as a matrix which yields the coarse mesh correction. This correction is then interpolated onto the next finest mesh and added to the previous results. Another relaxation pass using the interpolated and added result gives the smoothed correction which is in turn interpolated onto the next finer grid. This is continued until the original of finest grid is achieved ending the first multigrid cycle. If the prescribed convergence tolerance has not yet been achieved, then another cycle is gone through and repeated until convergence. Experience has shown that about two cycles are all that is necessary to reduce the residual error by more than an order of magnitude for the typical problem. The interpolation operator that has been chosen is linear-in-current and the operator that goes from the finer to the coarser grids is its inverse. The procedure used to get the coarser grid diffusion operators is based upon a simplified balance approach so that the original seven point operator is maintained on all grids. This is a compromise made for computational efficiency in the code.

Because of the compromises made in the implementation of the multigrid method for computational efficiency, the method does not always perform optimally as far as the number for cycles required for convergence. For certain very bad cases it may even fail; both these situations are noted in the synopsis of iteration performance when the code is run. In the most severe cases the DSA is simply turned off for that group and so the code will self recover to obtain a solution. But in general the performance of the method is very good as predicted and leads to a DSA method which performs admirably for typical reactor core calculations.

Massively Parallel Methods

The typical method for performing the inner iteration of the first-order form of the discrete ordinates equation involves a recursive sweep through all mesh cells. The simple way of performing such a sweep is to recursively solve down a row of cells, repeating this for each column in 2D, and then repeating this for each plane in 3D. A vectorized approach for 2D, as used in the TWODANT code, solves a diagonal line of cells in vector mode and then repeats this for all diagonal rows; the equations for a diagonal row of cells are independent and can be solved simultaneously. The THREEDANT code currently uses the vectorized "diagonal line" sweep which is repeated for each plane. To improve the speed of the 3D sweep and to provide enough work for a massively parallel processor (MPP) like Thinking Machines Corporation's CM2 and CM5, a new "diagonal plane" scheme was developed. The "diagonal plane" sweep is the 3D analog to the 2D "diagonal line" sweep. In 3D, the independence of the equations allows a diagonal plane of cells to be solved simultaneously, instead of just a diagonal line of cells.

The new "diagonal plane" sweep was implemented in a small test code on our CM2 for arbitrary X-Y-Z geometries. The bottom face of the 3D mesh is mapped onto a 2D set of CM2 processors, while the vertical columns associated with each mesh on the bottom face are treated as local arrays on each processor. This arrangement allows for the simultaneous solution of a diagonal plane of cells, which is then repeated for each diagonal plane needed to cover the 3D mesh. One drawback of the scheme is that the number of diagonal cells builds up as 1, 3, 6, 10, 15,... until it reaches a maximum, and then eventually scales back down to 1. Thus, processors are wasted at the beginning and at the end of the calculation. To reflect this, we define the Parallel Computational Efficiency (PCE) as follows:

$$\text{PCE} = \frac{N_{SA}}{N_D N_P}, \text{ where} \tag{4}$$

N_{SA} is the number of space-angle cells in the phase space, N_D is the number of diagonal planesin a given sweep, and N_P is the number of processors.

Two variations of the "diagonal plane" sweep have been tried. Method 1, "successive in angle, successive in quadrants", was designed to have a large PCE (80% to 90%) in order to maximize computational efficiency. Method 2, "simultaneous in angle, successive in quadrants", was devised to reduce the large amount of time spent in communications in Method 1. The PCE for Method 2 ranged from only 40% to 60%, but proved to be faster overall than Method 1 for smaller problems. The grind times (total CPU time per space-angle phase space element) and the PCE's for the two methods are shown in Table 1 for various mesh sizes and CM2 configurations.

Table 1. Grind Times (μsec) for 512/1024/2048 Floating Point Processors

Mesh Size	VP Ratio	PCE_1	Method 1	PCE_2	Method 2
64x32x32	4/2/1	80.3%	1.84/2.32/2.34	40.5%	.703/.866/.946
64x32x64	4/2/1	89.1%	1.62/2.05/2.06	57.7%	.482/.593/.592
64x64x64	8/4/2	85.9%	.857/.733/1.08	50.4%	.389/.275/.334
64x64x96	8/4/2	90.1%	.810/.689/1.02	60.4%	.323/.228/.278
128x64x96	16/8/4	85.8%	.440/.366/.356	50.3%	.325/.190/.132
128x96x128	24/12/6	87.4%	.318/.242/.358	53.6%	.270/.156/.132
128x128x96	32/16/8	81.9%	.317/.196/.193	43.0%	.319/.178/.109
128x128x128	32/16/8	85.8%	.302/.188/.183	50.2%	.276/.155/.094
128x128x192	32/16/8	90.1%	.288/.178/.174	60.2%	.232/.130/.079
192x128x192	48/24/12	87.9%	.277/.164/.120	54.7%	.244/.141/.078
192x192x128	72/36/18	80.1%	.285/.158/.153	40.1%	.311/.166/.136
256x192x128	96/48/24	77.5%	.283/.155/.089	36.5%	.329/.176/.097
256x256x128	128/64/32	75.1%	.286/.152/.086	33.4%	.355/.185/.101

THREEDANT Test Problems

We have selected two problems to present results on the performance of the THREEDANT code. These come from a benchmarking effort proposed by the Japanese [5] and have been computed by various methods and codes throughout the world.

Problems

The first problem is a three-dimensional representation of a small LWR whose schematic is shown in Ref. 5; the problem was run in S_4 quadrature and the spatial meshing was 25x25x25 and 50x50x50. The second problem is a mildly complicated mockup of an experimental fast reactor (FBR) core and the schematic is shown in Ref. 5. The problem was also run in S_4 quadrature and for two meshings: 32x32x20 and 64x64x40.

Iterative Performance

We present some of the calculational results in Table 2 in order to ascertain iteration performance of both the DSA used and the multigrid method used in conjunction with it. We first present a synopsis of the efficiency with which the diffusion operator is inverted,

Table 2. Calculational Results for Diffusion Accelerated S_N in Three Dimensions.

		LWR 25x25x25		LWR 50x50x50		FBR 32x32x20		FBR 64x64x40	
		SR[a]	MG[b]	SR	MG	SR	MG	SR	MG
Work Units[c]	grp 1	2601	233	8634	217	1068	441	2304	407
	grp 2	798	245	2556	185	1839	489	5505	284
	grp 3					1995	444	4683	334
	grp 4					1401	535	4584	412
Spectral Radius (ρ)	grp 1	0.98	0.58	0.99	0.69	0.87	0.48	0.97	0.67
	grp 2	0.93	0.55	0.98	0.72	0.93	0.51	0.99	0.60
	grp 3					0.91	0.50	0.98	0.58
	grp 4					0.88	0.53	0.99	0.66
CPU time	DSA	29.7	8.6	319.4	30.3	53.3	32.3	629.8	123.9
T_d µs		0.56	1.15	0.23	0.60	0.41	0.84	0.23	0.53
CPU time	S_N	4.4	4.4	20.0	21.4	20.2	20.2	100.1	93.8
T_{Sn} µs		0.73	0.73	0.44	0.45	0.76	0.76	0.40	0.40
K_{eff}		0.96235		0.96248		0.97060		0.97134	

a. Successive Over-Relaxation used to solve the DSA equations.

b. Multigrid Method used to solve the DSA equations

c. A **work unit** is the computational effort needed to do one line relaxation sweep over the three-dimensional mesh.

on the one hand using a simple successive relaxation approach (SR), and on the other hand using the multigrid approach. The first four rows give the total number of work units for each group used to converge the diffusion operator via each method for the four problem/ meshings. A **work unit** is the computational effort needed to do one line relaxation sweep over the three-dimensional mesh. As can be seen, the SR method requires a factor of 10 to 40 times as much work as the multigrid for the LWR problem and a factor of 3 to 15 for the FBR problem. The larger ratios are for the finer meshes. It is well known that the SR method has a higher spectral radius of convergence as the mesh is refined, and it is evident that the multigrid method does not suffer from this degradation. The next four rows give the average spectral radius of convergence for each method to verify this. The next row displays the CPU time spent solving the DSA equations while converging the transport solution. We see improvements from multigrid to SR by factors of from 3 to 10 for the LWR problem and 2 to 5 for the FBR problem. The next row gives the time per cell per work unit for the diffusion solver in microseconds. It is seen that these times for the SR are better than for multigrid because the latter has a great deal of overhead associated with it. In the next two rows we present the corresponding data for the transport solver. We note that for this S-4 case, the DSA and transport parts of the solution require about the same amount of CPU time to achieve the converged solution.

From these results, which are typical for reactor core calculations, we see that the iterative performance and calculational efficiency of the THREEDANT code is at about the level we anticipated. The grind times for the inversion of the transport operator is

around 0.4 μs which is that attained in TWODANT. The performance of the DSA solver is a little worse than in two dimensions but it is acceptable.

In Table 3 we present some preliminary results of calculations of the benchmark problems on three classes of machines: a vector super computer, a scientific work station, and a massively parallel SIMD machine. These are represented by the Cray YMP, a SUN 4/75, and the Thinking Machines, CM2 respectively. In section A of Table 3 we compare the total CPU time for three benchmark problems, the two described above plus another small FBR. The comparison is between the YMP and the SUN 4/75 showing the total CPU time and the grind time on each platform. The ratio of CPU times vary from a factor of 15 to 35 depending upon the amount of vector efficiency on the YMP which is a function of problem size. These results also show that reasonably sized three-dimensional problems can be run on a scientific workstation in an acceptable amount of time. The second section compares the performance of the transport inversion on a massively parallel SIMD machine, the CM2, and the YMP for the LWR 50x50x50 problem run in S_6 quadrature. These are preliminary results without DSA since it is not yet implemented on the CM version of the code. There was also a glitch in the CM version in that only about 47 outers could be run before a storage overflow problem stopped execution. Thus the first and second columns give the YMP times and grind time for the problem run with DSA, without DSA but to completion, and without DSA run to 47 outers. The third and fourth columns give the available CM times; this shows that the CM version performs at present a little better than the YMP for this problem, and from the results of the previous section, we expect it to perform much better on larger problems. This is a remarkable result showing that the first order form of the transport equation is successfully parallelized on a SIMD machine.

Table 3A. A Comparison of Execution Times for Benchmark Problem - A for the Cray YMP, SUN 4/75, and CM2 computers.

Problem Part A	Cray YMP		SUN 4/75	
	CPU time (s)	Grind time (μs)	CPU time (s)	Grind time (μs)
LWR	22.2	0.73	508.0	17.3
FBRS	23.9	1.16	382.7	17.0
FBRL	89.1	0.63	3001.7	18.3

Table 3B. A Comparison of Execution Times for Benchmark Problem - B for the Cray YMP, SUN 4/75, and the CM2 computers.

Problem Part B	Cray YMP		CM2	
	CPU time (s)	Grind time (μs)	CPU time (s)	Grind time (μs)
LWR (DSA)	76.7	0.47		
LWR (no DSA)	1703.9	0.47		
LWR (47 outers	1347.8	0.47	1485.4	0.43

OTHER DEVELOPMENTS

Given the diversity of this conference, we thought that it would be productive to describe other relevant activities that might be of interest. There are two other projects that may be of interest to the radiation transport community. First is the addition of a coupled S_N/Monte-Carlo hybrid transport algorithm that is being tested in TWODANT. The second project is a modification of the TWODANT code to allow a general quadrilateral mesh in place of the usual orthogonal mesh. This code is called TWODANT/GQ, for TWODANT / General Quadrilateral.

S_N/Monte Carlo Hybrid Method in TWODANT

In the fully coupled Monte Carlo/S_N response matrix method, spatial and/or energy regions of a problem are defined in which either a Monte Carlo or an S_N calculation is performed. The regions are then connected through the common boundary fluxes, for spatial interfaces, and group sources, for energy interfaces. The fully coupled Monte Carlo/S_N technique differs from previous coupling methods in that no assumptions are made about geometric separation or decoupling. Instead, the common boundary fluxes at a Monte Carlo/S_N spatial interface are determined through an iterative process which uses a response matrix for the Monte Carlo region, and standard S_N techniques in the S_N region. Thus, the fully coupled technique is ideally suited for problems involving both optically thick and optically thin regions which are tightly coupled, with the Monte Carlo technique being used in the optically thin region, and the S_N technique in the optically thick.

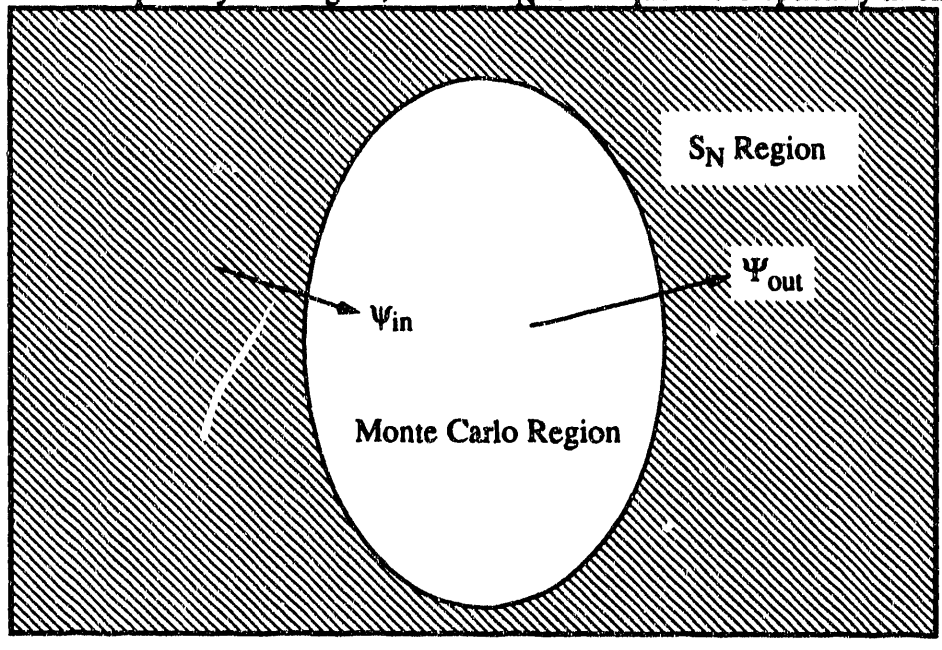

Figure 2. Boundary fluxes at an S_N/Monte Carlo interface.

Consider a Monte Carlo region embedded in an S_N region (Fig. 2), where Ψ_{in} and Ψ_{out} represent the boundary fluxes entering and leaving the Monte Carlo region and $\vec{n}$ is the outward-directed normal. The elements of Ψ_{out} are the incoming angular fluxes to the S_N region, where each element corresponds to a unique combination of spatial mesh cell,

energy group, and quadrature direction Ω^m, with $\Omega^m \cdot n > 0$. The interface angular fluxes for $\Omega^m \cdot n < 0$ are the elements of Ψ_{in}. The outgoing flux from the Monte Carlo region is related to the incoming flux from the S_N region by

$$\Psi_{out} = R\Psi_{in} + S_{out}. \tag{5}$$

where S_{out} is the exiting flux from the Monte Carlo region under vacuum boundary conditions ($\Psi_{in} = 0$). The element $r_{jj'}$, of the response matrix R represents the angular flux leaving the Monte Carlo region in S_N state j due to a unit incident angular flux in S_N state j'. Because Ψ_{in} is generally not known, Eq. (1) is solved iteratively by

$$\Psi_{out}^{(p+1)} = R\Psi_{in}^{(p)} + S_{out} \tag{6}$$

where, for example, we can set

$$\Psi_{out}^{(1)} = S_{out} \tag{7}$$

and $\Psi_{in}^{(p)}$ is obtained using an S_N solver with the prescribed boundary flux $\Psi_{out}^{(p)}$.

The fully coupled Monte Carlo/S_N method has been implemented in the S_N code TWODANT by adding special-purpose Monte Carlo subroutines to calculate the response matrices and group sources, and linkage subroutines to carry out the interface flux iterations. The common angular boundary fluxes are included in the S_N code as interior boundary sources, leaving the logic for the solution of the transport flux unchanged, while, with minor modifications, the diffusion synthetic accelerator remains effective in accelerating the S_N calculations. The Monte Carlo routines have been successfully vectorized, with approximately a factor of five increase in speed over the non-vectorized version.

The hybrid method is capable of solving forward, inhomogeneous source problems in X - Y and R - Z geometries. This capability now includes multigroup problems involving upscatter and fission in non-highly multiplying systems. The hybrid method has been applied to several challenging test problems with good results.

TWODANT/GQ

TWODANT/GQ is a modified version of the TWODANT code that has been generalized to solve problems using a general quadrilateral mesh. This allows for a much more accurate representation of curved surfaces. TWODANT/GQ is being developed for reactor physics applications as a part of the NPR Program. Input to TWODANT/GQ is eased by the Graphical User Interface that assists the user in problem setup and editing.

SUMMARY

We have presented some new and useful transport codes that are in the friendly-user stage of development. As that process unfolds, more users will be included, and results will be presented. Interested users should contact the Radiation Transport Group, X-6, at Los Alamos National Laboratory (505-667-4189) or the authors for more information.

REFERENCES

1. R. D. O'Dell, F. W. Brinkley, D. R. Marr, R. E. Alcouffe, "Revised User's Manual for ONEDANT: A Code Package for One-Dimensional Diffusion Accelerated Neutral Particle Transport," Los Alamos National Laboratory report, LA-9184-M Rev., 1989.

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

3. E. W. Larsen, "Unconditionally Stable Diffusion-Synthetic Acceleration Methods for the Slab Geometry Discrete-Ordinates Equations. Part I: Theory," *Nucl. Sci. Eng.*, **82**, 1982.

4. R. E. Alcouffe, "A Multigrid Solution of the Three-Dimensional DSA Equation: A Question of Efficiency for the Three-Dimensional Transport Iteration," Proceedings of Advances in Mathematics, Computation, and Reactor Physics, Pittsburgh, PA. (1991).

5. T. Takeda, H. Ikeda, "3-D Neutron Transport Benchmarks," OECD/NEA Committee on Reactor Physics report, NEACRP-L-330, (1991).