International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)
Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009)

A PARALLEL ALGORITHM FOR SOLVING THE INTEGRAL FORM OF THE DISCRETE ORDINATES EQUATIONS

R. Joseph Zerr[*]
Department of Mechanical and Nuclear Engineering
The Pennsylvania State University
138 Reber Building, University Park, PA, USA
rjz116@psu.edu

Yousry Y. Azmy
Department of Nuclear Engineering
North Carolina State University
1110 Burlington Laboratories, Raleigh, NC, USA
yyazmy@ncsu.edu

ABSTRACT

The integral form of the discrete ordinates equations involves a system of equations that has a large, dense coefficient matrix. The serial construction methodology is presented and properties that affect the execution times to construct and solve the system are evaluated. Two approaches for massively parallel implementation of the solution algorithm are proposed and the current results of one of these are presented. The system of equations may be solved using two parallel solvers—block Jacobi and conjugate gradient. Results indicate that both methods can reduce overall wall-clock time for execution. The conjugate gradient solver exhibits better performance to compete with the traditional source iteration technique in terms of execution time and scalability. The parallel conjugate gradient method is synchronous, hence it does not increase the number of iterations for convergence compared to serial execution, and the efficiency of the algorithm demonstrates an apparent asymptotic decline.

Key Words: neutron transport, discrete ordinates, parallel, block Jacobi, conjugate gradient

1. INTRODUCTION

Parallel algorithms for solving the transport equation have been widely discussed and implemented for approximately 25 years. The discretized transport problem was decomposed in one or more of the independent phase-space variables, data was distributed among the participating processors, and the transport equation was solved concurrently. Summarily, extensive literature is available for energy [1], angular [2], and spatial [3–6] domain decompositions, and further research demonstrated the utility of a combination of domain decompositions [7–9].

The value of modern parallel algorithms is typically measured by speedup and scalability [10]. Speedup refers to the ratio of the serial execution time to the parallel execution time on multiple

[*] Currently located at North Carolina State University, 3105 Burlington Laboratories, Raleigh, NC, USA

processing units. Scalability is a measure of how many processing units can be conceivably used to solve a decomposed problem while maintaining reasonable parallel efficiency. Further, algorithms exhibiting scalability are desired to do so with a speedup that warrants the additional costs associated with deploying computing power.

Energy domain decomposition and angular domain decomposition both offer straightforward means to solve the neutron transport equation in parallel. However, each of these methods is limited in scalability. Energy domain decomposition is limited to the number of energy groups and is asynchronous, and angular domain decomposition is limited to the number of ordinates in the angular quadrature set and is synchronous only in non-curvilinear coordinate systems. [1–2]

Spatial domain decomposition algorithms were sought to achieve greater scalability. Not surprisingly, the number of cells in a transport problem often significantly exceeds the numbers of groups or discrete ordinates. Various schemes have been devised to distribute cells of a larger spatial region across several computing nodes and couple them via interface angular fluxes.

Early schemes divided the region into sub-domains of varying shapes for Cartesian [3] and cylindrical [7] geometries. The transport equation is solved over all cells in the sub-domains, and all sub-domains are solved in parallel. The angular fluxes at the sub-domain interfaces are shared with neighbors to be used as incoming boundary conditions for the next iteration. The process is repeated until convergence.

In the 1990s, considerable effort was focused on the diagonal plane sweep or wavefront method [4–6]. Similar to a mesh sweep the outward angular fluxes for the starting cell are passed to the three adjacent cells in three-dimensional geometry. In the next three cells the cell-centered and outgoing angular fluxes are computed independently of, and in parallel with each other. Outgoing angular fluxes are passed to their neighbors. Directions are pipelined for additional computational efficiency. The wavefront method requires no alteration (except communication) from a serial mesh sweep, making it an intrinsic domain decomposition [5, 11].

Analysis of this scheme uses a two-dimensional computer cluster topology and has demonstrated considerable speedup on thousands of processors [4–6] because communication time was a small fraction of total execution time; the cumulative grind time was the dominant factor in such calculations. More recently, Humbert [12] applied the wavefront method to a three-dimensional computer topology. Humbert concluded that very good speedup could be attained for hundreds of processors. However, the method is not scalable as increasing the number of processors increases the amount of time spent in communication.

Other notable research involves the use of Krylov subspace methods, namely the conjugate gradient (CG) method, to replace iterative procedures of neutronics equations. Böhm, Brehm, and Finnemann [13] used a parallel CG solver for diffusion equations. Gupta and Modak [14] demonstrated how a discrete ordinates problem with the diamond difference scheme could be posed as a CG problem, replacing the source iteration (SI) scheme. Chen and Sheu [15] investigated the use of preconditioners with the CG method for neutron transport problems.

The motivation for this work is to develop a new algorithm that solves the transport equation in a massively parallel environment. Since grind time is still the dominant component of transport solvers [16], this work proposes an alternate methodology that abandons the mesh sweep and SI schemes. We discuss new strategies for the construction of the discrete form of the transport problem and the determination of the *scalar* flux solution from the resulting system of linear equations via parallel algorithms. We believe our algorithms, by eliminating the costs of highly repetitive mesh sweeps, can reduce overall execution time for very large problem sizes in massively parallel computing environments.

This paper is divided into the following sections. Section 2 briefly introduces the discretized transport equations used in our analysis. Section 3 describes some of the properties of our system of equations and how varying parameters affect problem size. Section 4 discusses two distinct paths that one can take to solve for the scalar flux distribution in parallel. Section 5 presents the results attained thus far for composing and solving the system of equations in parallel. Section 6 discusses the conclusions drawn from our work and outlines the future work to be performed.

2. THE AHOT-N METHODOLOGY

For this study we adopt the arbitrarily high order transport nodal (AHOT-N) equations [17, 18] as the starting point for our new algorithm. These equations can be easily implemented in a mesh sweep of an SI scheme using a weighted diamond difference (WDD) structure. Presented here are key equations for three-dimensional geometry as an introduction to the methodology and to introduce the notation. Readers are directed to the references for more detailed derivations.

2.1. The AHOT-N Equations

In AHOT-N the spatial distribution of the flux over a computational cell is computed as a truncated series of normalized Legendre polynomials. The energy group index has been suppressed.

$$\psi_{n,i,j,k,t,u,v} \equiv \int_{-a_i}^{+a_i} \frac{dx}{2a_i} P_t(x) \int_{-b_j}^{+b_j} \frac{dy}{2b_j} P_u(y) \int_{-c_k}^{+c_k} \frac{dz}{2c_k} P_v(z) \psi_n(x,y,z) \tag{1}$$

$$\psi_{n,i,j,k,x,u,v}(\pm a_i) \equiv \int_{-b_j}^{+b_j} \frac{dy}{2b_j} P_u(y) \int_{-c_k}^{+c_k} \frac{dz}{2c_k} P_v(z) \psi_n(\pm a_i,y,z) \tag{2}$$

The moments for the *y*- and *z*-directional edges, $\psi_{n,i,j,k,t,y,v}(\pm b_j)$ and $\psi_{n,i,j,k,t,u,z}(\pm c_k)$, respectively, can be defined analogously to Eq. (2). The discrete ordinate, *x*-, *y*-, and *z*-dimension indices are denoted with *n*, *i*, *j*, and *k*, respectively. $2a_i$, $2b_j$, and $2c_k$ are the *x*-, *y*-, and *z*-dimensions of the cell, respectively. The continuous angular flux distribution in the direction of *n* is $\psi_n(x,y,z)$. The functions $P_t(x)$, $P_u(y)$, $P_v(z)$ are the t^{th}, u^{th}, and v^{th} orders, respectively, of the Legendre polynomial, normalized over the range of the corresponding independent variables.

In AHOT-N the spatial expansion order of all fluxes in all dimensions ranges from 0 to Λ. Four sets of equations are needed to describe the flux for the increasing spatial order. The first set of $(\Lambda+1)^3$ equations expresses the conservation of angular flux spatial moments over the cell.

$$\begin{aligned}
&sg(\mu_n)^t \varepsilon^x_{n,i}\left\{\psi^o_{n,i,j,k,x,u,v} - (-1)^t \psi^i_{n,i,j,k,x,u,v}\right\} + sg(\eta_n)^u \varepsilon^y_{n,j}\left\{\psi^o_{n,i,j,k,t,y,v} - (-1)^u \psi^i_{n,i,j,k,t,y,v}\right\} \\
&+ sg(\xi_n)^v \varepsilon^z_{n,k}\left\{\psi^o_{n,i,j,k,t,u,z} - (-1)^v \psi^i_{n,i,j,k,t,u,z}\right\} - 2sg(\mu_n)\varepsilon^x_{n,i} \sum_{l=o(t)}^{t-1\,*} (2l+1)\psi_{n,i,j,k,l,u,v} \\
&- 2sg(\eta_n)\varepsilon^y_{n,j} \sum_{l=o(u)}^{u-1\,*} (2l+1)\psi_{n,i,j,k,t,l,v} - 2sg(\xi_n)\varepsilon^z_{n,k} \sum_{l=o(v)}^{v-1\,*} (2l+1)\psi_{n,i,j,k,t,u,l} \\
&+ \sigma^t_{i,j,k}\psi_{n,i,j,k,t,u,v} = \sigma^s_{i,j,k}\phi_{i,j,k,t,u,v} + S_{i,j,k,t,u,v}
\end{aligned} \tag{3}$$

sg is the signum function and acts on the *x*-, *y*-, and *z*-components of the angular direction cosines. The *i* and *o* superscripts refer to the incoming and outgoing angular flux edge moments, respectively. All the summations have an increment of two, denoted as Σ^*. The summations' starting indices are defined with $o(u)=[(u+1) mod (2)]$ (analogously for other directions). The macroscopic cross sections are denoted by $\sigma^t_{i,j,k}$ for total and as $\sigma^s_{i,j,k}$ for isotropic scattering. The scalar flux and fixed source are $\phi_{i,j,k,t,u,v}$ and $S_{i,j,k,t,u,v}$, respectively. The epsilon terms are defined analogously, and for the *x*-direction specifically, it is

$$\varepsilon^x_{n,i} \equiv \left|\mu_n\right| \Big/ 2a_i \,. \tag{4}$$

The next three sets, each comprising $(\Lambda+1)^2$ equations, are weighted difference formulas to relate angular flux spatial moments within the cell to the transverse-moments of the angular flux on cell *x*-, *y*-, and *z*-faces. The *x*-face equation is

$$\begin{aligned}
&\left(\frac{1+\alpha_{n,i,j,k}}{2}\right)\psi^o_{n,i,j,k,x,u,v} + \left(\frac{1-\alpha_{n,i,j,k}}{2}\right)\psi^i_{n,i,j,k,x,u,v} = \sum_{l=0,even}^{\Lambda\,*} (2l+1)\psi_{n,i,j,k,l,u,v} \\
&+ sg(\mu_n)\alpha_{n,i,j,k} \sum_{l=1,odd}^{\Lambda\,*} (2l+1)\psi_{n,i,j,k,l,u,v}
\end{aligned} \,. \tag{5}$$

The spatial weights, e.g. $\alpha_{n,i,j,k}$ in Eq. (5), have been defined by Azmy [17] as a series of terms, depending on the spatial order, of the ratio between ε_n from Eq. (4) and σ^t.

The calculation for each computational cell can be composed of a system of equations from Eqs. (3) and (5). One coefficient matrix operates on a vector for the angular flux moments within the cell and outgoing at the faces: the unknown quantities. Another coefficient matrix operates on the known neutron sources—scattering and distributed—and the incoming angular flux cell-face moments. A new matrix Γ is formed by inverting the unknown vector's matrix and multiplying it with the known vector's matrix. Then the problem for each cell is simplified to the form $x=\Gamma b$. This system has $(\Lambda+1)^3+3(\Lambda+1)^2$ equations.

$$\left[\psi,\psi^{o,z},\psi^{o,y},\psi^{o,x}\right]^T_{n,i,j,k}=\Gamma_{n,i,j,k}\left[\sigma^s\phi^p+S,\psi^{i,z},\psi^{i,y},\psi^{i,x}\right]^T_{n,i,j,k},$$

$$\Gamma_{n,i,j,k}\equiv\begin{bmatrix}\gamma^{aa} & \gamma^{axy} & \gamma^{axz} & \gamma^{ayz}\\ \gamma^{xya} & \gamma^{xyxy} & \gamma^{xyxz} & \gamma^{xyyz}\\ \gamma^{xza} & \gamma^{xzxy} & \gamma^{xzxz} & \gamma^{xzyz}\\ \gamma^{yza} & \gamma^{yzxy} & \gamma^{yzxz} & \gamma^{yzyz}\end{bmatrix}_{n,i,j,k} \tag{6}$$

The moment indices are not necessary for the sub-vectors which are composed of all moments. The *x*, *y*, and *z* superscripts in the equation indicate the dimension of interest in the equation; *x*, *y*, and *z* superscripts in the sub-matrices of Γ refer to the plane of integration from Eq. (2). The scalar flux is given the superscript *p* to denote it as an iterate in the SI scheme to determine a new scalar flux solution from its *p*revious value.

2.2. Using the Iteration Jacobian Matrix to Solve for the Scalar Flux

In the SI scheme one solves for the left hand side (LHS) angular flux moments of Eq. (6) for all angles and uses the angular quadrature to determine a new scalar flux.

$$\phi^v=A\left(\sigma^s\phi^p+S\right) \tag{7}$$

ϕ^v is the new scalar flux iterate and *S* is the source, both vectors. σ^s is the self-scattering cross section matrix, and *A* is a coefficient matrix whose elements are constructed from the sub-matrices of the Γ-matrix in the discretized transport equation, Eq. (6). By assuming vacuum boundary conditions for the domain, anisotropic sources that would otherwise appear on the RHS are neglected. Upon iterative convergence of Eq. (7), successive iterates of the scalar flux are equal in the iterative limit, i.e. the solution ϕ^∞ satisfies the following relation [10],

$$\phi^\infty=\left(I-A\sigma^s\right)^{-1}AS \tag{8}$$

where *I* is the identity matrix. In the SI scheme *A* is never constructed, and the solution is computed from successive mesh sweeps instead of solving the system of equations in (8).

The impetus to our new approach begins with construction of *A*, followed by explicitly solving the system in Eq. (8). First it is recognized that this matrix is the iteration Jacobian of Eq. (7).

$$A_{(i,j,k,t,u,v)(i',j',k',t',u',v')}=\frac{1}{\sigma^s_{i',j',k'}}\frac{\partial\phi^v_{i,j,k,t,u,v}}{\partial\phi^p_{i',j',k',t',u',v'}} \tag{9}$$

One must perform a single mesh sweep along all discrete ordinates in the angular quadrature to construct *A*. Instead of computing the cell-centered and outward angular flux moments given the incoming fluxes, the cell-moments of the angular flux of one node are coupled to the cell-moments of the scalar flux in all upstream cells for a specific discrete ordinate. Ultimately the

scalar flux spatial moments in any given cell will be related to the scalar flux spatial moments in all other cells. By differentiating the AHOT-N/WDD system of equations (6) with respect to ϕ^p, one can demonstrate the aforementioned coupling. The flux moment indices are suppressed and each equation represents the full range of flux spatial moments.

$$\frac{\partial \psi_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} = \gamma^{aa}_{n,i,j,k}\sigma^s_{i,j,k}\frac{\partial \phi^p_{i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{axy}_{n,i,j,k}\frac{\partial \psi^{i,z}_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{axz}_{n,i,j,k}\frac{\partial \psi^{i,y}_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{ayz}_{n,i,j,k}\frac{\partial \psi^{i,x}_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} \tag{10a}$$

$$\frac{\partial \psi^{o,z}_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} = \gamma^{xya}_{n,i,j,k}\sigma^s_{i,j,k}\frac{\partial \phi^p_{i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{xyxy}_{n,i,j,k}\frac{\partial \psi^{i,z}_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{xyxz}_{n,i,j,k}\frac{\partial \psi^{i,y}_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{xyyz}_{n,i,j,k}\frac{\partial \psi^{i,x}_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} \tag{10b}$$

The *y*- and *x*-direction equations are written analogously to Eq. (10b). The first term in each expression equals zero unless $i, j, k = i', j', k'$ because no formulaic relation exists among the previous iterate of scalar flux moments of all the cells. On the other hand, the incoming angular flux moments at the faces of a given cell are equal to the outgoing angular flux moments from the three adjacent upstream cells. Thus the three incoming angular flux moment terms in each equation must be evaluated for the cell's upstream neighbors.

We define the final three equations as matrices whose elements are accumulated during the mesh sweep. At each cell in the sweep, we compute the coupling for the three adjacent downstream cells to all the cells currently considered in the sweep.

$$Z_{(i,j,k+1)(i',j',k')} \equiv \frac{\partial \psi^{o,z}_{i,j,k}}{\partial \phi^p_{i',j',k'}}, \quad Y_{(i,j+1,k)(i',j',k')} \equiv \frac{\partial \psi^{o,y}_{i,j,k}}{\partial \phi^p_{i',j',k'}}, \quad X_{(i+1,j,k)(i',j',k')} \equiv \frac{\partial \psi^{o,x}_{i,j,k}}{\partial \phi^p_{i',j',k'}} \tag{11}$$

These matrices are composed of blocks at each upstream cell i', j', k'; each block's dimensions are $(\Lambda+1)^2\times(\Lambda+1)^3$. The angle index is suppressed because the mesh sweep in one angle is independent from all others. However, the angle manifests itself in direction cosines used to compute the γ sub-matrices as well as the progression order in the sweep.

From Eqs. (10) and (11) one can observe the relation between adjacent cells.

$$Z_{(i,j,k+1)(i,j,k)} = \sigma^s_{i,j,k}\gamma^{xya}_{i,j,k}, \quad Y_{(i,j+1,k)(i,j,k)} = \sigma^s_{i,j,k}\gamma^{xza}_{i,j,k}, \quad X_{(i+1,j,k)(i,j,k)} = \sigma^s_{i,j,k}\gamma^{yza}_{i,j,k} \tag{12}$$

The derivatives involving the incoming angular flux moments in Eq. (10) may be substituted with the adjacent cells' outgoing expressions as in Eq. (11). Repeating this for all upstream cells ultimately yields the following formulae to recursively update the *X*, *Y*, and *Z* matrices from all the non-adjacent upstream cells.

$$Z_{(i,j,k+1)(i',j',k')} = \gamma^{xyxy}_{i,j,k}Z_{(i,j,k)(i',j',k')} + \gamma^{xyxz}_{i,j,k}Y_{(i,j,k)(i',j',k')} + \gamma^{xyyz}_{i,j,k}X_{(i,j,k)(i',j',k')} \tag{13a}$$

$$Y_{(i,j+1,k)(i',j',k')} = \gamma^{xzxy}_{i,j,k}Z_{(i,j,k)(i',j',k')} + \gamma^{xzxz}_{i,j,k}Y_{(i,j,k)(i',j',k')} + \gamma^{xzyz}_{i,j,k}X_{(i,j,k)(i',j',k')} \tag{13b}$$

$$X_{(i+1,j,k)(i',j',k')} = \gamma^{yzxy}_{i,j,k}Z_{(i,j,k)(i',j',k')} + \gamma^{yzxz}_{i,j,k}Y_{(i,j,k)(i',j',k')} + \gamma^{yzyz}_{i,j,k}X_{(i,j,k)(i',j',k')} \tag{13c}$$

At each cell the values for *X*, *Y*, and *Z* from all upstream cells have been computed for that cell. The *A* matrix is updated using these values. The final three terms in Eq. (10a) are computed.

$$\frac{\partial \psi_{i,j,k}}{\partial \phi^{p}_{i',j',k'}} = \gamma^{axy}_{i,j,k} Z_{(i,j,k)(i',j',k')} + \gamma^{axz}_{i,j,k} Y_{(i,j,k)(i',j',k')} + \gamma^{ayz}_{i,j,k} X_{(i,j,k)(i',j',k')} \tag{14}$$

Further, one computes the first term in Eq. (10a).

$$\frac{\partial \psi_{i,j,k}}{\partial \phi^{p}_{i,j,k}} = \gamma^{aa}_{i,j,k} \sigma^{s}_{i,j,k} \tag{15}$$

The *A* matrix is updated with Eqs. (14) and (15) lastly by summing the contributions from all $N(N+2)$ angles in the quadrature set.

$$A_{(i,j,k)(i',j',k')} = \frac{1}{\sigma^{s}_{i',j',k'}} \sum_{n=1}^{N(N+2)} \omega_n \frac{\partial \psi_{n,i,j,k}}{\partial \phi^{p}_{i',j',k'}} \tag{16}$$

The *A* blocks updated with Eq. (16) have dimensions $(\Lambda+1)^3 \times (\Lambda+1)^3$. The locations of the blocks are determined by the indices on *A*. Diagonal blocks of *A* result from the expression given by Eq. (15), whereas the off-diagonal blocks result from expression given by Eq. (14); for downstream cells the LHS equals zero in Eq. (14) and thus has no effect in Eq. (16).

The *A* matrix is made symmetric easily. Each block of *A* described by Eq. (16) relates the flux in one cell caused by the flux in another. Because we assume the scattering and fixed sources are isotropic, the reciprocity relation holds: the flux at some point r_1 caused by an isotropic source at point r_2 is equal to the flux at r_2 caused by the isotropic source at r_1. To make *A* symmetric all elements in the set of $(\Lambda+1)^3$ equations of each spatial cell are multiplied by three factors for reciprocity: scattering cross section, cell volume, and a flux moment factor.

Symmetric matrices have advantages over their non-symmetric counterparts. If solving the system in Eq. (8) directly, Cholesky factorization may be applied, improving performance. Furthermore, when symmetric systems are also positive definite, the conjugate gradient (CG) method may be employed. Other Krylov subspace methods apply for coefficient matrices that are not symmetric positive definite (SPD). However, CG is the best choice when the system is SPD.

No analytic method has been determined yet to prove the positive definiteness for a general case of any problem size and material properties. However, experience for many test cases and a range of parameters has shown no convergence difficulties, and the computed solution has consistently agreed within the convergence criterion with the solution determined directly with Gaussian elimination.

3. PROPERTIES AFFECTING PROBLEM SIZE AND SOLUTION

We have implemented the three-dimensional AHOT-N/WDD solution methodology in a serial computer code to test accuracy and gather information about how the methodology is affected by various problem parameters. The AHOT-N/WDD equations may be solved using the SI approach or using the *A* iteration Jacobian matrix and Eq. (8). In the latter case, once the matrix is constructed and made symmetric it can be solved directly with the LAPACK subroutine DPOSV [19] or with CG iterations. Several expected relations between serial execution time and variable problem parameters have been confirmed by our research and are briefly highlighted here.

For brevity we define the number of cells as *M*. Refining the spatial mesh for a sample problem increases *M* while keeping fixed the overall physical dimensions of the problem. The number of SI and CG iterations is insensitive to *M*. SI execution times grow linearly with *M*, but CG time per iteration grows like M^2 because of the larger matrix and inner products. Additional cells require an additional row and column in *A*, and matrix construction and direct solution times thus grow like M^2 and M^3, respectively.

Raising the angular quadrature order for the problem also has straightforward consequences. The number of source iterations is insensitive to increasing the total number of discrete ordinates (*N* for brevity), but the SI execution time per iteration grows linearly with *N*. Constructing *A* also grows linearly with *N*. However, CG iteration time and direct solution time are unaffected since the matrix size does not change.

The scattering ratio of a material is adjusted by changing the scattering cross section while keeping the total cross section fixed. This will not affect the size of *A*, the direct solution time, SI time, or CG iteration time. The SI scheme displays a markedly slow convergence rate in highly scattering media. The CG method also requires a greater number of iterations when the scattering ratio of the materials increases and approaches unity, but the effect is considerably smaller.

4. COMPOSING AND SOLVING THE SYSTEM OF EQUATIONS IN PARALLEL

A is large and dense; it is a square matrix with dimension $M\times(\Lambda+1)^3$. Both storing a large matrix and solving its system serially are computationally prohibitive in large applications. Consequently, this technique generally is not competitive for serial implementation. However, we propose that the solution of the scalar flux moments can be solved on a parallel system with two different approaches designed to reduce per processor memory requirement and execution time.

Ultimately, our new approaches seek to be competitive with SI by avoiding the repetitive mesh sweeps and the costly grind time associated with them. The *A* matrix described in Sec. 2 is independent of the scattering source. Neglecting any change in cross sections that would accompany decay chain calculations, *A* needs to be constructed only once at the beginning of a calculation and multiplied with the in-group source. The scalar flux moments are then computed

by solving the algebraic system comprising Eq. (8). Two approaches to parallelizing this general methodology emerge.

4.1. Spatially Decomposing the Full Problem into Sub-Domains

In the first approach the overall region is divided into several sub-domains. This method is similar to early spatial domain decompositions described in Sec. 1. Each sub-domain is treated as an independent problem. We compute the scalar flux moments for all cells in the sub-domain. Using this information, we then compute all the outgoing angular flux moments at the boundaries of the sub-domain. The outgoing angular flux moments are passed between adjacent sub-domains in an iterative fashion. The exchanged data package is a set of boundary conditions for a new calculation to compute the updated scalar flux distribution within the sub-domain. An iterative process takes place until convergence of the cell-average, i.e. zeroth moment, of the scalar flux is achieved. Spectral properties of the iterative process have yet to be investigated for this scheme.

Two operators are necessary for this approach. They can be pre-computed only once and concurrently among all processes. To compute the scalar flux distribution for the sub-domain, we must account for the dependence of the fixed (isotropic) source and the anisotropic boundary conditions: $\phi = \mathbf{\Phi}(S,\psi_{in}) = \Phi_1 S + \Phi_2 \psi_{in}$. In the case of vacuum boundary conditions, the second term is dropped and the first term is equivalent to the RHS of Eq. (8). The second operator computes the outgoing angular flux moments at the edges of the boundary cells from the computed scalar flux spatial moments, source, and boundary conditions: $\psi_{out} = \mathbf{\Psi}(\phi,S,\psi_{in})$. These operators still involve large matrices whose order grows quickly with the number of spatial cells, $O(M^2)$, but partitioning the full region into sub-domains allows one to greatly reduce per processor memory and/or allows for a much larger problem size.

4.2. Parallel Construction and Storage of *A* with Parallel Solvers

In the second approach, the *A* matrix is composed and solved in parallel directly. Angular and spatial domain decompositions already demonstrate techniques capable of parallelizing the mesh sweep. Angular decomposition can only scale to hundreds of processors for most realistic applications. Spatial decomposition scales higher, but it does so at the cost of increased communication and processor idleness that could inhibit the parallel efficiency when applied to only a single sweep over all angles. Moreover, the unique coupling structure of the matrix may lend itself to a more suitable, massively parallel algorithm to construct *A*. Solving the system of equations in parallel can be accomplished in several ways, including the Block Jacobi (BJ) and parallel CG [13, 20] methods. Research is still ongoing to develop a massively parallel algorithm for the construction of *A*.

4.2.1. The parallel Block Jacobi method

In BJ each process is assigned a block of the matrix whose order is determined by the total number of equations divided by the selected number of blocks, assuming the quotient is an integer value. Moreover, the processes are assigned the corresponding block-size portions of the solution and right hand side (RHS) vectors. Each process inverts its owned diagonal block and

stores the result. In parallel each process multiplies an off-diagonal block by its owned solution sub-vector and passes it to the next process. Repeating this step and combining the sub-vectors $P-1$ times, where P is the number of processes, leaves each process with the sub-vector needed to complete its own RHS update. Once the RHS sub-vector is multiplied by the saved inverted diagonal block, each process will have computed its portion of the new scalar flux iterate. Clearly BJ is an asynchronous decomposition.

4.2.2. The parallel conjugate gradient method

The parallel CG solution follows the same sequence of steps as a serial CG solution, except matrix-vector multiplications and inner products are performed concurrently to reduce execution time. To compute the scaling factors for the search direction and solution vectors, the inner product of the residual with itself is performed in parallel and the results are reduced, summed, and broadcast to all participating processes. At the end of a given iteration, the search direction sub-vectors owned by individual processes are broadcast to the set of participating processors in a ring topology so that each process has the full search direction for the next iteration. CG parallelization is synchronous; hence it has the benefit of not degrading the iterative convergence rate of serial calculations. The parallelization penalty comprises communication of the residuals' inner products and the search direction sub-vectors to other processes, but this is balanced with the benefits from reduced execution time due to concurrency.

5. COMPUTATIONAL RESULTS

Thus far our research has focused on the approach outlined in Sec. 4.2. In this section, measured performance via numerical experiments is presented, related to solving the system of equations with the BJ and CG schemes. The goals of these experiments are to verify the parallel algorithms and to identify challenges to achieving good parallel performance. The message passing interface (MPI) is used to implement all parallel instructions.

Test runs were performed on the LION-XO distributed memory cluster at Penn State University [21]. LION-XO does not run in dedicated mode, and resource contention has been observed to have serious impacts. Users running parallel jobs compete for bandwidth over the network and through the switches and for memory and access to the memory. LION-XO is comprised of 132 computing nodes. 40 compute nodes are dual AMD Opteron processors rated at 2.4 GHz. The other 92 nodes are quad AMD processors rated at 2.6 GHz. All nodes are connected by a gigabit Ethernet switch and an Infiniband network. MPI jobs on LION-XO are limited to 16 nodes and 2 processors per node. We used only the dual processor machines to reduce the number of network communications and thereby the total communication time. These nodes have 8 GB of memory.

AMD Opteron machines have non-uniform memory access (NUMA) designs. For nodes with two processors, each processor has its own memory controller and a directly connected physical memory set. Communication with other memory sets is performed over a HyperTransport link. There is no policy on LION-XO for how the memory of a particular job is allocated to the physical memory space.

Numerical experiments' results are provided for four one-group test cases of AHOT-N order zero, Λ=0, employing the parallel BJ and CG methods. All problems use a 20×20×24 Cartesian mesh, two materials, and a fixed source distribution. Table I provides further information about the cross section data and the fully symmetric angular quadrature orders. Therefore the memory requirement for matrix A alone is 9600^2 double precision words, or 703 MB. Additional memory is necessary for the other variables. However, these are much smaller and add relatively little burden to the system compared to A. In the case of SI calculations, memory requirements are of the order of the number of spatial cells for cell data such as the material and source maps. SI memory is less than 1 MB for the test cases proposed.

Table I. Parameters for four test cases

Case	σ^t_1	σ^t_2	σ^s_1	σ^s_2	S_N
1	0.75	0.50	0.45	0.30	8
2	1.00	2.00	0.80	1.50	12
3	2.00	3.00	1.80	2.00	12
4	4.00	3.00	4.00	3.00	16

The parallel solution results are given as relative speedup S_P and efficiency E_P using the execution times with a single process T_1 and with P processes T_P.

$$S_P = T_1 / T_P \tag{17}$$

$$E_P = S_P / P \tag{18}$$

Generally, a program must handle two potential bottlenecks while running on a non-dedicated system such as LION-XO. First, all users performing parallel calculations compete for time on network and through the switches for message communication. During heavy system load, this can cause significant variance in execution time of parallel jobs. Second, each program on a node contends with others for access to the physical memory space as opposed to virtual memory on disk. Moreover, even when our program is occupying both processors on a single node, the memory intensive algorithm requires more communication between the cache and memory. On LION-XO, an MPI job occupying both processors of a single node is managed with shared memory, i.e., the data and instructions share a memory address space, over both sets of the physical memory. Combined with the NUMA architecture of AMD Opterons, these two features create the potential problem that the two memory controllers of a node must communicate more frequently, adversely affecting performance. We have encountered these problems with the new algorithm, but serial SI calculations that require much less memory have been relatively immune.

Results of the BJ experiments on the LION-XO system are given in Figs. 2 and 3. The speedup and efficiency are based on the P=2 case because the case with a single block is equivalent to

direct solution. All four cases show the same trend in efficiency: dropping for small *P*, increasing briefly, then leveling for high *P*. With large blocks the program must cope with the problems described for the message communication and the cache. As more processes are introduced, the memory requirement per process drops like 1/*P*. Furthermore, the reduction in the size of the blocks results in fewer operations necessary to invert the diagonal block, asymptotically at a cubic rate for increasing number of processes.

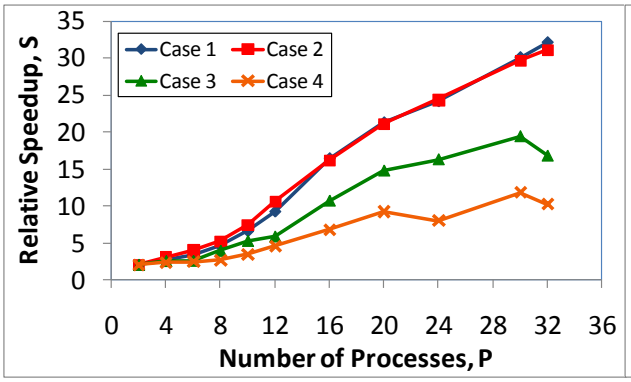

Figure 2. BJ test cases' parallel speedup

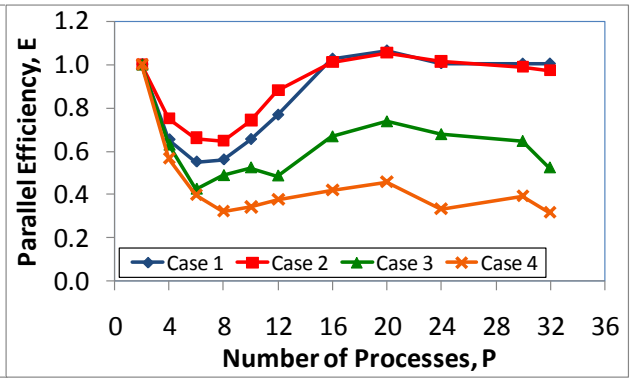

Figure 3. BJ test cases' parallel efficiency

The BJ method is hindered by the increasing number of iterations with increasing number of processes shown in Table II. When divided into smaller blocks, the scheme consumes more iterations. However, the average time spent per iteration decreases. The competing effects are evident in Fig. 3. Cases 1 and 2 both need less than 30 iterations to converge even at *P*=32, and the iteration time has decreased so much that the parallel efficiency is approximately 100%. Cases 3 and 4 do not benefit from the same recovery; the growth rate of the number of iterations dominates the decreasing iteration time.

Table II. Parallel BJ number of iterations

Case	Number of Processes										
	2	**4**	**6**	**8**	**10**	**12**	**16**	**20**	**24**	**30**	**32**
1	11	12	13	14	15	16	18	19	20	20	20
2	15	15	16	17	19	20	24	27	28	29	29
3	30	33	36	43	52	57	75	90	100	103	105
4	62	83	111	141	177	199	266	324	367	383	384

Parallel CG results on LION-XO are given in Figs. 4 and 5. The results presented are a reduction from several repetitive runs for each case and number of processors intended to reduce variance in the measured execution time. The data was reduced to what we believe is best representative of typical LION-XO performance. In many cases, particularly for a larger number of processes,

very little execution time variance was encountered, in contrast to runs with fewer processes. One potential explanation for this is the aforementioned cache-memory communication bottleneck. Increasing the number of processes reduces the amount of data handled by each process. Consequently the cache needs fewer exchanges with memory, which should both improve performance and make the execution time more predictable.

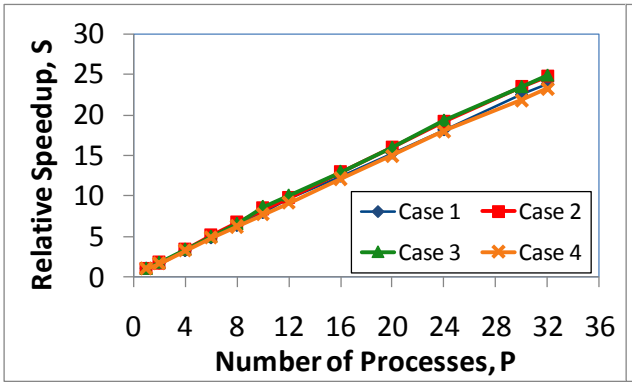

Figure 4. CG test cases' parallel speedup

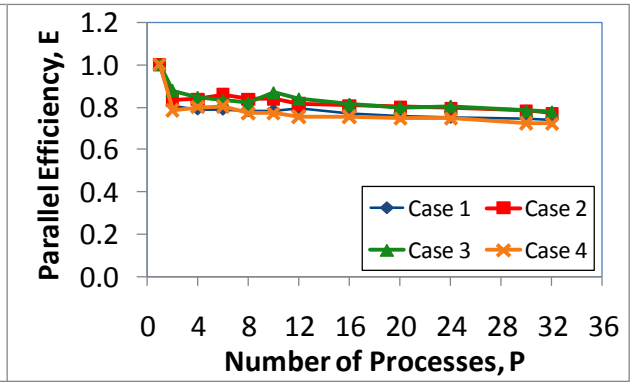

Figure 5. CG test cases' parallel efficiency

The results exhibit the expected general trends—an increase in speedup and decrease in efficiency with increasing number of processes. Although these results are relative only to serial CG solution, important lessons can be learned. First, because the parallel CG method is synchronous, the number of iterations does not increase as it did in the case of BJ. As seen in both Figs. 4 and 5, the curves assume an asymptotic trend. The decrease in efficiency is expected with parallel grain refinement. However, reaching an asymptotic regime indicates the potential for high scalability. Second, from the argument in Sec. 3, we know that CG does not suffer the same, or as severe of consequences from varying typical problem parameters including the number of discrete ordinates or increasing the scattering ratio.

Our ultimate goal is to improve upon the SI scheme with a novel algorithm for massively parallel computing architectures and very large problems in terms of number of cells. Since constructing A is computationally very expensive, comparing full execution time between the methods shows SI to be the clear victor in most cases. Table III compares execution times between the serial CG and SI calculations. The CG solution, for $\Lambda=0$, uses fewer iterations and less time per iteration than SI in all cases studied. Coupled with the known speedup from Fig. 4, we project that serial SI's advantage diminishes with increasing P. In the case of purely scattering materials, the full execution time is already better for the new algorithm. Yet for it to be competitive with SI for all situations—and further, to be competitive with parallel SI algorithms such as the wavefront approach—two things must happen: 1. Significantly decrease matrix construction time via parallel algorithm and 2. Use the advantage the CG method has over the SI scheme, essentially replacing the longer grind time with fewer, shorter CG iterations.

Table III. Serial SI and CG execution time and number of iterations

Case	Source Iterations	Solve SI (s)	Construct Matrix (s)	CG Iterations	Solve CG (s)
1	31	64	1771	14	6
2	66	288	3521	22	10
3	493	2115	3416	50	22
4	2140	16148	5809	83	29

6. CONCLUSIONS

All the parallel performance results obtained so far indicate that significant speedup can be achieved by the parallel solution of the linear system derived from the iteration Jacobian matrix. Two approaches can be applied to solving the system of equations in parallel, BJ and CG. The latter is preferred because the number of iterations does not change with increasing number of processors, and the time per CG iteration is typically smaller than the time per source iteration.

A massively parallel algorithm for constructing a matrix for the entire domain is still under development as we attempt to balance all the operations of a full differentiation sweep equally among an increasing number of processes. Angular and spatial domain decompositions should work, but will not yield sufficient speedup. A more scalable parallel construction algorithm is essential for the approach outlined in Sec. 4.2 to be viable. Matrix construction requires a significant amount of execution time as well as memory in serial implementation. The matrices for the test problems—which are relatively small compared to typical transport problems—required thousands of seconds to construct in serial and nearly a gigabyte of memory to store.

We have not presented results on the approach presented in Sec. 4.1 in this paper, but we expect to do so in the future; full development of the operators introduced is near completion. Once complete, this method must be examined for its spectral properties and its scalability.

ACKNOWLEDGMENTS

The funding for this work has been provided by Los Alamos National Laboratory.

REFERENCES

1. B. R. Wienke and R. E. Hiromoto, "Parallel S_N Iteration Schemes," *Nuclear Sciences and Engineering*, **90**, pp.116–123 (1985).
2. Y. Y. Azmy, "On the Adequacy of Message-Passing Parallel Supercomputers for Solving Neutron Transport Problems," *Proceedings of Supercomputing '90*, IEEE Computer Society Press, pp.693–699 (1990).
3. M. Yavuz and E.W. Larsen, "Iterative Methods for Solving *x-y* Geometry and S_N Problems on Parallel Architecture Computers," *Nuclear Science and Engineering*, **112**, pp.32–42 (1992).
4. R. S. Baker and R. E. Alcouffe, "Parallel 3-D S_N Performance for DANTSYS/MPI on the Cray T3D," *Proceedings of the Joint International Conference on Mathematical Methods*

and Supercomputing for Nuclear Applications, Saratoga Springs, NY, USA, October 5–9, **1**, pp.377–393 (1997).
5. R. S. Baker and K. R. Koch, "An S_N Algorithm for the Massively Parallel CM-200 Computer," *Nuclear Science and Engineering*, **128**, pp.312–30 (1998).
6. A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications," *International Journal of High Performance Computing Applications*, **14**, pp.330–346 (2000).
7. R. Mattis and A. Haghighat, "Domain Decomposition of a Two-Dimensional S_N Method," *Nuclear Science and Engineering*, **111**, pp.180–196 (1992).
8. G. E. Sjoden and A. Haghighat, "PENTRAN – A 3-D Cartesian Parallel S_N Code with Angular, Energy, and Spatial Decomposition," *Proceedings of the Joint International Conference on Mathematical Methods and Supercomputing for Nuclear Applications*, Saratoga Springs, NY, USA, October 5–9, **1**, pp.553–562 (1997).
9. M. R. Dorr and C. H. Still, "Concurrent Source Iteration in the Solution of Three-Dimensional, Multigroup Discrete Ordinates Neutron Transport Equations," *Nuclear Science and Engineering*, **122**, pp.287–308 (1996).
10. H. F. Jordan and G. Alaghband, *Fundamentals of Parallel Processing*, Prentice Hall, Pearson Education, Inc., Upper Saddle River, NJ, USA (2003).
11. Y. Y. Azmy, "Multiprocessing for Neutron Diffusion and Deterministic Transport Methods," *Progress in Nuclear Energy*, **31**, No. 3, pp.317–368 (1997).
12. P. Humbert, "Parallelization of PANDA Discrete Ordinates Code Using Spatial Decomposition," *Proceedings of PHYSOR-2006, ANS Topical Meeting on Reactor Physics* [on CD-ROM], Vancouver, BC, Canada, September 10–14 (2006).
13. A. Böhm, J. Brehm, and H. Finnemann, "Parallel Conjugate Gradient Algorithm for Solving the Neutron Diffusion Equation on SUPRENUM," *Proceedings of the 5th International Conference of Supercomputing*, Cologne, West Germany, pp. 163–171 (1991).
14. A. Gupta and R. S. Modak, "On the Use of the Conjugate Gradient Method for the Solution of the Neutron Transport Equation," *Annals of Nuclear Energy*, **29**, pp.1933–1951 (2002).
15. G. S. Chen and R. D. Sheu, "Application of Two Preconditioned Generalized Conjugate Gradient Methods to Three-Dimensional Neutron and Photon Transport Equations," *Progress in Nuclear Energy*, **45**, No. 1, pp.11–23 (2004).
16. J. A. Fischer, "Comparison of Spatial and Angular Domain Decomposition Algorithms for Discrete Ordinates Transport Methods Using Parallel Performance Models," *Progress in Nuclear Energy*, M.S. Thesis, The Pennsylvania State University (2003).
17. Y. Y. Azmy, "The Weighted Diamond-Difference Form of Nodal Transport Methods," *Nuclear Science and Engineering*, **198**, pp.29–40 (1996).
18. Y. Y. Azmy, "A New Algorithm for Generating Highly Accurate Benchmark Solutions to Transport Test Problems," *Proceedings of the XI ENFIR/IV ENAN Joint Nuclear Conferences*, Pocos de Caldas Springs, MG, Brazil, August 18–22 (1997).
19. E. Anderson, et al., *LAPACK User's Guide*, 3rd Ed., made available online http://www.netlib.org/lapack/lug/index.html, Last Accessed February 24, 2009 (1999).
20. Y. Saad, *Iterative Methods for Sparse Linear Systems*, 1st Ed., made available online http://www-users.cs.umn.edu/~saad/books.html, Last Accessed February 21, 2009 (1996).
21. "LION-XO PC Cluster", http://gears.aset.psu.edu/hpc/systems/lionxo/, Last Accessed February 24, 2009 (2009).