Project 09-797

A Novel Algorithm for Solving the Multidimensional Neutron Transport Equation on Massively Parallel Architectures

Integrated University Programs

Dr. Yousry Y. Azmy
North Carolina StateUniversity

Keith Bradley, Technical POC

National Nuclear Security Administration

Final Report NEUP Project 09-797

A Novel Algorithm for Solving the Multidimensional Neutron Transport Equation on Massively Parallel Architectures

Submitted by Project Principal Investigator

Yousry Y. Azmy
Department of Nuclear Engineering
North Carolina State University

Contents

Abstract

We employ the Integral Transport Matrix Method (ITMM) as the kernel of new parallel solution methods for the discrete ordinates approximation of the within-group neutron transport equation. The ITMM abandons the repetitive mesh sweeps of the traditional source iterations (SI) scheme in favor of constructing stored operators that account for the direct coupling factors among all the cells' fluxes and between the cells' and boundary surfaces' fluxes. The main goals of this work are to develop the algorithms that construct these operators and employ them in the solution process, determine the most suitable way to parallelize the entire procedure, and evaluate the behavior and parallel performance of the developed methods with increasing number of processes, *P*. The fastest observed parallel solution method, Parallel Gauss-Seidel (PGS), was used in a weak scaling comparison with the PARTISN transport code, which uses the source iteration (SI) scheme parallelized with the Koch-Baker-Alcouffe (KBA) method. Compared to the state-of-the-art SI-KBA with diffusion synthetic acceleration (DSA), this new method—even without acceleration/preconditioning—is competitive for optically thick problems as *P* is increased to the tens of thousands range. For the most optically thick cells tested, PGS reduced execution time by an approximate factor of three for problems with more than 130 million computational cells on *P* = 32,768. Moreover, the SI-DSA execution time's trend rises generally more steeply with increasing *P* than the PGS trend. Furthermore, the PGS method outperforms SI for the periodic heterogeneous layers (PHL) configuration problems. The PGS method outperforms SI and SI-DSA on as few as *P* = 16 for PHL problems and reduces execution time by a factor of ten or more for all problems considered with more than 2 million computational cells on *P* = 4,096.

1. Introduction

Solving the neutron transport equation is a computationally difficult task, thus providing very challenging and rewarding research opportunities. Furthermore, the advent of massively parallel computing environments has made the application of efficient schemes to solve the transport equation in parallel a high priority. The rapid growth in computing platforms is creating a demand for computational methods that can fully utilize new architectures' resources in an efficient, scalable manner. Researchers seek to balance the work load across all participating processors while not altering the serial algorithm in a manner that tends to diminish the benefits of dividing the computational load over multiple processors.

The parallelization of the transport equation has been researched intensively for over two decades. Early algorithms dealt with the decomposition of the energy [1] and angular [2],[3],[4] domains across processes. Yet distributing work based on energy or angular variable decomposition alone is inadequate when tens of thousands of processing elements (PEs) are available. This inadequacy is driven by the practical bounds to which the energy and angular variables are commonly refined. Nevertheless, extensive literature is available on these subjects, and the listed references serve as introductory sources for interested readers.

Alternatively, problems with ever-increasing numbers of spatial cells are highly desired to analyze physically larger regions and/or to employ finer spatial meshes. Therefore, spatial domain decompositions have been developed to use the greater number of unknowns to achieve high scalability on increasingly large computing clusters. Focus has rightly shifted to decomposition in multiple variables [5],[6],[7] with spatial domain decomposition [7],[8],[9],[10],[11],[12] of the within-group equations receiving the most attention.

For first-order, discrete ordinates neutron transport solvers on a structured spatial mesh, the Koch-Baker-Alcouffe (KBA) [10],[11], or wavefront, method has become a common choice for spatial domain decomposition of the within-group equations. For an *n*-dimensional (*n*=2, 3) system, the domain is mapped to an *n*−1 processor topology. Mesh sweeps are performed in parallel over individual sub-domains by communicating angular fluxes on sub-domain boundaries to downstream neighbors, and advancing along diagonal planes. The KBA method has been shown to be highly efficient and suffer small execution time penalty for communication. Further, it suffers small load imbalances to maintain synchronous behavior compared to serially performed mesh sweeps. In a synchronous algorithm, all independent processes simultaneously perform the same instructions as the serial version of the algorithm, but on different data. The work is reordered only in the sense that it is distributed for faster execution. The aggregate of all processes' instructions is equivalent to the serial algorithm, hence the all intermediate and final numerical values resulting from the computation match to within numerical precision. The KBA method has been implemented with traditional source iterations (SI) in the PARTISN transport code [13], and more recently it has been used as an efficient way to describe the action of the inverted transport operator necessary for a GMRES(m) scheme in Denovo [14]. Each code's iterative solution method is augmented with a separately parallelized diffusion synthetic acceleration (DSA) routine: as a true acceleration step in PARTISN and as a preconditioner for the GMRES wrapper in Denovo.

Conversely in an asynchronous algorithm, the instructions undergo reordering to achieve parallelism, causing intermediate and final results from the parallel computation to differ from their serial counterparts beyond numerical precision; hence the total number of operations performed can change with the number of participating processes. The parallel block Jacobi (PBJ) method [7],[8],[9] is an alternative spatial domain decomposition. A global domain is decomposed into smaller sub-domains. The solution of the transport equation is performed concurrently over each sub-domain and interface angular fluxes are shared between adjacent sub-domains. The exchanged angular fluxes serve as new boundary conditions for the independent sub-domain problems. The process is repeated until the global solution is adequately converged. The incoming fluxes into each sub-domain on internal interfaces lag by one iteration. Hence the method's PBJ pedigree. This algorithm is clearly asynchronous as solving the global problem would not divide the work into determining intermediate solutions over the sub-domains. An important point to recognize for PBJ is that the local solution method for each sub-domain is not restricted to SI [7], and could be replaced by a Krylov-based solver [12].

This project has focused on the development of a novel kernel for handling the local problem within the PBJ framework of the global problem based on the Integral Transport Matrix Method (ITMM). The ITMM seeks operators that act directly on the cells' scalar flux and incoming angular fluxes on the boundaries of the sub-domains. Such an approach abandons repetitive mesh sweeps and other expensive iterative routines. Considerable computational burden is shifted from the repetitive local iterations to up-front, highly concurrent operator construction. Formulating and storing the operators can become prohibitively costly in terms of computational resources. For a large problem, the global domain is thus spatially decomposed into sub-domains and solved with PBJ per the aforementioned description.

Significant improvement is attained when this method is slightly altered by further decomposing sub-domains into smaller ones, assigning multiple sub-domains to the same PE, and employing a parallel red-black Gauss-Seidel (PGS) iterative strategy to the global solution. The method and measured performance of this new approach constitute the bulk of this report.

2. Objective

The overall objective of this project is to devise a novel parallel algorithm suitable for solving the discrete ordinates approximation of the transport equation on massively parallel platforms. We achieved this goal by abandoning the mesh sweep algorithm that is at the core of traditional solution schemes, in favor of an equivalent formulation that naturally decomposes the spatial domain into sub-domains coupled only on subdomain interfaces. However, unlike earlier efforts, the solution of the transport equation within each sub-domain is not based on a mesh sweep but rather on direct or iterative solution of a matrix equation resulting from an equivalent integral formulation of the discretized transport operator.

Investigation of parallel algorithms for solving the discrete ordinates equations so far has focused on adapting traditional sequential solution schemes to multiprocessor environments. This was accomplished via: (a) energy domain decomposition of the outer iterations [1]; (b) angular domain decomposition of the inner iterations [2],[3],[4]; (c) spatial domain decomposition of the mesh sweep[7],[9],[10],[11],[14]; and (d) hybrid methods that combine some or all of the above [5],[6].

While these earlier efforts have achieved a measurable level of success for certain classes of problems and multiprocessor platforms, efficient solution of large transport problems, especially on massively parallel platforms, remain a formidable challenge. Using parallel performance models it was illustrated that transport computations are still computation-, not communication-bound [11] and [15]. In other words, refining and improving communication strategies, while undoubtedly beneficial, does not reduce execution time significantly because grind time still comprises a large fraction of the total computational burden.

It has been suggested before that mesh sweep-based solution algorithms, while ideal on serial platforms, are not adequate for multiprocessors, especially massively parallel architectures. [4] An alternative recursive algorithm based on constructing the exact transport matrix that operates on the scalar flux (or angular moments of the flux) in the same sense as the integral transport operator has been proposed earlier for multidimensional configurations, and has been illustrated on serial computers. [16] This operator comprises a full (dense) matrix, in contrast to the lower triangular matrix representing the mesh sweep, that when inverted on the first-collision source immediately produces the fully-collided scalar flux without the need for inner iterations. The matrix formulation is mathematically equivalent to the inner iteration scheme in that the two algorithms produce the same solution to within arithmetic precision and the mesh sweep convergence criterion.

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 compared to traditional algorithms. However, within the scope of this project we considered two novel approaches for solving this matrix equation for the scalar flux (and more generally flux angular moments) on multiprocessor computers with the objective of reducing execution time and per processor memory requirement.

Ultimately, our new approaches seek to be competitive with mesh sweep-based solution algorithms by replacing inner iterations with more efficient subspace iterative schemes, by reformulating the solution algorithm to reduce the effective grind time, and by eliminating the sequential bottleneck embodied in the mesh sweep algorithm. We succeeded in achieving this essential objective of the project, but we have also discovered new phenomena in the iterative behavior connected to our new algorithm that have not been reported, and perhaps never observed before. These pose new challenges that deserve further exploration without diminishing the magnitude of the accomplishments of this project.

3. Detailed Report on Project Tasks

In this section we provide a detailed description of the accomplishment of each task using the same numbering sequence as in the awarded proposal. Note that for practical reasons not foreseen at the time of composing the awarded proposal the tasks described below were not completed in the same sequential order as listed here.

3.1 Task 1 – Construct & Verify the Exact Matrix Operators

This research entails replacing the traditional inner source iterations to solve the within group transport equation with the construction of the *integral discrete ordinates* transport matrix and subsequent solution of the resulting system of equations. The construction stage relies on a single mesh sweep, which has been termed the *differential* mesh sweep for reasons that will become evident shortly.

3.1.1 Reformulating the Source Iteration Scheme

The starting point for this method is the within-group, fully discretized balance equation with $I \times J \times K$ spatial cells and N_t angular directions. After rearranging, the balance equation for each cell may be written with the known values on the RHS, [16]

$$\begin{aligned}\varepsilon^x_{nijk}\psi_{n,i_{out},j,k} + \varepsilon^y_{nijk}\psi_{n,i,j_{out},k} + \varepsilon^z_{nijk}\psi_{n,i,j,k_{out}} + \psi_{nijk} \\ = c_{ijk}\phi_{ijk} + \sigma^{-1}_{tijk}q_{ijk} + \varepsilon^x_{nijk}\psi_{n,i_{in},j,k} + \varepsilon^y_{nijk}\psi_{n,i,j_{in},k} \\ + \varepsilon^z_{nijk}\psi_{n,i,j,k_{in}},\end{aligned} \tag{3.1.1}$$

Where we used standard notation and defined the reciprocal of the computational cell's size in the x-direction,

$$\varepsilon^x_{nijk} \equiv \frac{|\mu_n|}{\sigma_{tijk}\Delta x_i} \tag{3.1.2}$$

with analogous definitions for the y- and z-directions. The distributed source is assumed isotropic and the angular subscript n is accordingly dropped from the q term. The balance equation is augmented by the diamond difference relations in each of the three directions,

$$\psi_{nijk} = \frac{1}{2}\left(\psi_{n,i_{out},j,k} - \psi_{n,i_{in},j,k}\right), \tag{3.1.3}$$

with analogous expressions for the y- and z-directions. The four equations for each cell can be arranged into a matrix system of equations by using the coefficients of the known quantities (RHS) and of the unknowns (LHS):

$$\begin{bmatrix} 1 & \varepsilon^z_{nijk} & \varepsilon^y_{nijk} & \varepsilon^x_{nijk} \\ 1 & -0.5 & 0 & 0 \\ 1 & 0 & -0.5 & 0 \\ 1 & 0 & 0 & -0.5 \end{bmatrix} \begin{bmatrix} \psi_{nijk} \\ \psi_{n,i,j,k_{out}} \\ \psi_{n,i,j_{out},k} \\ \psi_{n,i_{out},j,k} \end{bmatrix} = \begin{bmatrix} 1 & \varepsilon^z_{nijk} & \varepsilon^y_{nijk} & \varepsilon^x_{nijk} \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} c_{ijk}\phi^p_{ijk} + \sigma^{-1}_{tijk} q_{ijk} \\ \psi_{n,i,j,k_{in}} \\ \psi_{n,i,j_{in},k} \\ \psi_{n,i_{in},j,k} \end{bmatrix}. \tag{3.1.4}$$

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 *previous* iterate.

In the transport sweep, such a system typically is not constructed and solved, instead the DD equations are substituted into the balance equation, the cell average flux is computed, and the three outgoing edge fluxes are computed. However, in this work we abandon that approach and the recurring SI mesh sweeps necessary to converge a flux distribution consistent with the self-scattering source.

The vector of fluxes on the LHS of Eq. (3.1.4) is the desired solution in the SI scheme for each cell; the angular fluxes on the outgoing faces of the cell are used as incoming fluxes to the neighboring downwind cells. The coefficient matrix on the LHS of Eq. (3.1.4) is inverted and left-multiplied on both sides of the equation. Using *Mathematica* [17], this is accomplished analytically, yielding formulas in terms of the ε factors for the matrix elements. A single coefficient matrix remains:

$$\begin{gathered} \begin{bmatrix} \psi & \psi_{k_{out}} & \psi_{j_{out}} & \psi_{i_{out}} \end{bmatrix}^T_{nijk} = \boldsymbol{\Gamma}_{nijk} \begin{bmatrix} c\phi^p + \sigma_t^{-1} q & \psi_{k_{in}} & \psi_{j_{in}} & \psi_{i_{in}} \end{bmatrix}^T_{nijk}, \\ \boldsymbol{\Gamma}_{nijk} \equiv \begin{bmatrix} \gamma^{aa} & \gamma^{axy} & \gamma^{axz} & \gamma^{ayz} \\ \gamma^{xya} & \gamma^{xyxy} & \gamma^{xyxz} & \gamma^{xyyz} \\ \gamma^{xza} & \gamma^{xzxy} & \gamma^{xzxz} & \gamma^{xzyz} \\ \gamma^{yza} & \gamma^{yzxy} & \gamma^{yzxz} & \gamma^{yzyz} \end{bmatrix}_{nijk}. \end{gathered} \tag{3.1.5}$$

The superscripts notation of the 16 matrix elements in Eq. (3.1.5) has two parts. The first part describes the component to which the element is contributing in the vector of unknowns, and the second part describes the component it is multiplying in the vector of known quantities. The partial superscript a refers to the cell average flux and scattering and fixed source terms. The partial superscript xy refers to the outgoing/incoming edge flux averaged over an xy face, i.e., $z = constant$, and the superscripts for the other faces are interpreted analogously.

3.1.2 The Definition of Integral Discrete Ordinates Transport Matrix

With the mesh sweep the SI scheme solves for the LHS angular flux of Eq. (3.1.4) for all computational cells and all angles and uses the angular quadrature to determine a new scalar flux, [16]

$$\boldsymbol{\phi}^{v} = \boldsymbol{A}(\boldsymbol{C}\boldsymbol{\phi}^{p} + \boldsymbol{\Sigma}_t^{-1}\boldsymbol{q}), \tag{3.1.6}$$

where $\boldsymbol{\phi}^{v}$ is the new scalar flux iterate, $\boldsymbol{\phi}^{p}$ is the previous flux iterate, and $\boldsymbol{q}$ is the distributed source, all vectors of length equal to the number of cells. $\boldsymbol{C}$ is the scattering ratio diagonal matrix and $\boldsymbol{\Sigma}_t^{-1}$ is the reciprocal of the total interaction cross section diagonal matrix; each has as its dimension the number of cells, and $\boldsymbol{C}$ is the product of the diagonal matrices $\boldsymbol{\Sigma}_s\boldsymbol{\Sigma}_t^{-1}$. $\boldsymbol{A}$ is a coefficient matrix whose elements are constructed from the elements of the $\boldsymbol{\Gamma}$ matrix in the discretized transport equation (3.1.5). By assuming vacuum boundary conditions for the problem domain, incoming boundary flux that would otherwise appear on the RHS are neglected.

The $\boldsymbol{A}$ matrix is given by the iteration Jacobian of Eq. (3.1.6):

$$\frac{\partial \phi_{i,j,k}^{v}}{\partial \phi_{i',j',k'}^{p}} = a_{(i,j,k)(i',j',k')}c_{i',j',k'} \equiv j_{\phi(i,j,k)(i',j',k')} \tag{3.1.7.a}$$

and

$$\frac{\partial \boldsymbol{\phi}^{v}}{\partial \boldsymbol{\phi}^{p}} = \boldsymbol{A}\boldsymbol{C} \equiv \boldsymbol{J}_{\phi}. \tag{3.1.7.b}$$

By substitution, Eq. (3.1.6) can now be rewritten as:

$$\boldsymbol{\phi}^{v} = \boldsymbol{J}_{\phi}(\boldsymbol{\phi}^{p} + \boldsymbol{\Sigma}_s^{-1}\boldsymbol{q}). \tag{3.1.8}$$

[Note: division by the scattering cross section must be specially treated in cases of void or purely scattering regions.] Upon iterative convergence of Eq. (3.1.8), successive iterates of the scalar flux are equal in the iterative limit, i.e. the solution $\boldsymbol{\phi}^{\infty}$ satisfies the following relation [16],

$$\boldsymbol{\phi}^{\infty} = \left(\boldsymbol{I} - \boldsymbol{J}_{\phi}\right)^{-1}\boldsymbol{J}_{\phi}\boldsymbol{\Sigma}_s^{-1}\boldsymbol{q}. \tag{3.1.9}$$

where $\boldsymbol{I}$ is the identity matrix. As in the case of $\boldsymbol{\Gamma}$ for the per cell basis, in the SI scheme $\boldsymbol{J}_{\phi}$ is never constructed, and the solution is computed from successive mesh sweeps instead of solving the system of equations in (3.1.9). $\left(\boldsymbol{I} - \boldsymbol{J}_{\phi}\right)$ is called the *integral transport matrix* [18] because it represents an accumulation of coupling factors among all cells, in all directions. The same name is applied to the rest of the method because the remaining ITMM operators also are constructed with this explicit coupling among cells and domain boundaries.

3.1.3 The Differential Mesh Sweep

The impetus to the integral discrete ordinates approach begins with the construction of $\boldsymbol{J}_{\phi}$, followed by explicitly solving the system of equations in (3.1.9). One must perform a single mesh sweep along all discrete ordinates in the angular quadrature to construct $\boldsymbol{J}_{\phi}$. Instead of computing the cell average (center) and outward angular fluxes given the incoming fluxes, the cell average angular flux of one node is coupled to the cell average *scalar* flux in *all* upstream cells for a specific discrete ordinate. Ultimately, with the summation by the angular quadrature, the scalar flux in any given cell will be related to the scalar flux in all other cells, making the system fully coupled.

By differentiating the balance and DD system of equations (3.1.5) with respect to $\boldsymbol{\phi}^p$, one can demonstrate the aforementioned coupling [16]:

$$\frac{\partial \psi_{n,i,j,k}}{\partial \phi^p_{i',j',k'}} = \gamma^{aa}_{nijk} c_{ijk} \frac{\partial \phi^p_{i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{axy}_{nijk} \frac{\partial \psi_{n,i,j,k_{in}}}{\partial \phi^p_{i',j',k'}} + \gamma^{axz}_{nijk} \frac{\partial \psi_{n,i,j_{in},k}}{\partial \phi^p_{i',j',k'}} + \gamma^{ayz}_{nijk} \frac{\partial \psi_{n,i_{in},j,k}}{\partial \phi^p_{i',j',k'}} \qquad (3.1.10.a)$$

and

$$\frac{\partial \psi_{n,i,j,k_{out}}}{\partial \phi^p_{i',j',k'}} = \gamma^{xya}_{nijk} c_{ijk} \frac{\partial \phi^p_{i,j,k}}{\partial \phi^p_{i',j',k'}} + \gamma^{xyxy}_{nijk} \frac{\partial \psi_{n,i,j,k_{in}}}{\partial \phi^p_{i',j',k'}} + \gamma^{xyxz}_{nijk} \frac{\partial \psi_{n,i,j_{in},k}}{\partial \phi^p_{i',j',k'}} + \gamma^{xyyz}_{nijk} \frac{\partial \psi_{n,i_{in},j,k}}{\partial \phi^p_{i',j',k'}}. \qquad (3.1.10.b)$$

The y- and x-direction equations are written analogously to Eq. (3.1.10.b). The first term in each expression equals zero unless $i,j,k = i',j',k'$ because no formulaic relation exists among *all* cells' previous iterate of the scalar flux, just a relation individually per cell. On the other hand, the incoming angular fluxes at the faces of a given cell are equal to the outgoing angular fluxes from the three adjacent upstream cells. Thus the three incoming angular flux terms in each equation must be evaluated for the cell's upstream neighbors. Given Eqs. (3.1.10.a) and (3.1.10.b), it becomes clear why this type of mesh sweep is termed the differential mesh sweep, to be distinguished from the SI mesh sweep.

The algorithm implemented to construct the $\boldsymbol{J}_{\phi}$ matrix using the differential mesh sweep is tedious but straightforward, hence it is not replicated here but its details are reported in [19].

3.1.4 Properties Affecting J_{ϕ}

Given a problem configuration one can use SI or alternatively solve Eq. (3.1.9) with the factorization of $(\boldsymbol{I} - \boldsymbol{J}_{\phi})$ or with conjugate gradient iterations to obtain the group scalar flux distribution. Several expected relations between serial execution time and variable problem parameters have been confirmed by the research and are briefly highlighted and compared to similar tests for SI.

The matrix $\boldsymbol{J}_{\phi}$ is large and dense; it is a square matrix with dimension $N = I \times J \times K$. Refining the spatial mesh for a sample problem increases N while keeping fixed the overall physical dimensions of the problem. SI execution time grows linearly with N, but conjugate gradient execution time per iteration grows like N^2 because of the larger matrix and inner products. Additional cells require an additional row and column in $\boldsymbol{J}_\phi$, and matrix construction and direct solution times thus grow like N^2 and N^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_t, but the SI execution time per iteration grows linearly with N_t. Construction time for $\boldsymbol{J}_\phi$ also grows linearly with N_t. However, conjugate gradient iteration time and direct solution time are unaffected since the matrix size does not change with N_t.

The scattering ratio c 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 $\boldsymbol{J}_\phi$, the direct solution time, or the per iteration time for SI or conjugate gradient. As the domain is made optically thicker, the spectral radius of SI goes to c. Thus, the SI scheme displays a markedly slow convergence rate in highly scattering media that feature little leakage, as expected. The conjugate gradient method also requires a greater number of iterations when the scattering ratio of the host material increases and approaches unity, but the effect is considerably weaker, indicating better convergence properties than SI.

3.1.5 Nonzero Boundary Conditions

In the previous section we introduced the primary operator of the integral discrete ordinates transport method. Once $\boldsymbol{J}_\phi$ is constructed, solving the system of equations produces the scalar flux solution that would result from the fully converged SI. However, $\boldsymbol{J}_\phi$ is limited to vacuum boundary conditions. Given an incoming angular flux at the boundaries of the system, one needs an additional matrix operator to compute the consequent scalar flux distribution within the domain. Returning to Eq. (3.1.8), one can add another term to the RHS to account for nonzero boundary conditions:

$$\boldsymbol{\phi}^v = \boldsymbol{J}_\phi(\boldsymbol{\phi}^p + \boldsymbol{\Sigma}_s^{-1}\boldsymbol{q}) + \boldsymbol{K}_\phi\boldsymbol{\psi}_{in}. \tag{3.1.11}$$

The new term is the product of a matrix $\boldsymbol{K}_\phi$, whose construction is to be determined, and the vector $\boldsymbol{\psi}_{in}$. The vector $\boldsymbol{\psi}_{in}$ has the dimension of the number of cell boundary surfaces, $2(IJ + IK + JK)$, multiplied by the number of incoming ordinates to each surface, $N_t/2$. $\boldsymbol{K}_\phi$ is not necessarily a square matrix; its row dimension matches length of the $\boldsymbol{\phi}^v$ vector, and its column dimension matches length of the $\boldsymbol{\psi}_{in}$ vector. With the additional term on the RHS is:

$$\boldsymbol{\phi}^\infty = \left(\boldsymbol{I} - \boldsymbol{J}_\phi\right)^{-1}\boldsymbol{J}_\phi\boldsymbol{\Sigma}_s^{-1}\boldsymbol{q} + \left(\boldsymbol{I} - \boldsymbol{J}_\phi\right)^{-1}\boldsymbol{K}_\phi\boldsymbol{\psi}_{in}. \tag{3.1.12}$$

It is important to note that the differentiation with respect to $\boldsymbol{\phi}^p$ in Eq. (3.1.11) leads to the same definition of $\boldsymbol{J}_\phi$ as before due to the lack of dependence on $\boldsymbol{\phi}^v$ in the new term.

Writing out Eq. (3.1.5) elucidates the needed information to derive a construction scheme for $\boldsymbol{K}_\phi$:

$$\psi_{nijk} = \gamma_{nijk}^{aa}\left(c_{ijk}\phi_{ijk}^{p} + \sigma_{tijk}^{-1}q_{ijk}\right) + \gamma_{nijk}^{axy}\psi_{n,i,j,k_{in}} + \gamma_{nijk}^{axz}\psi_{n,i,j_{in},k} + \gamma_{nijk}^{ayz}\psi_{n,i_{in},j,k} \tag{3.1.13.a}$$

and

$$\psi_{n,i,j,k_{out}} = \gamma_{nijk}^{xya}\left(c_{ijk}\phi_{ijk}^{p} + \sigma_{tijk}^{-1}q_{ijk}\right) + \gamma_{nijk}^{xyxy}\psi_{n,i,j,k_{in}} + \gamma_{nijk}^{xyxz}\psi_{n,i,j_{in},k} + \gamma_{nijk}^{xyyz}\psi_{n,i_{in},j,k}. \tag{3.1.13.b}$$

The equations for outward fluxes in the x- and y-directions are analogous to (3.1.13.b). Unlike $\boldsymbol{J}_{\phi}$, differentiation with respect to the scalar flux is not used here. Instead we treat the boundary conditions as constants. Implicit boundary conditions, e.g., reflective or periodic, are handled iteratively: outgoing fluxes are directed back into the region and operated on by $\boldsymbol{K}_{\phi}$.

Because in the SI scheme the iteration Jacobian $\boldsymbol{J}_{\phi}$ will ultimately determine the scalar flux solution from the previous iterate of the scattering source and the fixed distributed source, the matrix $\boldsymbol{K}_{\phi}$ is constructed using only the last three terms on the RHS in Eqs. (3.1.13.a), (3.1.13.b), and their x- and y-analogues. During the differential mesh sweep used to construct $\boldsymbol{J}_{\phi}$, the same $\boldsymbol{\Gamma}$ matrix is used to compute and update elements of $\boldsymbol{K}_{\phi}$.

Equation (3.1.13.b) and analogous directional counterparts compute the outgoing angular flux of the given cell. Outgoing fluxes become the incoming fluxes for the three neighboring downwind cells. Therefore for any cell in the system, Eq. (3.1.13.b) can be used to recursively relate the cell's outgoing boundary fluxes for a given direction to the system's fixed boundary condition at all upstream cell boundary surfaces via its own incoming fluxes. Furthermore, as a sweep progresses the average angular flux in a cell also becomes coupled to the upstream system boundary conditions via the incoming fluxes at its own faces, indicated by Eq. (3.1.13.a).

Again, details of the construction algorithm for the $\boldsymbol{K}_{\phi}$ matrix are provided in [19].

3.1.6 Solving for Outgoing Angular Flux at the Boundaries

Outgoing angular fluxes are needed for the parallelization of the algorithm as described later. Therefore, operators to compute the outgoing angular flux due to contributions from the distributed fixed source, incoming angular flux boundary conditions, and scattering based on the scalar flux solution are derived. Observing Eq. (3.1.5), the desired form of the operators satisfies:

$$\boldsymbol{\psi}_{out} = \boldsymbol{J}_{\psi}(\boldsymbol{\phi} + \boldsymbol{\Sigma}_{s}^{-1}\boldsymbol{q}) + \boldsymbol{K}_{\psi}\boldsymbol{\psi}_{in}. \tag{3.1.14}$$

The construction of the $\boldsymbol{J}_{\psi}$ operator is a straightforward extension of the construction of $\boldsymbol{J}_{\phi}$, and it has the transposed dimensions and index ordering of $\boldsymbol{K}_{\phi}$.

$\boldsymbol{K}_{\psi}$ attenuates the angular flux for each direction individually without coupling to other directions. Allocating a matrix that has row and column dimensions equal to the length of the

$\boldsymbol{\psi}_{out}$ and $\boldsymbol{\psi}_{in}$ vectors would be unnecessarily large because the vast majority of elements, where $\widehat{\Omega}_{out} \neq \widehat{\Omega}_{in}$, would be zero. Thus, $\boldsymbol{K}_{\psi}$ is grouped into N_t square sub-matrices, each of which has dimension equal to the number of incoming/outgoing boundary surfaces for a single direction, $IJ + IK + JK$. This grouping implies that only elements of $\boldsymbol{K}_{\psi}$ that are stored are those that map incoming angular fluxes of a certain direction to outgoing angular fluxes of the same direction, $\widehat{\Omega}_{out} = \widehat{\Omega}_{in}$.

3.1.7 Additional Developments

3.1.7.1 Transposed Operators

Evident from their descriptions, each ITMM operator's row index corresponds to the current cell's order in the differential mesh sweep, and ITMM's column index corresponds to all upstream cells' indices. Constructing the operators in this manner is consistent with the equations as derived. However, constructing and using the transposed operators can yield a significant improvement in computational efficiency. Namely, codes programmed in the Fortran programming language have column-wise ordering in memory, i.e., the first (row) index varies faster than the second (column). Therefore, looping over elements of a matrix in this manner is more computationally efficient than the reverse order. Unfortunately this reverse order is more consistent with the ITMM equations as they are written. Taking the transpose of the ITMM equations allows the more efficient array index ordering to be utilized. This change does not affect how all the *vectors* are stored or used—the programmed code does not know the difference between a row vector and a column vector. Using transposed ITMM operators has been found to be a useful mechanism for reducing the execution time of the differential mesh sweep.

3.1.7.2 Extension to AHOT-N

The ITMM can be posed for any spatial discretization of the first-order form of the transport equation. First-order forms may be presented with various spatial discretizations, but traditionally they all involve source iterations manifested in the mesh sweep (with more recent implementations increasingly employing Krylov subspace iterations). To demonstrate this fact, with its broader implication that our new algorithm is far-reaching, an alternative spatial discretization, the Arbitrarily High Order Transport methods of the Nodal type (AHOT-N) spatial discretization was employed in an analogous derivation of the DD's ITMM operators.

In AHOT-N the spatial distribution of the flux over a computational cell is computed as a series of Legendre polynomials. The unknowns that are determined numerically as the solution of the discretized system of equations are the Legendre *spatial* moments of the angular flux within and on the faces of the cell. The spatial moment of the flux within the cell is defined by the integration over the volume of the cell of the spatially dependent angular flux multiplied by the Legendre polynomials in the three dimensions. The moments on the faces involve similar integrations over the transverse directions to the direction of interest for neutron transport. With these formal extensions the underlying discrete-variable equations constituting AHOT-N are analogous in structure, but with higher vector dimensionality than the DD equations. The steps comprising the derivation (and implementation) of the ITMM operators is thus a direct extension of the above described derivation that is fully detailed in [19].

3.1.8 Verification of ITMM Operators

Two approaches were identified early on for using the ITMM solution strategy. The approach we used for preliminary verification purposes is based on constructing the $\boldsymbol{J}_\phi$ matrix for problem configurations with vacuum boundary conditions then solving the resulting algebraic problem directly with parallel solvers. This can be accomplished in several ways, including the Block Jacobi (BJ) and parallel CG [20][21] methods.

3.1.8.1 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, requiring the quotient to be integer. 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 only once 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 parallelization of the serial solution.

3.1.8.2 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 with increasing P. 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.

3.1.8.3 Numerical Results

In this section measured performance via numerical experiments is presented, related to solving the system of equations with the BJ and CG schemes. The goal of these experiments is to verify the parallel algorithms and 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 [22]. LION-XO does not run in dedicated mode, and resource contention has been observed to adversely impact performance. 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 $\boldsymbol{J}_{\phi}$ alone is 96002 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 $\boldsymbol{J}_{\phi}$. 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 employed test cases.

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 reported 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.

$$S_P = T_1 / T_P \,,$$
$$E_P = S_P / P$$

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 the 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 MPI jobs on a single node are treated as shared memory. Combined with the NUMA architecture of AMD Opterons, these two features create the potential problem that 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. 3.1.1 and 3.1.2. 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: decreasing for small *P*, increasing briefly, then leveling off for large *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 *P*.

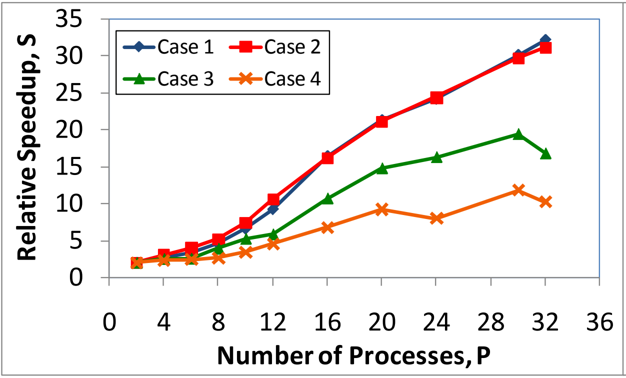


Fig. 3.1.1. BJ test cases' parallel speedup

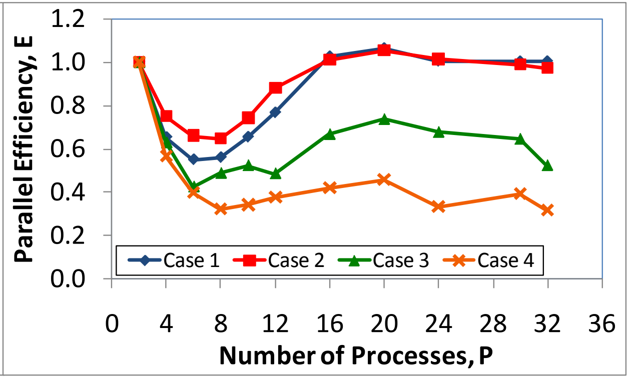


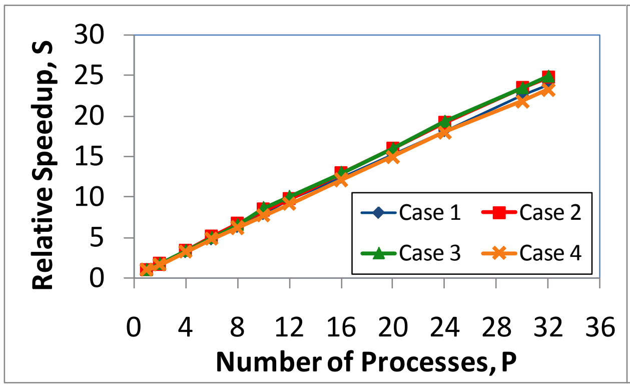
Fig. 3.1.2. BJ test cases' parallel efficiency

The BJ method is hindered by the increasing number of iterations with increasing *P* shown in Table 3.1.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. 3.1.2. 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.

Parallel CG results on LION-XO are given in Figs. 3.1.3 and 3.1.4. 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 due to runtime conditions. 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.

Table 3.1.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



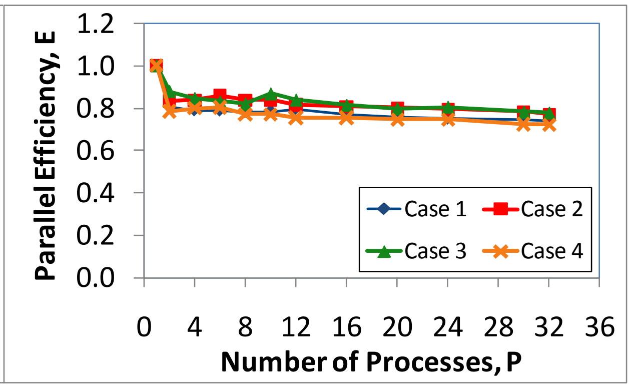


Fig. 3.1.3. CG test cases' parallel speedup **Fig. 3.1.4. CG test cases' parallel efficiency**

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. 3.1.3 and 3.1.4, 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 earlier argument, 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.

The numerical tests summarized in this section delineated the limitations of the direct solution approach treating the entire problem domain undivided in the computation of the $\boldsymbol{J}_{\phi}$ matrix and decomposing the matrix itself among participating processes in conducting the algebra of the solution process. The fact that the same converged solution was obtained with the new approach as with SI (to within iterative convergence) establishes correctness of the new algorithm and of the underlying $\boldsymbol{J}_{\phi}$ matrix. A more comprehensive verification of all ITMM operators is achieved in subsequent work where the domain decomposition was applied to the spatial extent of the test problems.

3.2 Task 2 – Examine Dependence of Convergence Rate on Spatial Decomposition

Observations of the iteration history for various cases attempted in the completion of this task revealed unexpected trends. In particular, the reduction in the iterative residual in the angular flux never settled into the monotonically decreasing profile characteristic of the asymptotic regime. After checking the code for potential bugs we decided to seek a justification for the observed behavior using spectral analysis tools. Dr. Dmitriy Anistratov and an undergraduate student summer-intern were recruited to conduct this research and their report on this task is attached as a separate PDF file.

3.3 Task 3 – Construct a preliminary parallel performance model

It is often valuable to predict the approximate execution time based on the size of the problem, the number of processes being deployed, and architectural specifications of the computing platform utilized. To achieve this predictive ability for an iterative algorithm, one must have knowledge of two items: 1) the anticipated number of iterations based on relevant problem parameters, and 2) a reliable estimate of the execution time per iteration given the workload assigned to each Processing Element (PE). The first item was to be completed within the scope of Task 2 but as reported above progress on that task was hampered by unexpected iterative convergence behavior in spite of the achieved progress on the ITMM spectral analysis reported in the attached PDF. The second item is the focus of this task, and completing it involves more thorough timing of individual sections of the code to determine how the number of cells per sub-domain, the number of angles, and the number of processes affect the execution time on a per iteration basis. The PIDOTS code utilized in the time measurements is described in the next section in the context of Task 4.

3.3.1 Detailed Execution Timing

The primary goal of decomposing the total execution time into several components is to determine which sections of the code consume the most amount of time and how the execution time for these sections is impacted by the size of the problem and the number of participating processes. Such detailed execution timing was sought from a fairly macroscopic perspective. That is, while perhaps possible to attain timing data for individual matrix-vector operations or point-to-point communications, the goal of this work was to observe the execution times of larger sections of the code that comprised many individual operations. Such divisions to isolate these portions were made based on the logical division of labor in the code (i.e., sub-routines), similarity of operations involved, and ability to reduce the number of variables on which that portion's execution time is dependent. That said, it was determined that execution could be divided into six components: construction of the ITMM operators; solution setup and factorization of the $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ matrix; computation of $\boldsymbol{\phi}$ using the factorized, stored integral transport matrix and source function; matrix-vector multiplications of $\boldsymbol{K}_{\phi}\boldsymbol{\psi}_{in}$ and $\boldsymbol{J}_{\psi}(\boldsymbol{\phi} + \boldsymbol{\Sigma}_s^{-1}\boldsymbol{q})$; matrix-vector multiplication of $\boldsymbol{K}_{\psi}\boldsymbol{\psi}_{in}$; and global reduction of maximum iterative error and communication of interfacial angular fluxes, $\boldsymbol{\psi}_{out} \rightarrow \boldsymbol{\psi}_{in}$. These will be denoted respectively as τ_{con}, τ_{itm}, τ_{ϕ}, $\tau_{\psi\phi}$, $\tau_{\psi\psi}$, and τ_{comm}. The remainder of this section documents the efforts to establish functional forms for these six components on the JPF system described in Sec. 3.4 below. The results were subsequently used to predict performance on the JPF system of PIDOTS for problems not used in constructing the performance model.

3.3.1.1 ITMM operator construction timing

The construction of the ITMM operators—$\boldsymbol{J}_{\phi}$, $\boldsymbol{K}_{\phi}$, $\boldsymbol{J}_{\psi}$, and $\boldsymbol{K}_{\psi}$—is accomplished via the differential mesh sweep over all discrete ordinates. At each cell during the mesh sweep along a single angle, each of the four operators is updated according to data accumulated from all upstream cells. That is, the operators are constructed in an element-wise fashion and share the same loops that comprise the sweep. Moreover, $\boldsymbol{J}_{\phi}$ and $\boldsymbol{J}_{\psi}$ share data accumulated in the $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ matrices. Likewise, $\boldsymbol{K}_{\phi}$ and $\boldsymbol{K}_{\psi}$ share the $\boldsymbol{XBC}$*, $\boldsymbol{YBC}$*, and $\boldsymbol{ZBC}$* matrices' data. Details on these intermediate matrices were reported in [19]. Because the operators share many instructions and pieces of data, it would be complicated to separate their construction into individual elements. Hence, a single construction time is measured for all operators. From matrix-dimensionality arguments it should be clear that the $\boldsymbol{J}_{\phi}$ matrix grows the fastest with increasing number of cells, like N^2 (not to be confused with '*N*' in S_N; *N* = *I J K*), but for many problems the $\boldsymbol{K}_{\phi}$ and $\boldsymbol{J}_{\psi}$ operators are largest due to their sizes' dependence on the number of incoming boundary surfaces, $N_b = IJ + IK + JK$, and the total number of angles, N_t.

The standard PIDOTS-PBJ code was modified to eliminate the solution portions and to repetitively, i.e. in a redundant loop, perform the differential mesh sweep in all directions. That is, the ITMM operators were constructed many times in an effort to reduce the variance in the computed average operator-construction time. This was especially important for faster executing problems, where timing a single cycle could be very close to the timer's resolution. Executions were performed multiple times on eight PEs (one node) of the JPF system, based on the fact that in the standard code these operators are constructed individually per sub-domain by the assigned process. Although operator-construction is a concurrent task across the processes, the construction time curves frequently exhibit a slow increase with increasing *P*. This is likely driven by contention of access to main memory by several PEs sharing a single memory controller. In an effort to model that effect, all executions were performed on eight PEs (one node) of the JPF system. The PEs performed the same differential mesh sweep. The measurements presented herein should capture the contention penalty of multiple PEs accessing the memory.

The construction time was measured for a set of cases by varying *N* and N_t. Table 3.3.I provides the problem size parameters considered and the number of redundant cycles the ITMM operators were constructed per case. Note that the *N* = 2,048 and 4,096 problems were limited by memory to lower-order quadrature sets. Further, fewer construction cycles were performed for the *N* ≥ 512 problems to reduce execution time. Lastly, the values of the cross sections, cell dimensions, and number of materials have no effect on construction time of the operators.

Table 3.3.I Cases considered for measuring ITMM operator-construction time.

N	N_t	Cycles
1	8, 24, 48, 80, 168, 288	100
8	8, 24, 48, 80, 168, 288	100
64	8, 24, 48, 80, 168, 288	100
128	8, 24, 48, 80, 168, 288	100
256	8, 24, 48, 80, 168, 288	100
512	8, 24, 48, 80, 168, 288	10
1,024	8, 24, 48, 80, 168, 288	10
2,048	8, 24, 48, 80	10
4,096	8, 24	10

Because the construction time has multiple dependencies that are difficult to separate, the measured timing results have been plotted against increasing *N* and increasing N_t. All results are based on one construction cycle—i.e., an average from multiple cycles actually measured.

The execution time of the sweep should depend on the number of cells and the number of angles. It is known from previous arguments that the size of $\boldsymbol{J}_\phi$ varies like N^2, the sizes of $\boldsymbol{K}_\phi$ and $\boldsymbol{J}_\psi$ vary like $N^{5/3}$, and the size of $\boldsymbol{K}_\psi$ varies like $N^{4/3}$. However, whereas the data accumulated over N_t sweeps is summed according to the quadrature to form $\boldsymbol{J}_\phi$, the other three operators instead combine the data to make a larger matrix. Therefore, the timed computational data for measuring the sweep time has been assumed to fit the polynomial function

$$\tau_{con} = \left(\alpha_1 N^2 + \alpha_2 N^{5/3} + \alpha_3 N^{4/3}\right) N_t \qquad (3.3.1)$$

The constant coefficients denoted by α_i , *i* = 1,2,3, are timing constants that represent the execution time required to compute the contribution to each element of the corresponding operator, and these will change with the computer system on which the code is tested. Noting the linear dependence on angle, the timing data accumulated according to the executions described by Table 3.3.I were normalized to a per angle basis. That is, construction times were divided by the corresponding N_t. The result was execution times that were only dependent on the number of spatial cells *N*. This data set was fit using the linear least squares approach to the parenthetical term in Eq. (3.3.1) to determine the timing constants. After reintroducing the angular dependence we obtain,

$$\tau_{con} = \left(2.15\times10^{-8} N^2 + 3.38\times10^{-8} N^{5/3} + 1.29\times10^{-7} N^{4/3}\right) N_t \, . \qquad (3.3.2)$$

Note that positivity of all coefficients is expected; the construction time is the sum of three lengths of time relating to different phases of the construction algorithm. A "goodness of fit" is desired to quantify the deviation or closeness of the fit and the actual measured data. The R^2

value provides such measure in many applications. data. Using the data accumulated for all *N* and N_t with the fit the ITMM operators' construction portion of execution time yields $R^2_{con} = 0.9970$.

The JPF construction timing model of Eq. (3.3.2) is plotted *versus* the contributing raw data, i.e. measured execution time per cycle, for cases of fixed N_t and varying *N* and for cases of fixed *N* and varying N_t. Figure 3.3.1 shows how the construction time varies as the operators are made larger by increasing *N*. The markers represent the measured times and the lines represent the fitted trends. All curves demonstrate the aforementioned superlinear behavior with *N*. Because the operators involve different powers of *N*, the cumulative behavior of this super-linearity is a combination of all. The fits are not perfect, but do increasingly well for large *N*. Deviations from the model by the raw data may be a result of the cache-memory communication costs, which are very small when the operators consume little memory, but increase with *N*. This is expected because the power-laws apply in an asymptotic sense, i.e., as $N \to \infty$. In Figure 3.3.2, the construction time curves are plotted versus N_t. The data agrees very well with the model of a linear relationship with N_t. The plot also shows more clearly how the model under-predicts construction time for small *N*.

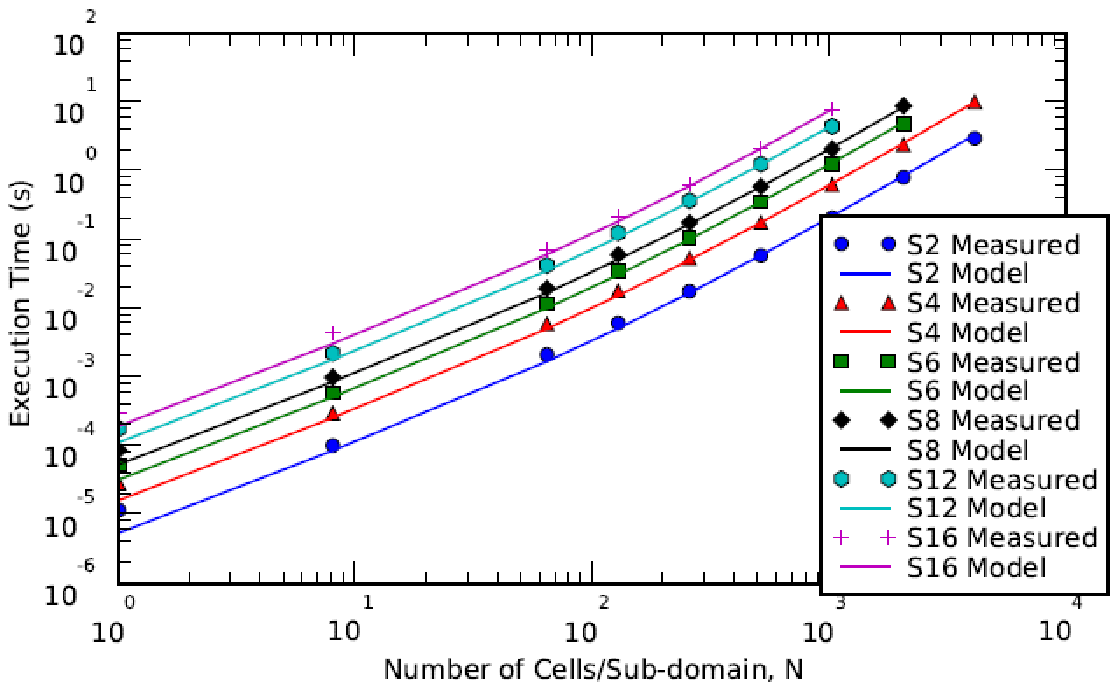


Fig. 3.3.1 ITMM Operators Construction Time *vs*. *N*, measured JPF data & Eq. (3.3.2) fit.

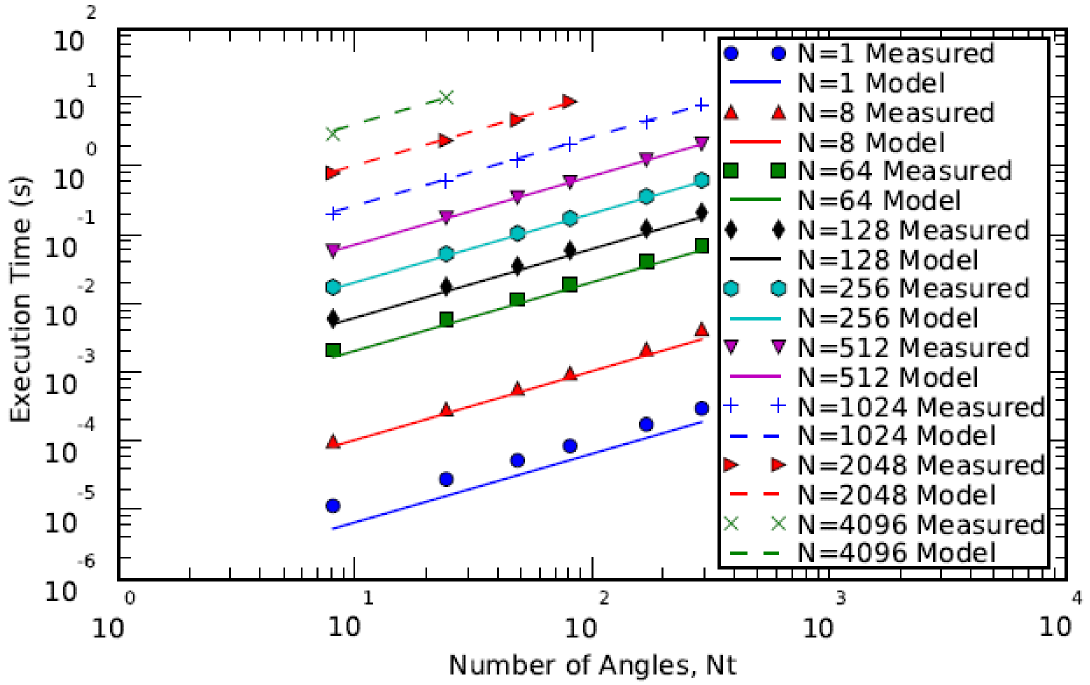


Fig. 3.3.2 ITMM Operators Construction Time *vs.* N_t, measured JPF data & Eq. (3.3.2) fit.

It must be noted that these relations can be applied in predicting the construction time for a given problem whether in strong and weak scaling senses. In weak scaling, the construction time computed from the selected formula is the estimate across all nodes, and in strong scaling, one need only determine how the increase in *P* affects *N*—e.g., doubling *P* halves *N*, that is substituted in Eq. (3.3.1).

3.3.1.2 Integral Transport Matrix setup and factorization timing

After $\boldsymbol{J}_{\phi}$ is constructed, several steps are performed before the iterative solution process begins. First, the product $\boldsymbol{J}_{\phi}\boldsymbol{\Sigma}_{s}^{-1}\boldsymbol{q}$ is computed and stored. Second, the integral transport matrix $(\boldsymbol{I}-\boldsymbol{J}_{\phi})$ is formed. Third, and most expensively, $(\boldsymbol{I}-\boldsymbol{J}_{\phi})$ is factorized and stored. In the case of isotropic scattering $(\boldsymbol{I}-\boldsymbol{J}_{\phi})$ is an *N*×*N* matrix, independent of N_t, so it is further assumed that the number of computations comprising the LU factorization grows like N^3. In a small test code, a dummy $\boldsymbol{J}_{\phi}$ matrix of different sizes *N* was manipulated and factorized repeatedly. The factorization process was timed to determine a proper relation of factorization execution time versus *N*. Using the same values of *N* as shown in Table 3.3.I, all but the *N* = 4,096 problem were executed three times with 100 redundant cycles per execution. The *N* = 4,096 problem was also executed three times but performed only 5 cycles per execution. Moreover, as in the case of the construction time, an attempt was made to capture memory-contention effects by repeating the operations on eight PEs of a single JPF node simultaneously.

The data was averaged for a single cycle and fit to the model

$$\tau_{itm} = \beta_1 N^3 + \beta_2 N^2 + \beta_3 N. \quad (3.3.3)$$

Again using linear least squares, the timing constants β were determined, and the JPF-specific model was developed:

$$\tau_{itm} = 8.42 \times 10^{-11} N^3 + 9.91 \times 10^{-9} N^2 + 2.14 \times 10^{-7} N. \quad (3.3.4)$$

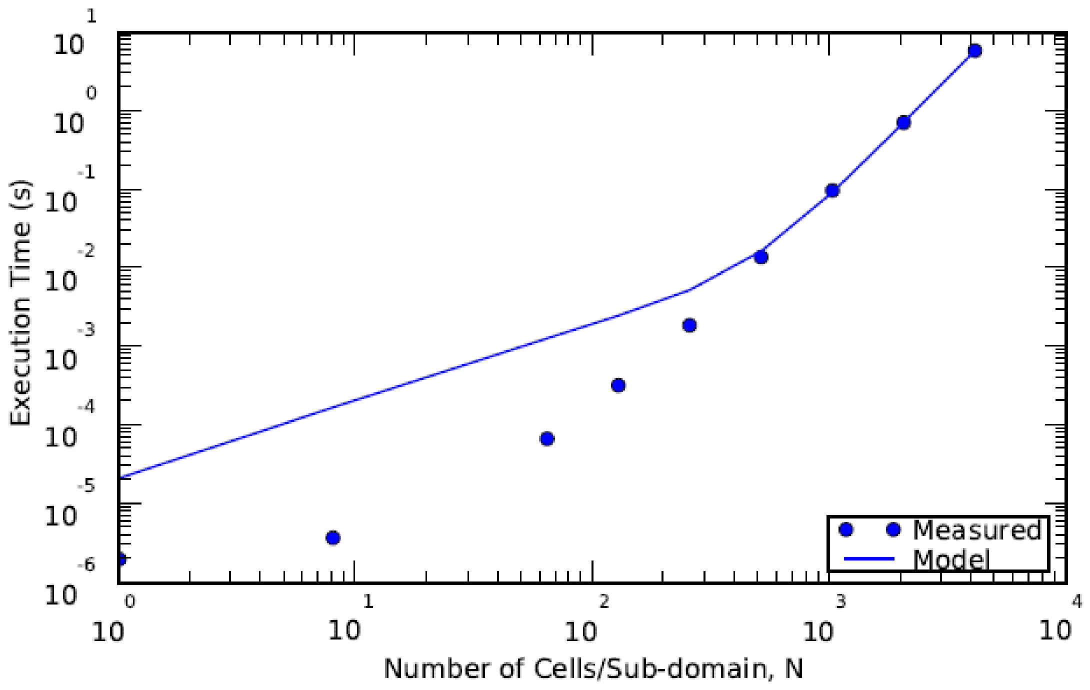


Fig. 3.3.3 Integral Transport Matrix Factorization Time *vs*. *N*, JPF data & Eq. (3.3.4) fit.

The goodness of fit for this section of the code was computed to be $R^2_{itm} = 0.9999$. Figure 3.3.3 displays the factorization timing results, comparing the measured times to the model. The model fit does not do a good job of predicting the time for small values of *N*, but improves greatly for larger *N*. As *N* increases, the factorization time dominates this portion of the PIDOTS execution, and the second two terms of Eq. (3.3.4) are small fractions of the first term. The model does well at capturing this effect as *N* increases.

Again note that the model can be applied in weak and strong scaling studies. In weak scaling, *N* is typically fixed with increasing *P*, and in strong scaling studies with PBJ, *N* decreases as the reciprocal of increasing *P*.

3.3.1.3 Solving for scalar flux timing

The scalar flux solution $\boldsymbol{\phi}$ is computed per Eq. (3.1.12) with the factorized $(\boldsymbol{I} - \boldsymbol{J}_{\phi})$ matrix. Solving this system with the factorized matrix should grow in number of operations like N^2.

Therefore, even though the solution process is repeated per global PBJ iteration due to the updated RHS vector, the factorization time—performed only once but growing like N^3—can still play a larger role in the total execution time if *N* is large.

A test code has been developed to measure the time associated with solving a dummy system of equations with a Gaussian elimination subroutine of LAPACK. Given the N^2 dependence, the presumed model equation is

$$\tau_\phi = \gamma N^2. \tag{3.3.5}$$

The code was executed on eight PEs (one node) of the JPF system three times. As in the case of the integral transport matrix setup/factorization timing, problems of sub-domain sizes *N* = 1, 8, 64, 128, 256, 512, 1,024, and 2,048 were executed three times with 100 cycles per execution, and the *N* = 4,096 problem was executed three times with 5 cycles per execution. The results were averaged for a single cycle, and the linear least squares technique fit the data yielded the following expression with $R_\phi^2 = 0.9835$,

$$\tau_\phi = 4.13 \times 10^{-9} N^2. \tag{3.3.6}$$

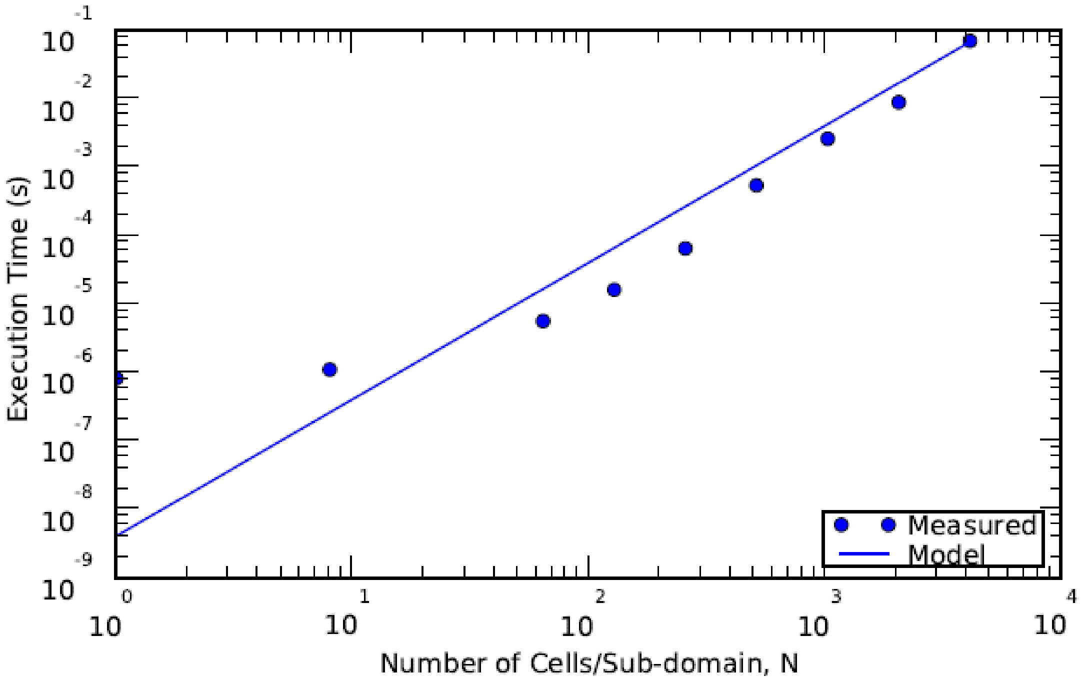


Fig. 3.3.4 Scalar Flux Solution Time *vs*. *N*, JPF data & Eq. (3.3.6) fit.

In Fig. 3.3.4, the execution time for solving the local algebraic system for $\boldsymbol{\phi}$ is shown. The fit is slightly over-predicting the time to complete the solution of the linear system, but the difference is very small in a practical sense—i.e., small fractions of a second. It is suspected here that when *N* is small, the operator fits more efficiently in cache and is more efficiently pipelined with optimized instructions for the processor. As *N* increases, these benefits decline and the curve more closely resembles the expected, quadratic form.

Like the expression for τ_{itm}, this expression can be applied in strong and weak scaling studies by either decreasing *N* in inverse-proportion with *P* (strong) or by holding *N* constant (weak).

3.3.1.4 Matrix-Vector multiplications involving angular flux timing

The iterative update process is comprised of four parts. The solution for the scalar flux with the factorized $(\boldsymbol{I} - \boldsymbol{J}_\phi)$ matrix has just been discussed. Additionally, three matrix-vector multiplications are performed every iteration. The first multiplication, $\boldsymbol{K}_\phi \boldsymbol{\psi}_{in}$, completes the update to the RHS of the $\boldsymbol{\phi}$-related system of equations. Once $\boldsymbol{\phi}$ is updated in each iteration, $\boldsymbol{\psi}_{out}$ is computed using two matrix-vector multiplications: $\boldsymbol{J}_\psi(\boldsymbol{\phi} + \boldsymbol{\Sigma}_s^{-1}\boldsymbol{q})$ and $\boldsymbol{K}_\psi \boldsymbol{\psi}_{in}$. Due to the fact that $\boldsymbol{K}_\phi$ and $\boldsymbol{J}_\psi$ have transposed dimensions, the matrix-vector multiplications involving these operators both consume NN_bN_t operations. Meanwhile, the $\boldsymbol{K}_\psi \boldsymbol{\psi}_{in}$ operation consumes $N_b^2 N_t$ multiplications. The previously used (in modeling construction time) asymptotic approximation $N_b \approx N^{2/3}$ was re-employed. The timing of these three multiplications has been divided into two components. The first two multiplications are grouped and timed together and are plotted versus the increasing size of the $\boldsymbol{K}_\phi$ and $\boldsymbol{J}_\psi$ operators, $N^{5/3}N_t$, and are consequently modeled with

$\tau_{\psi\phi} = \delta N^{5/3} N_t.$	(3.3.7)

The third multiplication is timed separately, plotted versus the size of the K_ψ operator, $\mathrm{N}^{4/3}\mathrm{N}_\mathrm{t}$, and modeled as

$\tau_{\psi\psi} = \epsilon N^{4/3} N_t.$	(3.3.8)

A test code was written to perform the above multiplications repeatedly, 1000 redundant cycles, for the different values of *N* and N_t given by Table 3.3.I. The code was executed three times on the JPF system, using eight PEs to, again, include the possible effects of memory contention. After averaging the data, the timing constants in Eqs. (3.3.7) and (3.3.8) were computed from a linear least squares fit. The $\tau_{\psi\phi}$ and $\tau_{\psi\psi}$ execution times are respectively modeled as

$\tau_{\psi\phi} = 2.54 \times 10^{-8} N^{5/3} N_t, \qquad R^2_{\psi\phi} = 0.9991.$	(3.3.9)

and

$\tau_{\psi\psi} = 4.28 \times 10^{-8} N^{4/3} N_t, \qquad R^2_{\psi\psi} = 0.9982.$	(3.3.10)

Fig. 3.3.5 shows the execution time for the $\mathrm{K}_\phi \psi_{\mathrm{in}}$ and $\mathrm{J}_\psi(\phi + \Sigma_s^{-1}\mathrm{q})$ multiplications, $\tau_{\psi\phi}$, *versus* $\mathrm{N}^{5/3}\mathrm{N}_\mathrm{t}$. Note that the fit lines up well with the data for larger systems. However, at some points, the execution time is significantly over-predicted by the model compared to the observed behavior. This is possibly a result of optimized operations when the system is very small—e.g., enhanced impact of pipelining instructions on the processor relative to larger problems.

The execution times of the $\mathrm{K}_\psi \psi_{\mathrm{in}}$ multiplications are plotted in Fig. 3.3.6 *versus* $\mathrm{N}^{4/3}\mathrm{N}_\mathrm{t}$. When the sub-domains are small, this multiplication will consume more time than the corresponding K_ϕ and J_ψ ones because of the large number of boundary surfaces relative to the number of cells in the sub-domain. As the operators increase in size the K_ϕ and J_ψ multiplications tend to dominate the execution time per iteration. The fit again does better at accurately predicting JPF execution time when the problem is larger, showing greater deviation as $\mathrm{N}^{4/3}\mathrm{N}_\mathrm{t}$ decreases.

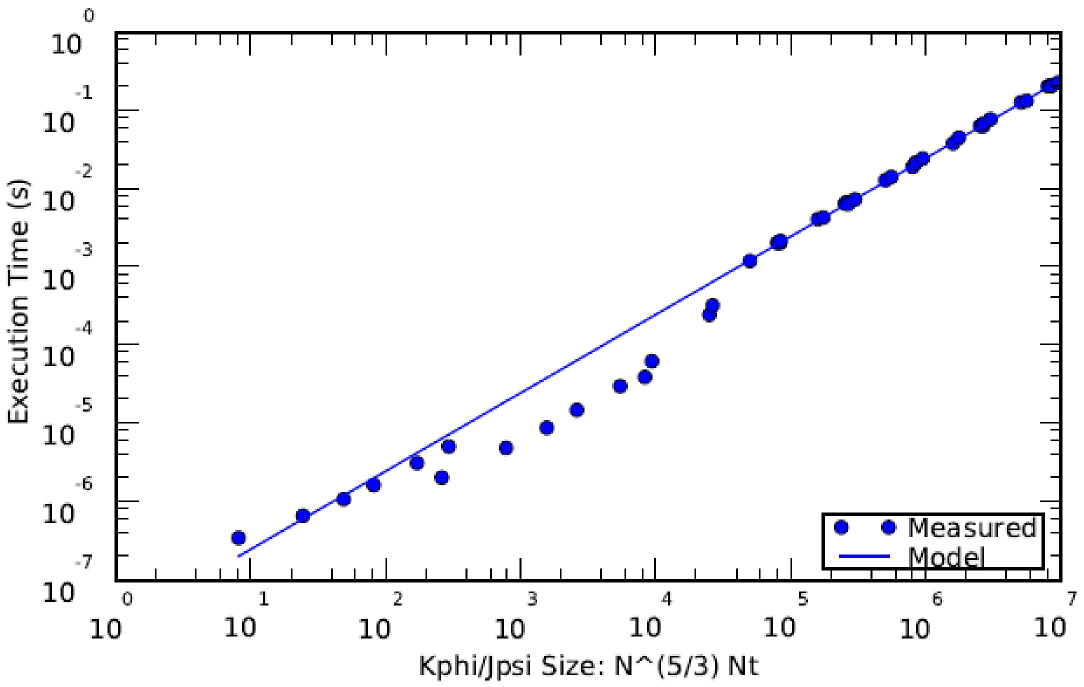


Fig. 3.3.5 K_ϕ and J_ψ Matrix-Vector Multiplications Time *vs*. $N^{5/3}N_b$, JPF data & Eq. (3.3.9) fit.

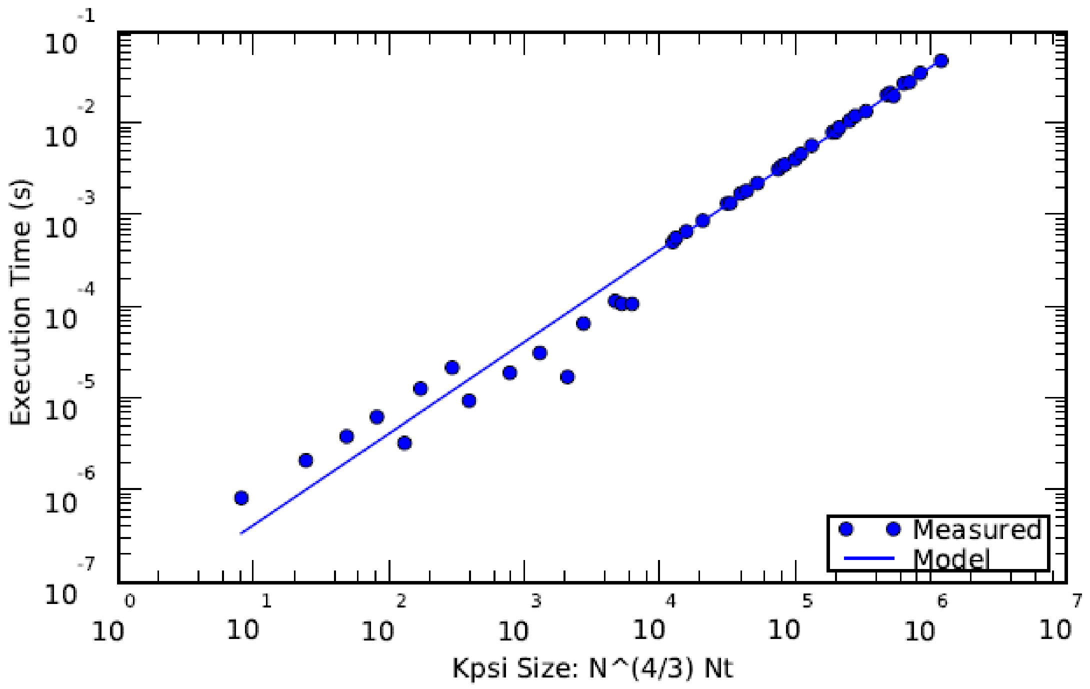


Fig. 3.3.6 K_ψ Matrix-Vector Multiplication Time *vs.* $N^{4/3}N_b$, JPF data & Eq. (3.3.10) fit.

As has been described before, weak scaling studies where P does not alter N should allow for a single estimate of these multiplication times that is independent of P. However, in strong scaling, the number of cells decreases reciprocally with increasing P and the number of boundary surfaces behaves like $\mathrm{P}^{2/3}$.

3.3.1.5 Communication timing

For all the preceding components, the dependence of timing on P has been implied by how P changes N and N_b. In fact, all measured timing results are taken from serial executions of test codes. However, the communication time is explicitly dependent on the number of processes because of the increasing cost of global communication when the process environment is enlarged and because of the increasing potential for contention over common network resources when more PEs are participating and sending more messages.

A final test code has been written to measure the communication time sending dummy messages in the same manner as in the PIDOTS-PBJ code. Vectors of lengths equal to those of the vectors of outgoing angular flux are sent to adjacent processes in a virtual Cartesian topology. Further, a global communication is used to reduce a dummy value of the largest iterative error per sub-domain to the maximum value among all processes and broadcast the result to all participating processes. These communication operations are repeated 1000 times per execution to reduce the potential noise in the timing of an individual cycle that can be attributed to contention for network resources. Test cases were performed on the JPF system in a weak scaling sense up to P = 4,096 for problems with N = 1, 8, 64, and 512 and S_4, S_8, S_{16}

quadrature sets. All cases were executed three times and the results are presented as an average of those runs.

Figure 3.3.7 shows the communication time curves for the 12 cases considered versus *P* for a single cycle—i.e., averaged results were divided by 1000. Note that all the curves show the general trend of increasing communication time with increasing *P*. Many cases exhibit a large jump in communication time when *P* increases from 8 to 16. This is expected because it marks the transition from communications being entirely intra-nodal (8 PEs per node) to relying on inter-nodal, thus requiring data transfer over the slower interconnection network. Cases with larger *N* and $\mathrm{N_t}$ do not demonstrate this effect with the same severity, if at all. This seems to indicate that longer vectors require significant transfer time even when being transmitted on a single node. Assuming *I* = *J* = *K*, the vectors being sent are of length $\frac{\mathrm{N_b}}{3}\frac{\mathrm{N_t}}{2}$, and Fig. 3.3.7 indicates almost entirely consistently that an increase in $\mathrm{N_b N_t}$ leads to a longer communication time. Instances of deviation from this expectation are likely caused by the potential noisiness of the timing data, which depends partially on system load at run time. Nevertheless, it must be noted that the communication time up to *P* = 4,096 appears to be a fairly small component of the execution time per PBJ iteration compared to the results of the preceding sections, especially $\tau_{\psi\phi}$.

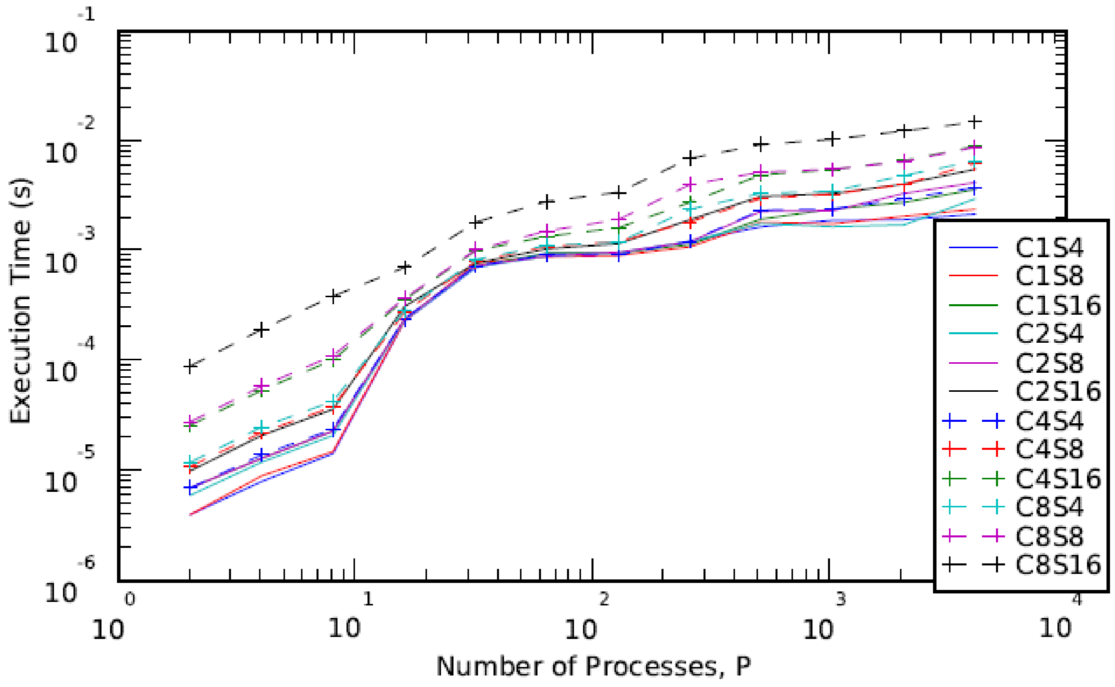


Fig. 3.3.7 PBJ Weak Scaling Communication Time per Iteration *vs*. *P*, varying *I*=*J*=*K*, S_N.

The results in Fig. 3.3.7 do not lend themselves to a simple fit dependent on *P* that could be readily used to estimate the communication time per PBJ iteration. Alternatively, Table 3.3.II tabulates the data from the execution of all cases. To estimate the communication time per iteration from Table 3.3.II one can directly read from the table the value if $\mathrm{N_b N_t}$ and *P* are equal to values in the rows and columns of the Table, respectively. Linear interpolation should provide a decent estimate of communication time per iteration when the desired value of $\mathrm{N_b N_t}$ or *P* is not listed.

Table 3.3.II Communication timing τ_{comm} data tabulated versus *P* and $N_bN_t/6$ (message length).

Msg. Length $N_bN_t/6$	Number of Processes, *P* 2	4	8	16	32	64
12	4.000E-06	8.000E-06	1.433E-05	2.317E-04	7.093E-04	9.500E-04
40	4.000E-06	9.000E-06	1.500E-05	2.317E-04	7.840E-04	8.860E-04
48	6.000E-06	1.200E-05	2.100E-05	2.380E-04	7.200E-04	8.967E-04
144	7.000E-06	1.300E-05	2.300E-05	2.390E-04	7.360E-04	9.573E-04
160	7.000E-06	1.300E-05	2.300E-05	2.417E-04	7.353E-04	9.007E-04
192	7.000E-06	1.400E-05	2.400E-05	2.410E-04	7.333E-04	9.213E-04
576	1.000E-05	2.100E-05	3.633E-05	3.137E-04	7.783E-04	1.050E-03
640	1.100E-05	2.233E-05	3.833E-05	2.767E-04	8.160E-04	1.101E-03
768	1.200E-05	2.500E-05	4.300E-05	2.873E-04	8.433E-04	1.128E-03
2304	2.600E-05	5.400E-05	1.023E-04	3.617E-04	1.013E-03	1.367E-03
2560	2.800E-05	5.900E-05	1.120E-04	3.740E-04	1.032E-03	1.533E-03
9216	9.000E-05	1.940E-04	3.890E-04	7.100E-04	1.837E-03	2.844E-03

Msg. Length $N_bN_t/6$	Number of Processes, *P* 128	256	512	1024	2048	4096
12	9.613E-04	1.223E-03	1.668E-03	1.911E-03	1.939E-03	2.174E-03
40	9.140E-04	1.093E-03	1.835E-03	1.771E-03	2.080E-03	2.409E-03
48	9.683E-04	1.230E-03	1.788E-03	1.672E-03	1.736E-03	2.994E-03
144	9.587E-04	1.150E-03	1.959E-03	2.421E-03	2.777E-03	3.647E-03
160	9.760E-04	1.217E-03	2.306E-03	2.337E-03	3.354E-03	4.216E-03
192	9.283E-04	1.227E-03	2.364E-03	2.423E-03	2.998E-03	3.799E-03
576	1.169E-03	1.930E-03	3.157E-03	3.346E-03	4.087E-03	5.564E-03
640	1.226E-03	1.843E-03	3.022E-03	3.303E-03	4.043E-03	6.326E-03
768	1.205E-03	2.406E-03	3.368E-03	3.520E-03	4.890E-03	6.625E-03
2304	1.634E-03	2.777E-03	4.950E-03	5.538E-03	6.682E-03	8.999E-03
2560	1.939E-03	4.032E-03	5.254E-03	5.578E-03	6.533E-03	8.835E-03
9216	3.434E-03	6.979E-03	9.343E-03	1.044E-02	1.247E-02	1.496E-02

3.3.2 Parallel Performance Model

Using the results of the previous section, a predictive parallel performance model can be formulated for estimating the execution time on the JPF system. The total predicted time τ_{tot} is the sum of all the components of the ITMM-PBJ solution detailed above,

$$\tau_{tot} = \tau_{con} + \tau_{itm} + (\tau_{\phi} + \tau_{\psi\phi} + \tau_{\psi\psi} + \tau_{comm})N_{its}, \quad (3.3.11)$$

where the τ_{con} and τ_{fac} components are performed once during construction and the remaining components are performed $\mathrm{N_{its}}$ times, the number of PBJ iterations. Note that none of the terms are presently given dependencies on *N*, $\mathrm{N_b}$, $\mathrm{N_t}$, or *P*. These are dependencies are included when considering whether the model is to be applied in a weak or strong scaling sense. In weak scaling, the first five terms are independent of *P*, and only depend on the problem size parameters:

$$\begin{aligned}\tau_{wk} = \tau_{con}(N, N_b, N_t) + \tau_{fac}(N) \\ + \left[\tau_{\phi}(N) + \tau_{\psi\phi}(N, N_b, N_t) + \tau_{\psi\psi}(N_b, N_t) \right. \\ \left. + \tau_{comm}(N_b, N_t, P)\right]N_{its}.\end{aligned} \quad (3.3.12)$$

These dependencies have been further simplified by the equations in the preceding sections that accompany the power fit trend lines. The model is adjusted in strong scaling, and the first five terms are additionally dependent on *P*, which scales the global problem size *N*.

$$\begin{aligned}\tau_{wk} = \tau_{con}(N, N_b, N_t, P) + \tau_{fac}(N, P) \\ + \left[\tau_{\phi}(N, P) + \tau_{\psi\phi}(N, N_b, N_t, P) + \tau_{\psi\psi}(N_b, N_t, P) \right. \\ \left. + \tau_{comm}(N_b, N_t, P)\right]N_{its}.\end{aligned} \quad (3.3.13)$$

The value of *P* has no effect on N_t, but it has an implied effect on *N* and N_b, where the number of cells decreases like $1/P$, and the number of boundary surfaces decreases like $1/P^{2/3}$.

This model currently does not estimate the number of PBJ iterations. N_{its} could be estimated from semi-analytic formulas based on empirical data. Alternatively, a more accurate prediction of the iterative behavior for varying problem parameters like *c* and *h* would likely be derived from a spectral analysis of the PBJ method. For our purposes known iteration counts from previously executed cases will be used to compare actual data with the model-estimated construction time and iterative solution time.

3.3.2.1 Weak Scaling Time Predictions

Using the weak scaling model defined by Eq. (3.3.12), the estimated execution times have been computed for the JPF weak scaling experiments. This study involved an 8×8×8 S_{16} base model. The h = 1.0 cm, c = 0.9 and 0.99 cases are chosen as examples to compare the estimated time based on the model versus the actual, observed execution times. Individual components of the model use the JPF-specific timing constants in Eqs. (3.3.2), (3.3.4), (3.3.6), (3.3.9), and (3.3.10) and use Table 3.3.II to determine the correct communication time.

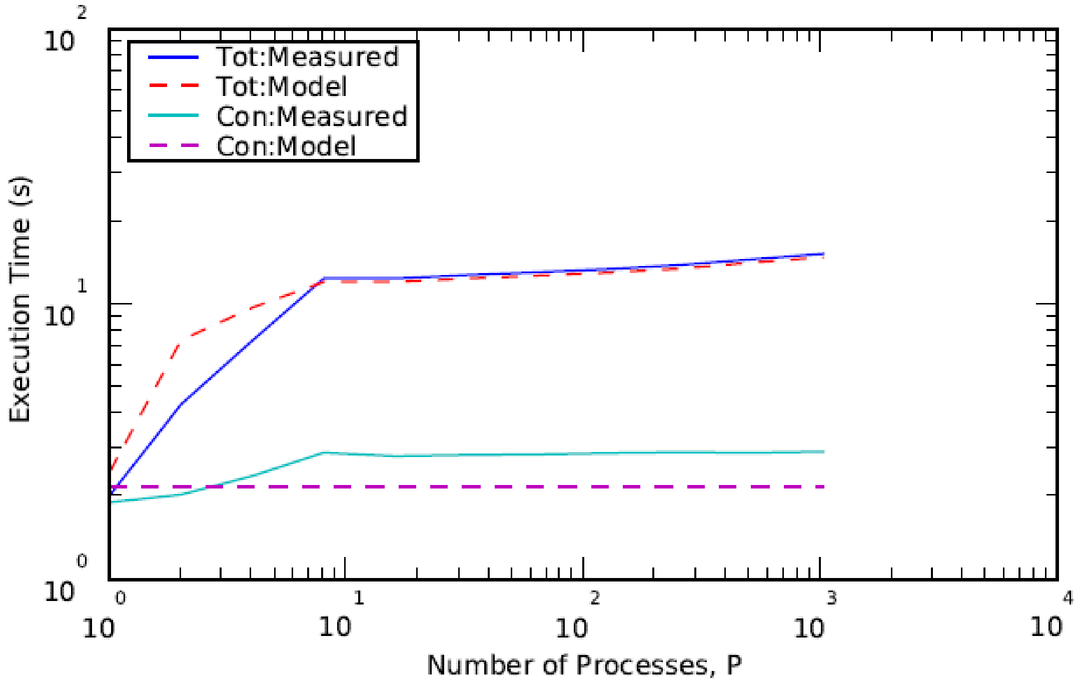


Fig. 3.3.8 Weak Scaling Exec. Time *vs*. *P* prediction, 8×8×8 base, S_{16}, *c* = 0.9, *h* = 1.0 cm.

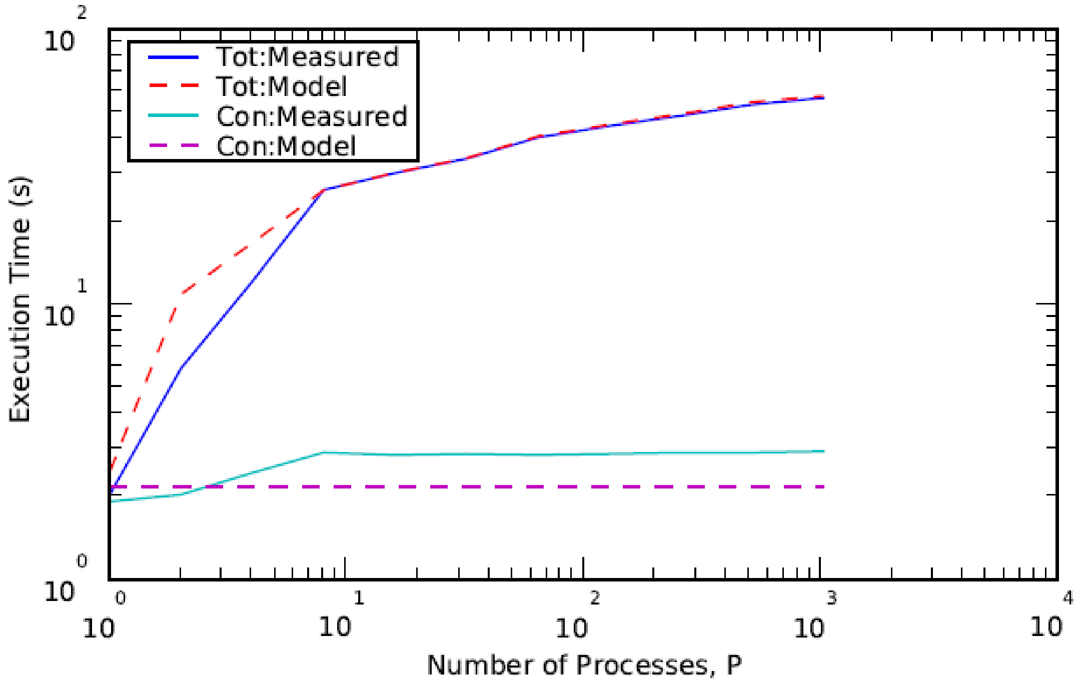


Fig. 3.3.9 Weak Scaling Exec. Time *vs*. *P* prediction, 8×8×8 base, S_{16}, *c* = 0.99, *h* = 1.0 cm.

Figure 3.3.8 shows the comparison for the *h* = 1.0 cm, *c* = 0.9 case up to *P* = 1,024. Likewise, the comparison for the *c* = 0.99 case is shown in Fig. 3.3.9. The construction time, $\tau_{\text{con}} + \tau_{\text{itm}}$, for both is the same, as *c* and *h* do not have an effect on these components of the code. Unfortunately, it is clear in both figures that the model currently devised under-predicts the construction time. This may be caused by not fully capturing the effect of memory contention or a deficiency in the model. In contrast, the execution time per iteration seems to be over-predicted by the model, leading to a noticeable difference when *P* is small. Yet as *P* increases, this over-prediction negates the construction time model's deficiency such that the total execution time computed from the performance model predicts the total execution time observed very well.

3.3.2.2 Strong Scaling Time Predictions

Using the strong scaling model defined by Eq. (3.3.7), the estimated execution times have been computed for the JPF strong scaling experiments. This study involved a 16×16×16 S_8 base model. The estimated execution time from the model is again compared to experimentally measured timing data from the *h* = 1.0 cm, *c* = 0.9 and 0.99 cases. Individual components of the model use the JPF-specific timing constants in Eqs. (3.3.2), (3.3.4), (3.3.6), (3.3.9), and (3.3.10). Table 3.3.II is used to determine the communication time, either by directly reading or by interpolating between points.

Figure 3.3.10 shows the results from the *c* = 0.9 case, and Fig. 3.3.11 shows the results from the *c* = 0.99 case. The performance model of Eq. (3.3.13) estimates execution times for strong scaling studies reasonably well. It is clear from both figures that the model over-predicts execution times for smaller values of *P*, the largest difference occurring at *P* = 1, where the construction time was most largely overestimated. The benefit of strong scaling studies is that the increase in *P* reduces the memory burden per PE. This seems to help the predictive model more accurately resemble reality and consequently the predicted construction times deviate less from the actual, observed construction times, and the total execution time is accurately predicted. Note that the measured execution time is rounded up to 0.001 s when *P* is very small due to a coded output limit to the clock's precision when such executions were originally performed; hence the curve assumes a flat line.

3.3.3 Remarks

By breaking up the PIDOTS-PBJ code into sections, a predictive parallel performance model was formulated for weak and strong scaling studies. All components, but the communication time, are only indirectly affected by changing *P*. Therefore, fits were based on the factor that most dominates the operations of that component. Communication timing is estimated using tabulated data that covers a wide range of vector lengths and *P*.

The results indicate that the current models predict total execution time in both weak and strong scaling studies sufficiently well. The model tends to under-predict construction time and over-predict iterative solution time, and the deviation with the data can be noticeable for small *P*. However, as *P* increases the model does a very good job at estimating the execution time, given the number of iterations the problem will consume. Further testing may reveal that no effects related to runtime conditions are being insufficiently captured by the model to explain the differences, especially in the construction time. In that case a deficiency in the model(s) must be sought to improve the predictions.

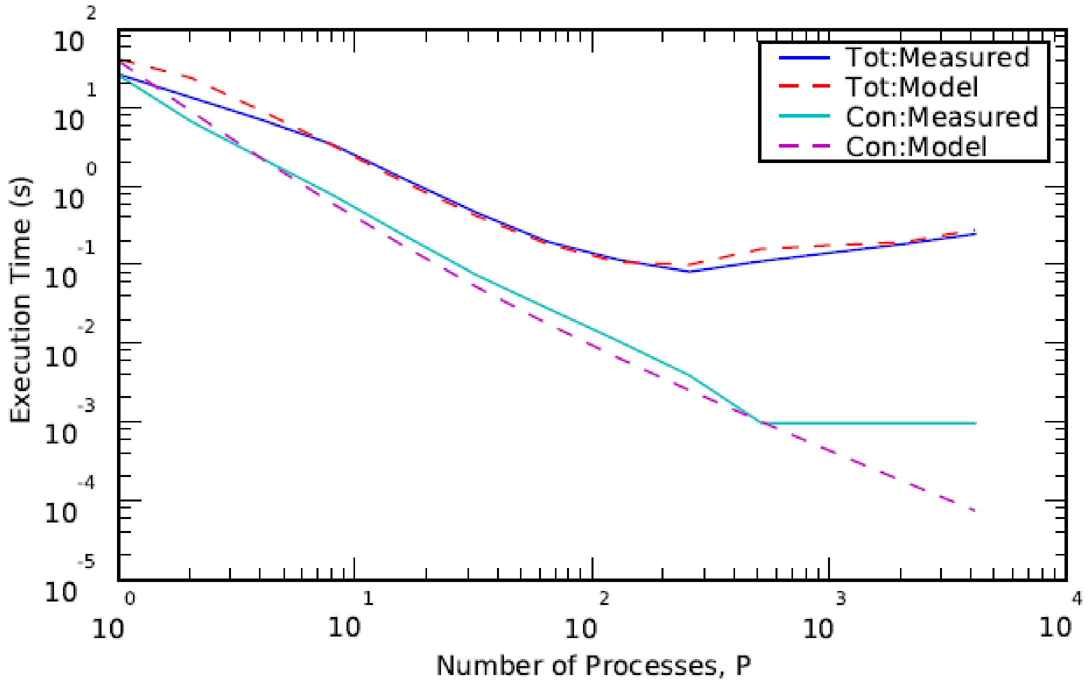


Fig. 3.3.10 Strong Scaling Exec. Time *vs*. *P* prediction, 16×16×16 base, S_8, c = 0.9, h = 1.0 cm.

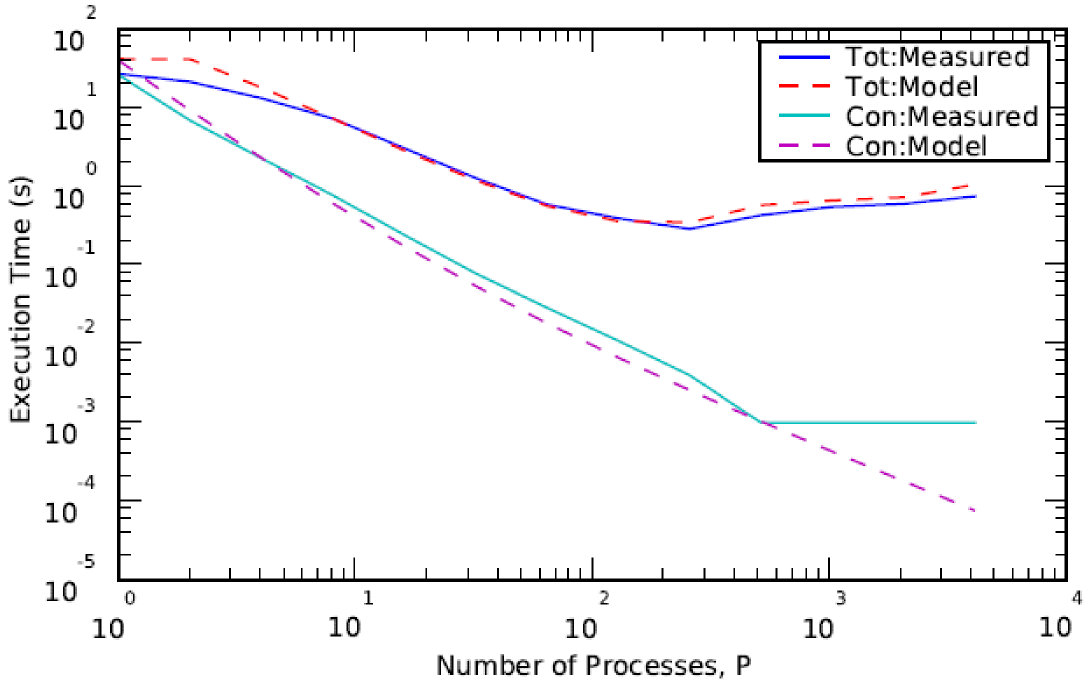


Fig. 3.3.11 Strong Scaling Exec. Time *vs*. *P* prediction, 16×16×16 base, S_8, c = 0.99, h = 1.0 cm.

3.4 Task 4 – Parallel implementation of the new algorithms

A computer code, the Parallel Integral Discrete Ordinates Transport Solver (PIDOTS), was written in the Fortran 90/95 standard to implement the ITMM and various parallel global solution techniques. It uses the Message Passing Interface (MPI) instruction set for parallelization. Multiple versions of PIDOTS were developed to test the various solution techniques. Specifically, in the PSD framework, [19] PBJ, PGS, and PGRMES (restarted) have been implemented in PIDOTS. Standard compiler optimization settings were utilized for improved performance, namely in the vectorization of matrix-vector operations. A flowchart describing the PIDOTS code is depicted in Fig. 3.4.1.

The primary concern of this project has been to develop a new neutron transport code and analyze its scalability to the massively parallel computing regime. Therefore computational test results related to the strong and weak scaling of the various parallel methods introduced in the previous chapter are presented first. Execution times will be broken down into ITMM operator construction time and global iterative solution time; the sum of the two times add up to the total execution time.

3.4.1 Supercomputing Platforms

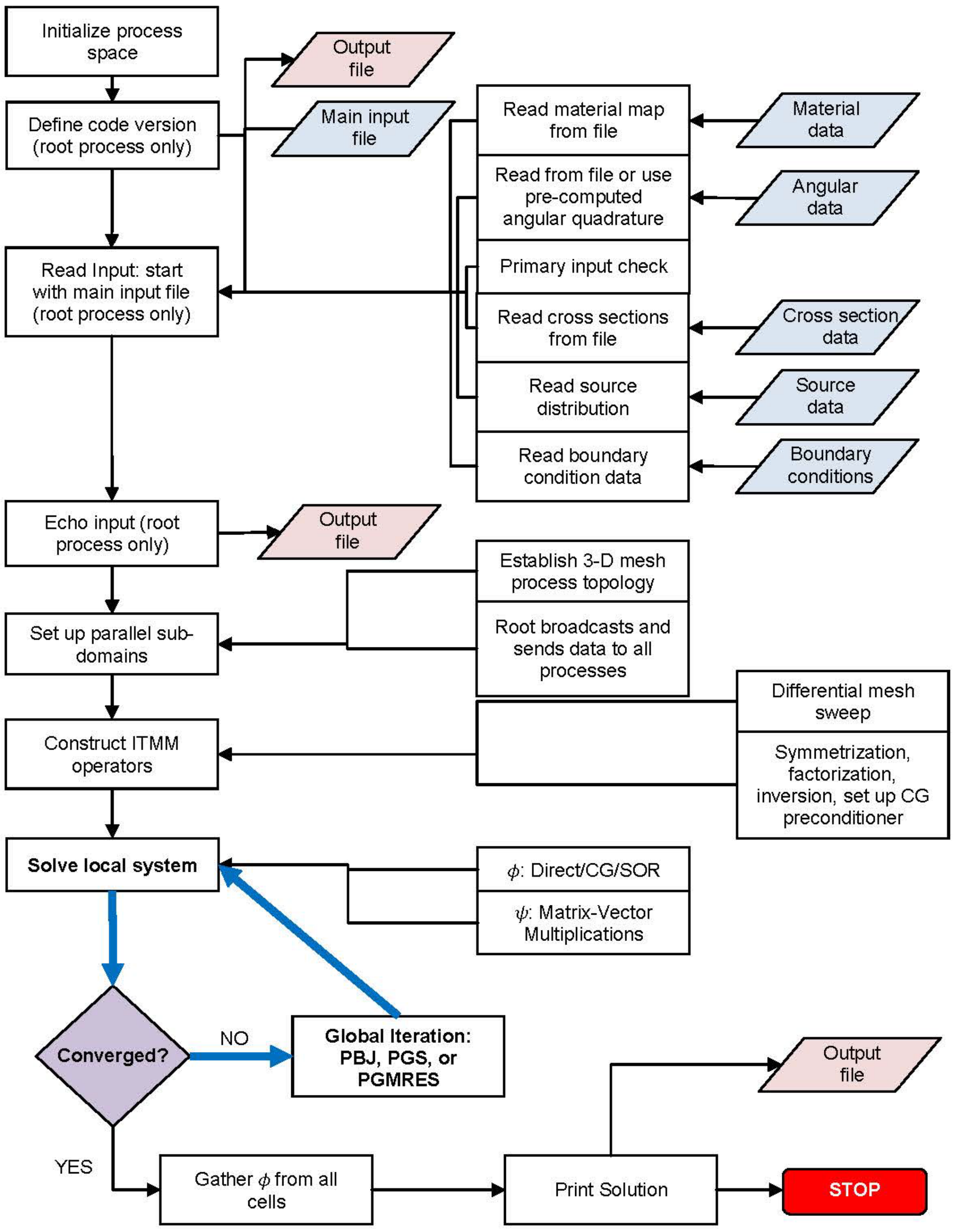
The timing results presented in this report were gathered from executions on three computer clusters of increasing size. All computing systems were distributed memory, MIMD [23] architectures. Among other features, they varied in the number of nodes, number of processors per node, processor speed, available memory, and interconnect network.

To establish terminology, a *node* refers to a blade server which is connected to other nodes via the network. All three supercomputers are *homogeneous*, meaning all nodes share the same design (processor speed, memory, processor design, etc.) Moreover, all nodes are multiprocessors. All three systems' nodes had two or more *sockets*, one processing chip per socket. Chips are *multicore*, i.e., they feature two or more central processing units (CPUs). Cores of the same socket share some levels of cache and also own at least a single small, fast level of cache for dedicated use.

Codes that employ MPI for parallelization are said to be decomposed into P independent *processes*. In our work, every processor, i.e., CPU, is assigned exactly one MPI process. However, to avoid confusion with other documents that may equate a processor with a socket, this report will use the term *processing element* (PE) to refer to a single CPU. Therefore, for any given problem P can equivalently refer to the number of deployed processes or the number of participating PEs, and effort will be made to distinguish the two when making a comment about the code (processes) *versus* a comment about the hardware (PEs).

Small tests on up to P = 256 were performed on the Yellowrail (YR) system at Los Alamos National Laboratory (LANL). YR is a distributed memory cluster of 139 nodes with 8 PEs and 16 gigabytes (GB) of memory per node; however, an administrative limit of 256 PEs per user-run is imposed. Nodes are organized into a single Connected Unit (CU), whereby a single, large switch connects all the nodes. The nodes are AMD Opteron blades. AMD Opteron machines have

NUMA designs. For nodes with two or more sockets, each chip has its own memory controller and a directly-connected physical memory set. *Intra*node communication is performed over a HyperTransport link.



Initialize process space
Output file
Define code version (root process only)
Main input file
Read material map from file
Material data
Read from file or use pre-computed angular quadrature
Angular data
Primary input check
Read Input: start with main input file (root process only)
Read cross sections from file
Cross section data
Read source distribution
Source data
Read boundary condition data
Boundary conditions
Echo input (root process only)
Output file
Establish 3-D mesh process topology
Set up parallel sub-domains
Root broadcasts and sends data to all processes
Differential mesh sweep
Construct ITMM operators
Symmetrization, factorization, inversion, set up CG preconditioner
Solve local system
ϕ: Direct/CG/SOR
ψ: Matrix-Vector Multiplications
Converged?
NO
Global Iteration: PBJ, PGS, or PGMRES
Output file
YES
Gather ϕ from all cells
Print Solution
STOP

Fig. 3.4.1 PIDOTS Flowchart.

With initial observations from YR, larger weak scaling tests were performed on LANL's Redtail (RT) system on up P = 1,024. RT is composed of 14 CUs of similar design to YR. Each CU has 131 AMD Opteron nodes. Each node has 8 PEs and 32 GB of memory. The maximum number of PEs per user-run is administratively limited to 1,024. The CUs are connected by a second stage InfiniBand interconnect of eight additional switches. Both YR and RT were predecessors to the better-known Roadrunner cluster, which features the Cell processors for enhanced, heterogeneous computing.

Due to limitations in the permitted number of PEs per user per job, highly massively parallel tests were not possible to perform on the YR and RT platforms. For access to thousands of PEs to test the code, the Cray XT5 JaguarPF (JPF) cluster at Oak Ridge National Laboratory (ORNL) was employed. JPF is composed of 18,688 nodes, each with 12 PEs and 16 GB of memory. The interconnection is static—a 3-D torus topology [24]. To handle the traffic of incoming/outgoing messages, each node is equipped with a SeaStar 2+ router. To fit the problems considered on the nodes, in terms of memory, runs were restricted to 8 PEs per node instead of using all 12 PEs.

Resource contention affects the timing results and can occur at intra- and inter-node levels. All systems ran programs in dedicated mode, restricting other users from issuing instructions to allocated nodes. However, within a single node, the most common resource contention occurs when multiple PEs compete both for space and access to shared cache and memory. At the network level, communications must either be passed among nodes, via routers, or through switches. During periods of heavy user load, message traffic on the network can affect performance. Due to these issues, all test cases were executed multiple times and average times are reported for a more accurate estimate of typical system performance, i.e. execution time.

3.4.2 The Base Model Problem

Consider a single sub-domain per PE—a PBJ-type problem. A generic model geometry has been created. Cell dimension h and material scattering ratio c are varied to investigate respectively the effects these parameters have on the ITMM's parallel solution strategies. For the purposes of this report, optical thickness refers to a measure of the probability of interaction between the neutron and the host medium. Fixing either h or σ_t and increasing the other is said to make the medium more optically thick. Moreover, the product of these two quantities results in the number of mean free paths, $mfp = h\sigma_t$ (unitless), that a neutron traverses across one edge of a cell. A large number of mfp corresponds to more interactions within a cell, and greater optical thickness. This definition of optical thickness is a departure from the more formal definition provided in Refs. [25] and [26], but it is consistent with terminology used in PBJ-related references and is valuable for the ensuing analysis.

The base model domain is a cube with side length L and four materials, as shown in Fig. 3.4.2, having no symmetries that may influence the iterative rate of convergence. The domain is discretized via a uniform mesh of cubic cells, and the number of cells in the base model is scaled to change the size of the ITMM operators. The diamond difference scheme is prone to negative fluxes when the spatial cells are too large given the total interaction cross section. To avoid the problem of negative flux solutions, each spatial cell was assigned a volumetric source q. The source strength was varied among the five regions, shown in Fig. 3.4.2. This base model is employed for strong and weak scaling studies with various parallel solution methods. For all tests, a single angular quadrature order is chosen and used for all variations of *c*, *h*, and *P*. Vacuum boundary conditions were always employed.

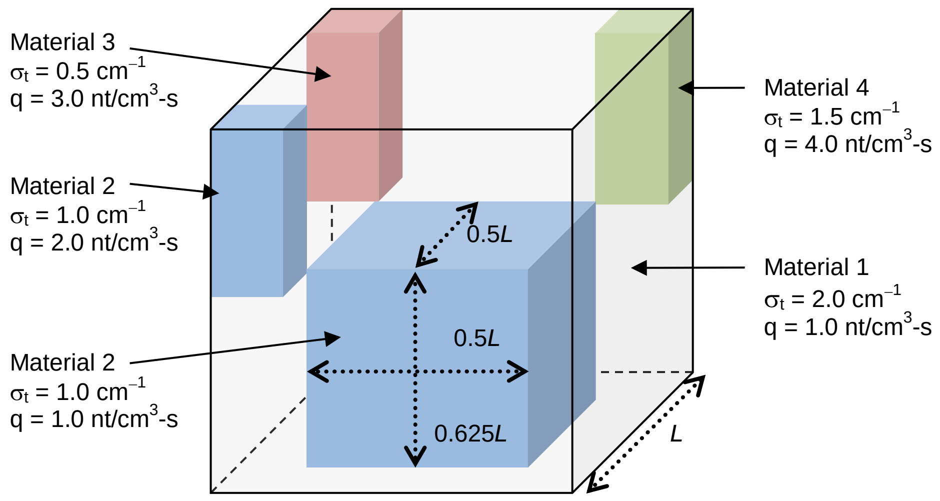


Figure 3.4.2 Test cases' base model of four materials and four source strengths.

In strong scaling studies the number of computational cells is fixed. However, one should note that studies of this kind in fact are not strong scaling per the most accurate definition; strong scaling studies represent a fixed number of unknowns or degrees of freedom which are distributed among the increasing number of participating processes. For strong scaling studies presented herein, even if the global domain size is fixed, the number of degrees of freedom increases with *P* as more boundary cells' angular fluxes are introduced as unknowns. Nevertheless, the strong scaling designation is maintained because the problem does reach a fixed amount of decomposition. Namely, the number of parallel processes is limited by the total number of cells, yielding 1×1×1 (-cell) sub-domains.

In weak scaling studies, the problem size grows with increasing *P*. For the purposes of this research, the number of unknowns per *P* was held constant while *P* was increased. Therefore, computational load related to construction and application of the ITMM operators does not change with *P*. In contrast, network load does increase, because the increase in *P* necessitates additional point-to-point messages between PEs and more effort to transmit global messages among all PEs. For weak scaling tests, additional processes are added to the 3-D virtual topology and the global domain is divided into sub-domains of equal size. The sub-domains are not periodic repetitions of the base model; instead, the base model domain is stretched in the dimension(s) that receives additional PEs and divided into cubic sub-domains.

Now consider cases where each PE is assigned multiple sub-domains—PGS-type problems. The base model itself is divided into equally sized cubic sub-domains, half red and half black arranged in a 3D-analogue of the checker-board pattern, all assigned to a single processor. As the problem is scaled to more PEs, the base model is distorted in the same manner described above: each additional PE gets a cubic region, which it divides into equally sized cubic sub-domains. Each PE must be assigned at least two sub-domains—one red, one black—to avoid processor idleness. However, for programming purposes, assigning each PE at least two sub-domain divisions in each dimension (minimum of eight sub-domains per PE) was easier because all PEs can use the same loops to determine which sub-domains are colored red and which are colored black. Therefore, all the PGS tests performed for this project employed a minimum of eight sub-domains per PE.

The base model has been employed for many test cases presented in the ensuing sections with varying number of cells in the base model and varying angular quadrature order. All angular quadrature sets S_N correspond to the LQ_N quadrature set, [25] a level (fully) symmetric quadrature with $N_t = N(N+2)/8$ discrete ordinates per octant. The large size of the ITMM operators and vectors make these methods limited by available memory. Therefore, problems with a large number of cells in the base model, e.g., 16×16×16, may be paired with the smaller S_4 and S_8 quadrature sets, and a smaller base model, e.g., 8×8×8, is paired with a higher S_{12} or S_{16} quadrature.

3.4.3 PBJ Scaling Results

Test cases with the PBJ global solution method have been developed to analyze strong and weak scalability. Moreover, throughout this section, PBJ results will frequently be used as a reference point against which all global solution methods with the ITMM will be compared.

3.4.3.1 Strong Scaling

Strong scaling studies have previously been performed for a different model on the YR system up to P = 256. [27] The results of that study indicated that the PBJ model performed well against serial SI (as implemented in PIDOTS) when the sub-domains were small—four cells—and when the optical thickness and scattering ratio were high. Strong scaling efficiencies relative to the corresponding serial SI executions were greater than 100%. The conclusion that PBJ is competitive with SI for high scattering and thick cells is not expected to change with a different model. However, gaining insight into the optimal size of the sub-domains for a larger problem is valuable.

A larger strong scaling study was performed on the JPF system with up to P = 4,096. The base model of Fig. 3.4.2 was used, where the entire system was a cube with 16 cells in each direction. The 16×16×16 model was parallelized by factors of two—in the cyclic order of doubling the process topology in the z-dimension first, then y, then x ($z \rightarrow y \rightarrow x$)—until 4,096 PEs were employed, each owning a single 1×1×1 cell. The cell dimensions were varied as h = 0.1 cm, 1.0 cm, and 10.0 cm. The scattering ratio was varied as c = 0.9 and 0.99.

In Fig. 3.4.3 the iteration counts for varying h and c = 0.9 are presented. Although the iteration counts between the methods do not compare equally in terms of number of operations, they do reveal trends with varying problem parameters. For optically thin systems like the h = 0.1 cm case, the number of PBJ iterations quickly eclipses SI. Problems with larger optical thickness involve more interactions before particles can escape, and the SI iteration count increases. Starting with a zero-flux initial guess, the n^{th} iterate of the SI scheme corresponds to the flux of neutrons that has undergone n collisions and has not been destroyed via absorption or leakage. [25] Hence the lower the probability of a neutron to leak or be absorbed, the larger the n^{th} iterate and thus the slower the convergence of SI.

In contrast, convergence of the PBJ method involves resolving the coupling among sub-domains via incoming/outgoing interface angular flux, and coupling between sub-domains and the global boundary conditions, i.e., the impact of angular fluxes exterior to the domain. Iterations are not physically related to the number of interactions a neutron undergoes. Rather they are tied to how much a change in the flux distribution in one sub-domain impacts the flux distribution in all others. PBJ sub-domains become increasingly *decoupled* with increasing *h*—their effects are more localized and convergence occurs faster—so long as *P* is large. When *P* is small, sub-domains are more tightly coupled to the fixed vacuum boundary conditions, and optically thin sub-domains allow for more rapid conveyance of those effects. Therefore, up to *P* = 32, the *h* = 0.1 cm case uses the fewest PBJ iterations. For P > 128, the number of iterations seems to be driven by the transport effects among sub-domains more than the effects from the boundary conditions. When *h* is large enough, the PBJ number of iterations approaches the SI count with increasing *P*.

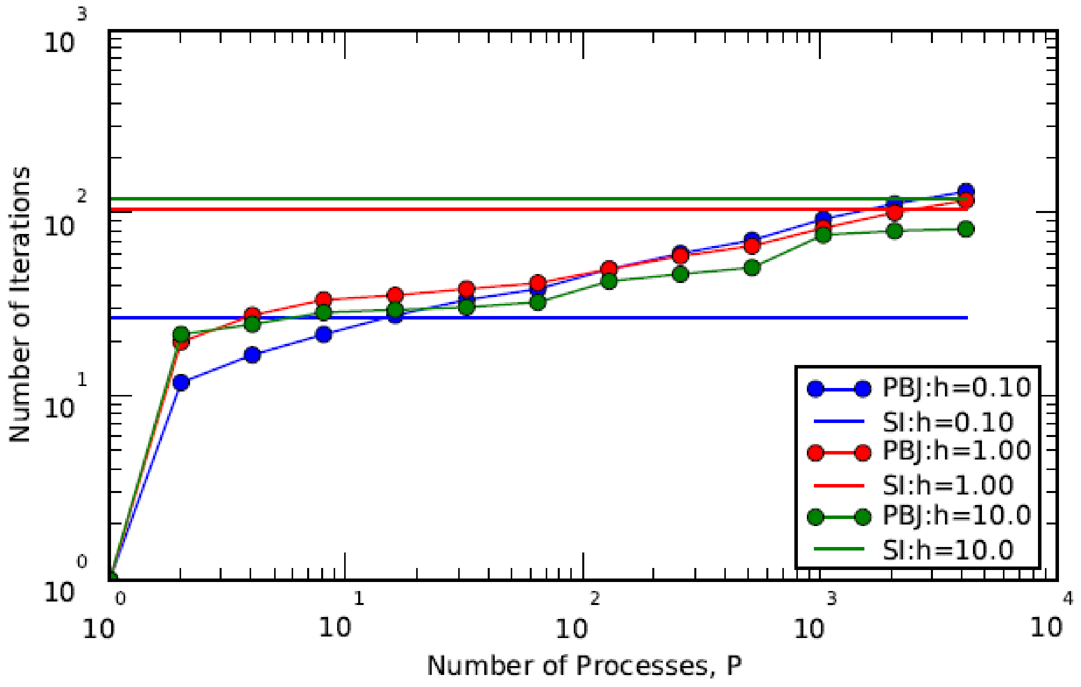


Fig. 3.4.3 PBJ strong scaling study, Iterations *vs. P*, 16×16×16 model, *c*=0.9, varying *h*.

The iteration counts for the *c* = 0.99 case are shown in Fig. 3.4.4. Similar trends as for the *c* = 0.9 case are apparent, but the steeper PBJ curves indicate that diminishing the absorption mechanism results in more tightly coupled sub-domains. Likewise, more source iterations are needed for convergence as neutrons undergo more interactions before removal from the system. Interestingly, the increase in number of iterations is much more prominent for the two thicker cases, than the thin, *h* = 0.1 cm case, possibly indicating that boundary conditions are still playing a prominent role in the convergence of the interface-flux iterates. A promising result for future tests is the flattening of the *h* = 10.0 cm curve compared to the optically thinner cases. Because the scalability of the parallel algorithm depends on the iteration counts, flatter growth is expected to yield better parallel performance.

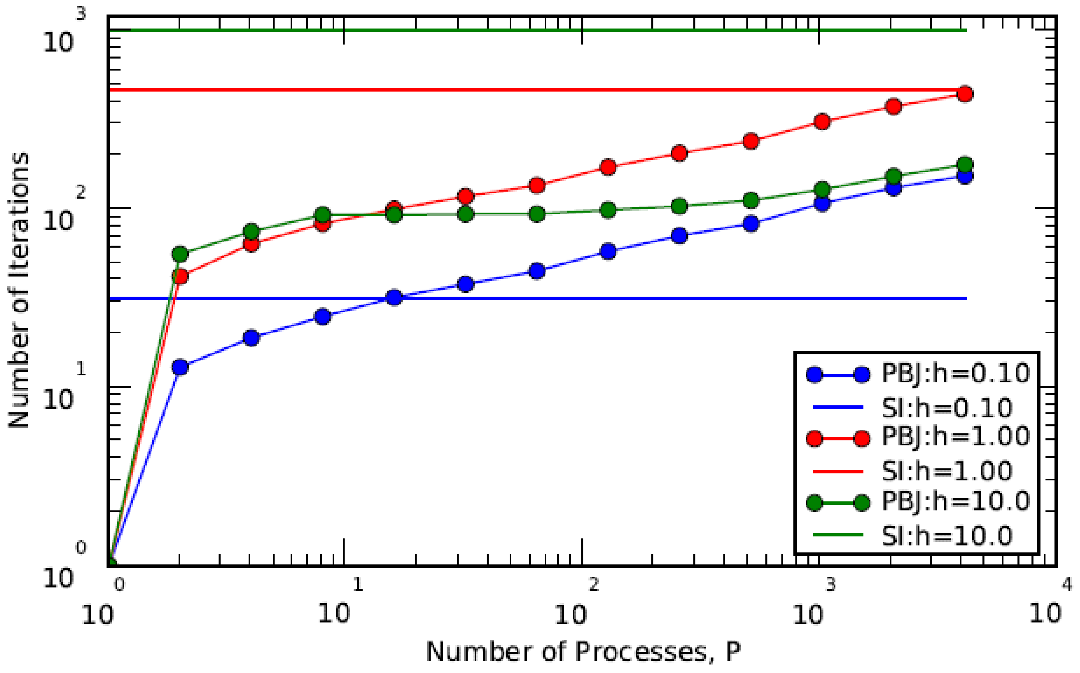


Fig. 3.4.4. PBJ strong scaling study, Iterations *vs. P*, 16×16×16 model, *c*=0.99, varying *h*.

The iterations of the two methods require a different set and number of operations. Therefore to have a complete discussion of the performance, execution time must be observed. In Fig. 3.4.5 the *c* = 0.9 strong scaling cases immediately demonstrate the cost operator construction time imposes on the PBJ method. Because varying *c* and *h* does not affect the operators' sizes and construction times, a single curve, averaged over all cases' runs, is presented. For small *P*, hence larger operators, the ITMM construction time is greater than the serial SI execution time, thus making it impossible for PBJ to do better regardless of the number of iterations. However, as *P* increases, construction time expectedly decreases super-linearly per the arguments made earlier. By *P* = 16, the iterative solution time dominates the PBJ total time, but the decreasing size of the operators continues to drive the total time downward. Unfortunately, the increasing PBJ iterations create a competing effect, tending to increase the execution time. At *P* = 128—32 cells per sub-domain—the effect of the iterations overcomes reduced operator size and the execution time increases. Interestingly, for this larger problem the optimal sub-domain sizes were noticeably larger than previous YR results in Reference [27].

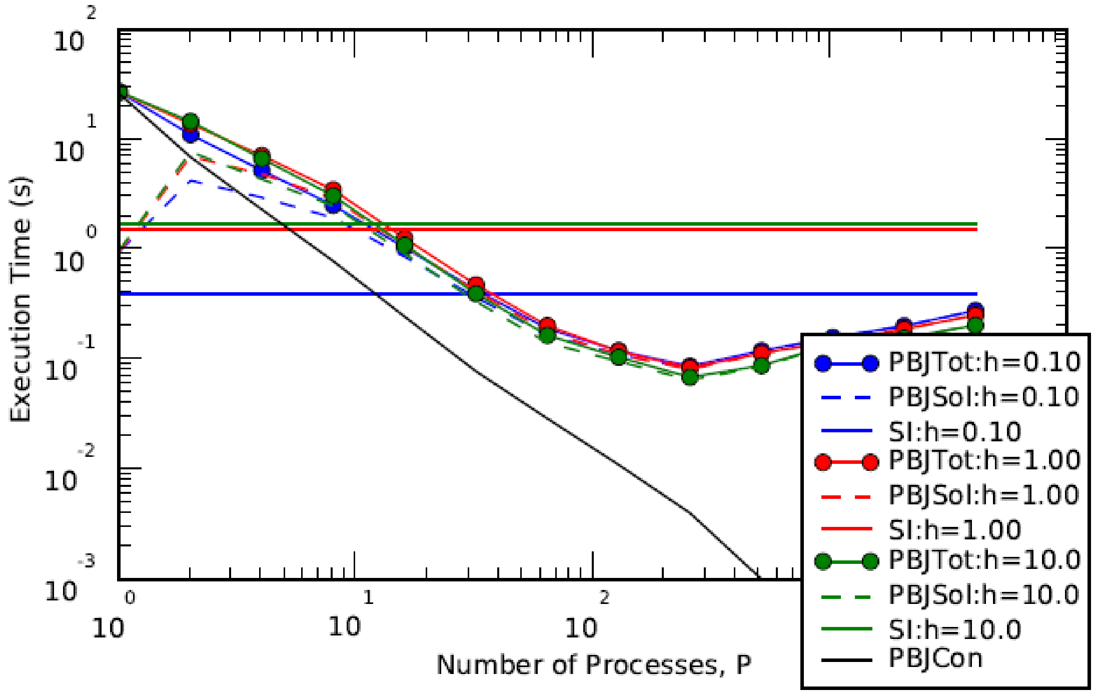


Fig. 3.4.5 PBJ strong scaling study, Execution Time *vs. P*, 16×16×16 model, *c*=0.9, varying *h*.

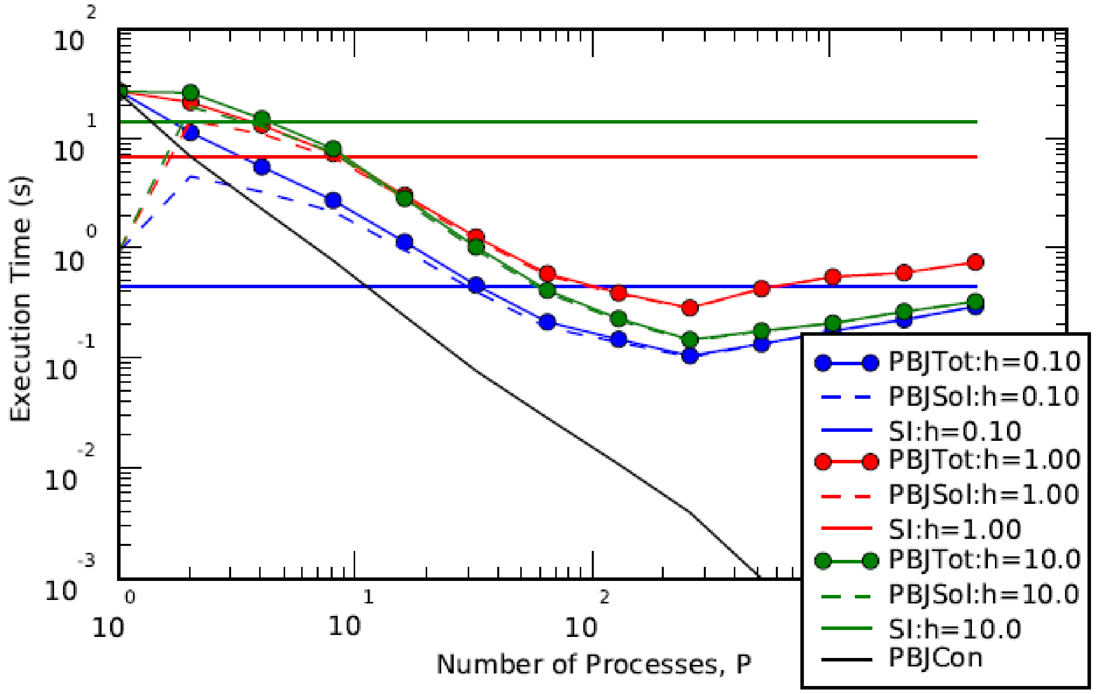


Fig. 3.4.6 PBJ strong scaling study, Execution Time vs. P, 16×16×16 model, c=0.99, varying h.

Parallel performance results are quantified via speedup S_P and efficiency E_P,

$$S_P \equiv T_1/T_P \quad and \quad E_P \equiv S_P/P, \qquad (3.4.1)$$

using execution times with a single process, T_1 (from SI in these cases), and with P processes, T_P. In most cases, with enough processors the PBJ algorithm could complete its execution faster than the serial SI cases. Generally, the PBJ plots in Figs. 3.4.5 and 3.4.6 show that optically thick and highly scattering problems have the best comparative features with their serial SI counterparts. In Figs 3.4.7 and 3.4.8 it is evident that only the most optically thick cases come close to ideal linear speedup. Although such a speedup is not achieved for this model, all runs demonstrate steady gains in speedup up to *P* = 128. When execution time begins to re-increase for the PBJ cases, the speedup begins to decrease.

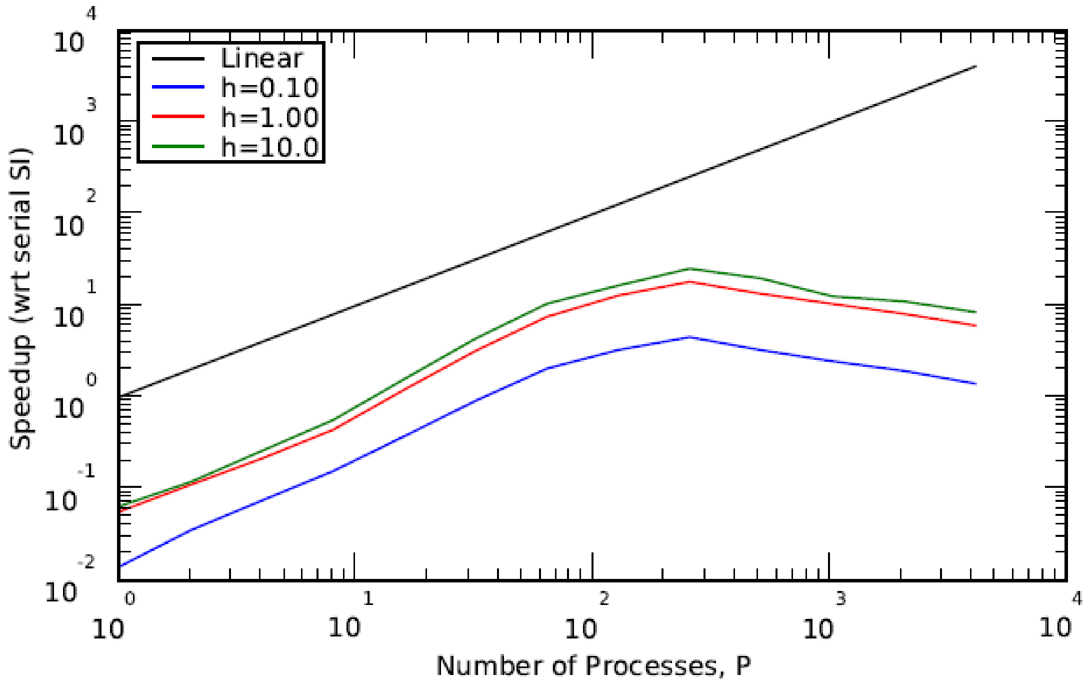


Fig. 3.4.7 PBJ speedup relative to respective SI cases, *c*=0.9, varying *h*.

Figure 3.4.9 clearly shows the weakness of the PBJ algorithm for fixed size problems. When *c* = 0.9, the optically thick cases have parallel efficiencies near 10%, which is small, and drop dramatically after *P* = 128 when the execution time begins to increase due to the increased iteration count. The optically thin case is much worse, only approaching 2% efficiency. For the highly scattering cases with *c* = 0.99, as long as the optical thickness is high, parallel efficiencies greater than 50% can be attained with up to *P* = 128, as shown in Fig. 3.4.10. However, decreasing the thickness leads to the same poor relative performance.

The above results indicate that for problems of fixed size, i.e. number of cells, each case has an optimal number of processors that balances the competing effects of super-linear decrease in ITMM operator sizes and increase in the PBJ iteration count. Approaching the balance is accompanied by steady speedup in the algorithm. However, after this balance has passed the efficiency of the method decreases significantly.

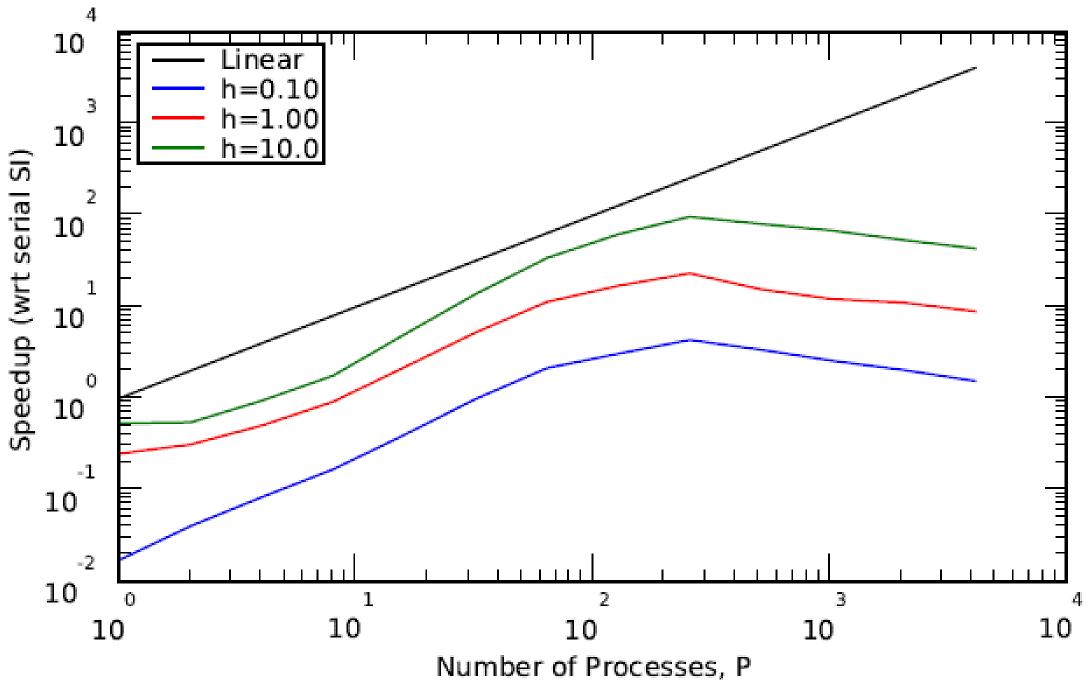


Fig. 3.4.8 PBJ speedup relative to respective SI cases, *c*=0.99, varying *h*.

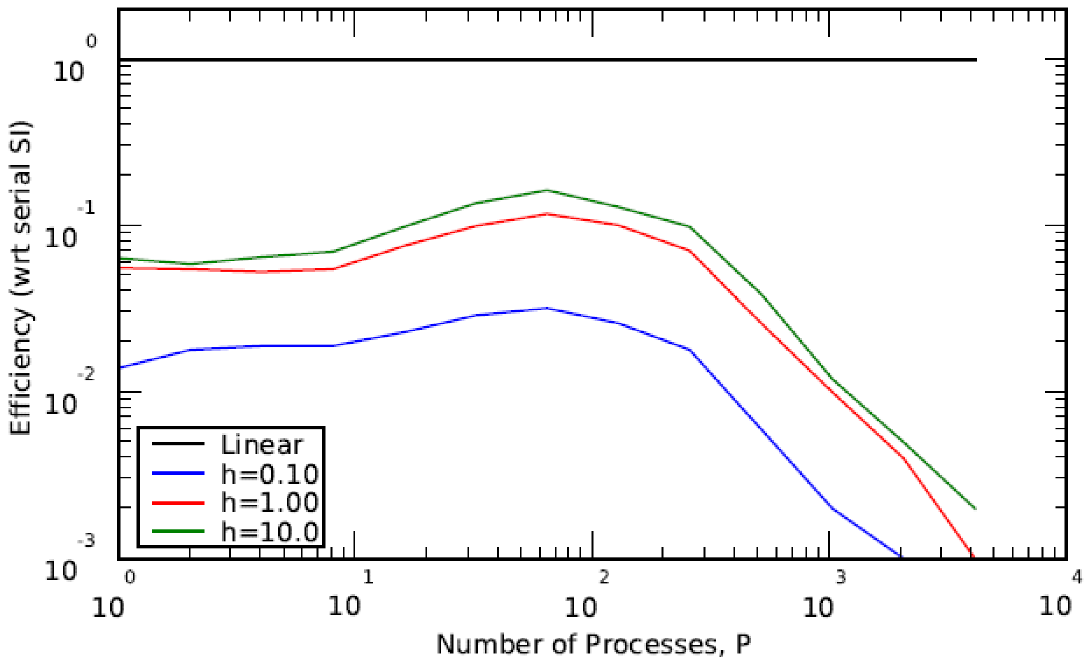


Fig. 3.4.9 PBJ efficiency relative to respective SI cases, *c*=0.9, varying *h*.

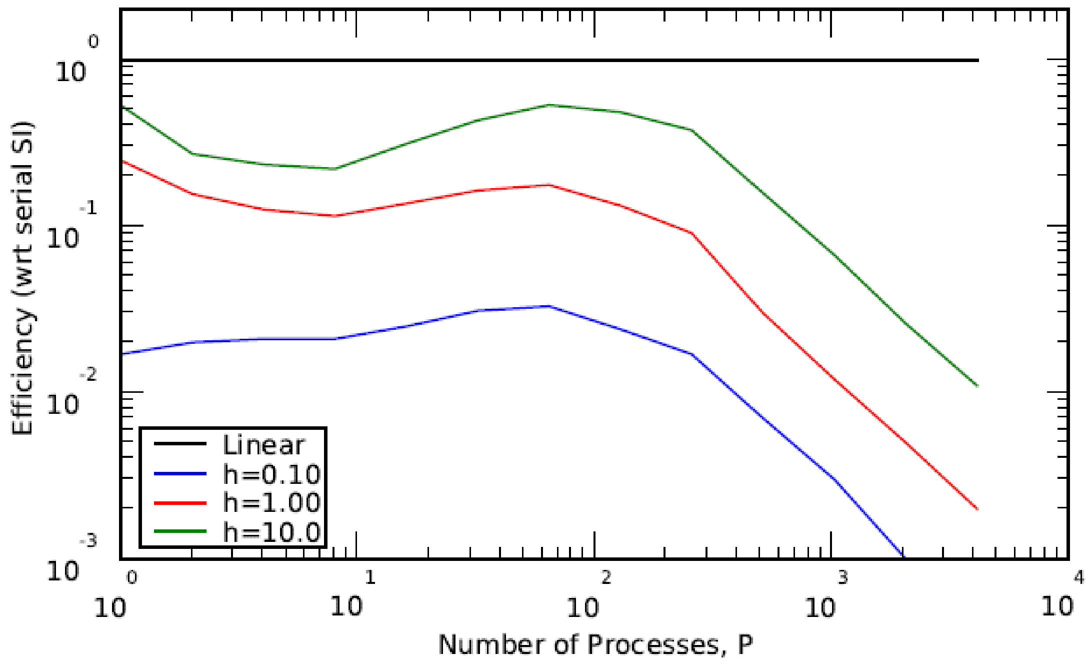


Fig. 3.4.10 PBJ efficiency relative to respective SI cases, *c*=0.99, varying *h*.

3.4.3.2 Weak Scaling

Strong scaling studies, while beneficial to understanding parallel performance, are not the only way to gauge a new method's scalability compared to current state-of-the-art. In weak scaling studies the problem size grows with the number of PEs. Weak scaling is designed such that the size of the sub-domain assigned to each PE does not change; therefore, operator construction time is approximately a fixed cost as *P* is increased. Increases in execution time will be driven by increased communication and increasing number of global iterations. If the number of iterations grows slowly, the scalability of the PBJ algorithm will make it an attractive choice in massively parallel regimes.

Initial weak scaling tests were performed on the YR system with up to *P* = 256. The 16×16×16 (4,096 cells) base model was used with the S_4 angular quadrature set; with increasing *P* each additional PE is assigned an additional 16×16×16-cell sub-domain. As previously stated, the base model is stretched in the direction of additional PEs in the virtual topology. Additional PEs are added in one dimension at a time, cycling through the dimensions $z \rightarrow y \rightarrow x$ to maintain a low global domain surface-to-volume ratio. The scattering ratio and cell dimensions were varied again as *c* = 0.9 and 0.99 and *h* = 0.1 cm, 1.0 cm, and 10.0 cm, respectively. Neither serial nor parallel SI runs were performed for this study, reserving a comparison for the next section between SI with KBA parallelization and the PGS method, primarily because PGS performs better than PBJ hence is the more likely approach for production level implementation.

When the scattering ratio is 0.9, the number of iterations, shown in Fig. 3.4.11, grows very slowly for the optically thicker—*h* = 1.0 and 10.0 cm—cases because sub-domain effects are highly localized. The large optical thickness means that neutrons are more likely to spend their entire lifetimes within the sub-domain where they are born or in a nearby sub-domain. When *P* is small, leakage out of the system dominates the problem and the *h* = 0.1 cm case uses the fewest iterations. Yet this iteration count grows much more quickly and shows very little change in slope from *P* = 2 to *P* = 256.

Figure 3.4.12 shows that increasing the scattering ratio to 0.99 shifts all curves upward relative to the *c* = 0.9 cases. The optically thinnest case has a steeper slope for *P* > 2 than the others and a steeper slope than its *c* = 0.9 counterpart, but for *P* in the range between 1 and ~64 the *h* = 0.1 cm case uses fewer iterations than the *h* = 1.0 cm and 10.0 cm cases. The latter behavior seems to be caused by the fact that the increase in *c* shifts the optically thicker curves more than the optically thinnest case. As the problem becomes less influenced by the boundary conditions with increasing *P*, the dominating effect is how tightly sub-domains are coupled via interfacial exchange of neutrons comprising interface angular fluxes. The high scattering ratio results in less localized transport effects, but the optically thin case seems to benefit from small physical dimensions and increased effect from boundary conditions—leakage—when *P* is small. Nevertheless, large optical thickness also results in more neutron absorption within sub-domains, thus decoupling them, and the iteration counts grow slowly with increasing *P*.

The iteration count curves for *h* = 1.0 and 10.0 cm cases have the desired trait of slow growth that leads to slowly growing execution time. In Fig. 3.4.13 the execution time ("Tot") for the *c* = 0.9 cases is decomposed into the iterative solution ("Sol") time and the operator construction ("Con") time. The "Sol" curves follow the same trend as the iteration count curves in Fig. 3.4.11 as expected. The serial PBJ case is solved with a single iteration (in the case of all vacuum boundary conditions) and therefore represents an absence of iterations and communication. As in the case of strong scaling studies, varying *c* and *h* does not affect construction time, so the single plot is an average from all runs. This curve is nearly flat because all PEs have the same size sub-domain as P increases. However, it is clear that operator construction time dominates the total execution. This result poses the immediate challenge of reducing ITMM operator construction time while maintaining the slow growth in iteration count.

In Fig. 3.4.14, the larger number of iterations due to the increased scattering ratio, *c* = 0.99, leads to the iterative solution time playing a larger role in the total execution time. In fact, dividing the "Sol" data points by the corresponding number of iterations shows that the PBJ algorithm is using ~0.6–0.7 s per iteration for this model. Moreover this time tends to increase with *P*, likely due to the need to communicate through the YR switch when *P* > 8 and increased contention as *P* increases and more messages are sent. However, the matrix-vector operations performed per iteration are the dominating factor in per iteration time. Therefore, the second challenge becomes reducing the matrix-vector multiplication times—namely, in the form of smaller operators—without greatly increasing the number of iterations or at least not changing the slopes of those curves for high *P*. The increasing number of iterations and necessary communication tends to increase the execution time versus *P*. These factors tend to harm parallel scalability, but a measure is helpful to know by how much the algorithm is affected relative to some reference. The definitions given in Eq. (3.4.1) are no longer suitable, because the size of the problem is not fixed, requiring less work per PE as *P* is increased. Rather, now the number of operations per iteration is fixed for all *P*. Running larger problems in serial is not possible due to insufficient memory, but the *P* = 1 execution time does represent optimal performance for the algorithm because it lacks iterations and communication. Reference [28] provides a definition for weak scaling efficiency:

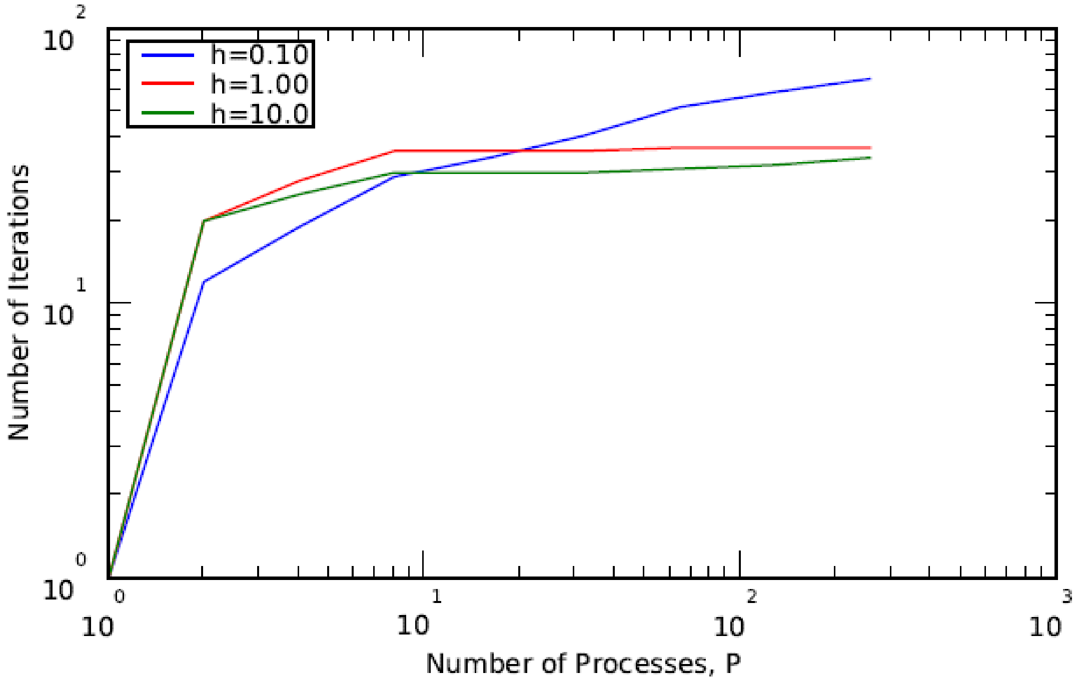


Fig. 3.4.11 YR PBJ weak scaling study, Iterations *vs. P*, 16×16×16 model, *c*=0.9, varying *h*.

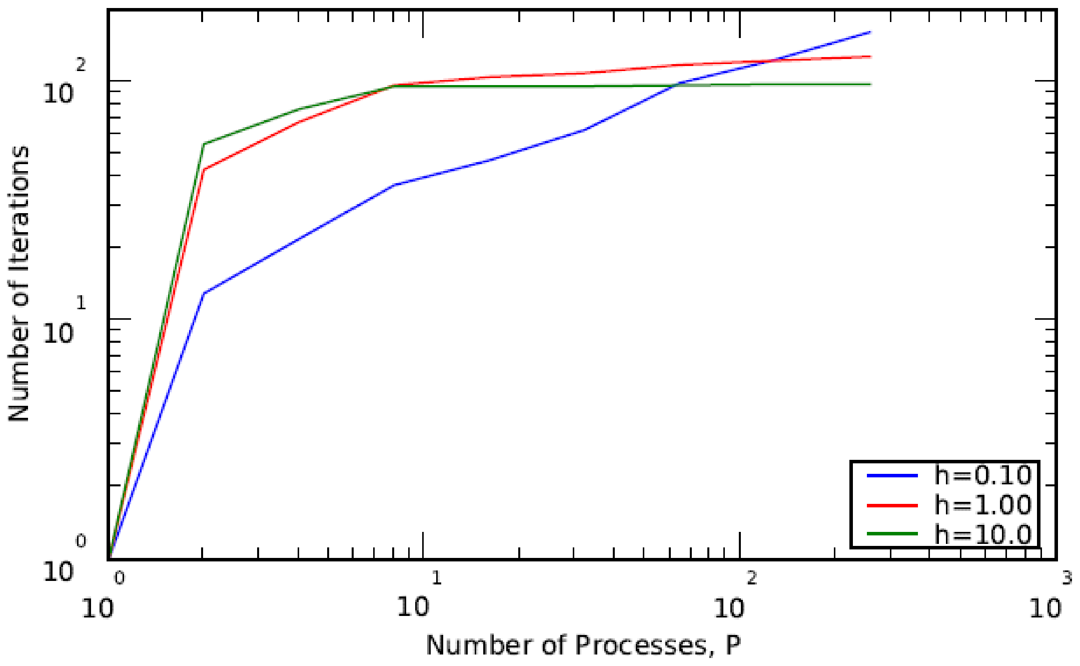


Fig. 3.4.12 YR PBJ weak scaling study, Iterations *vs. P*, 16×16×16 model, *c*=0.99, varying *h*.

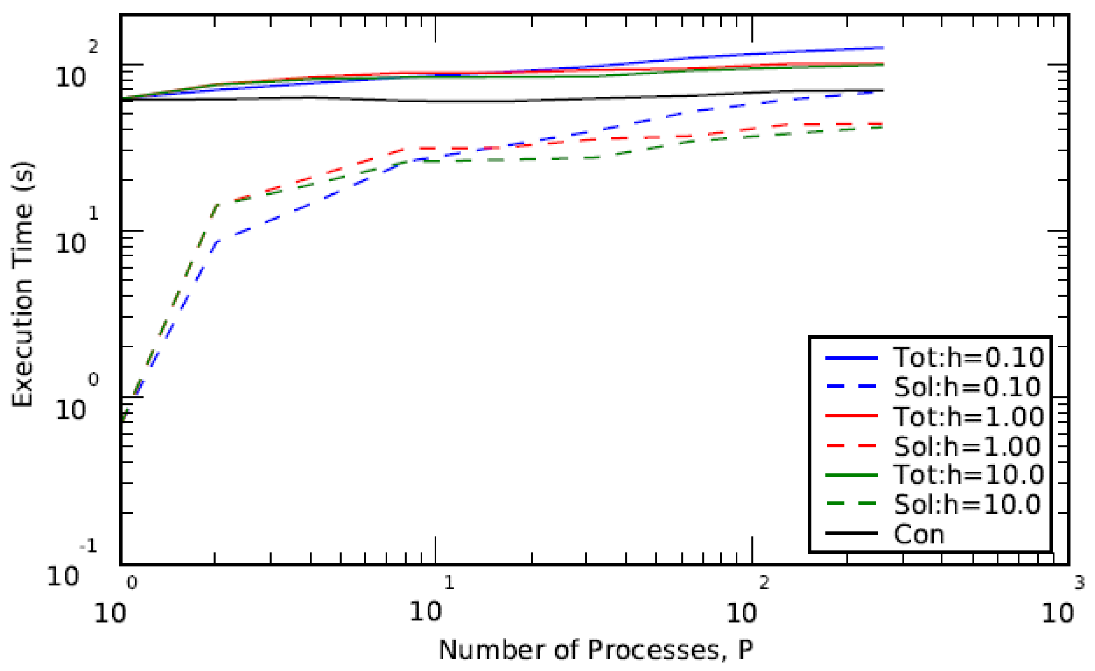


Fig. 3.4.13 YR PBJ weak scaling, Execution Time *vs. P*, 16×16×16 model, *c*=0.9, varying *h*.

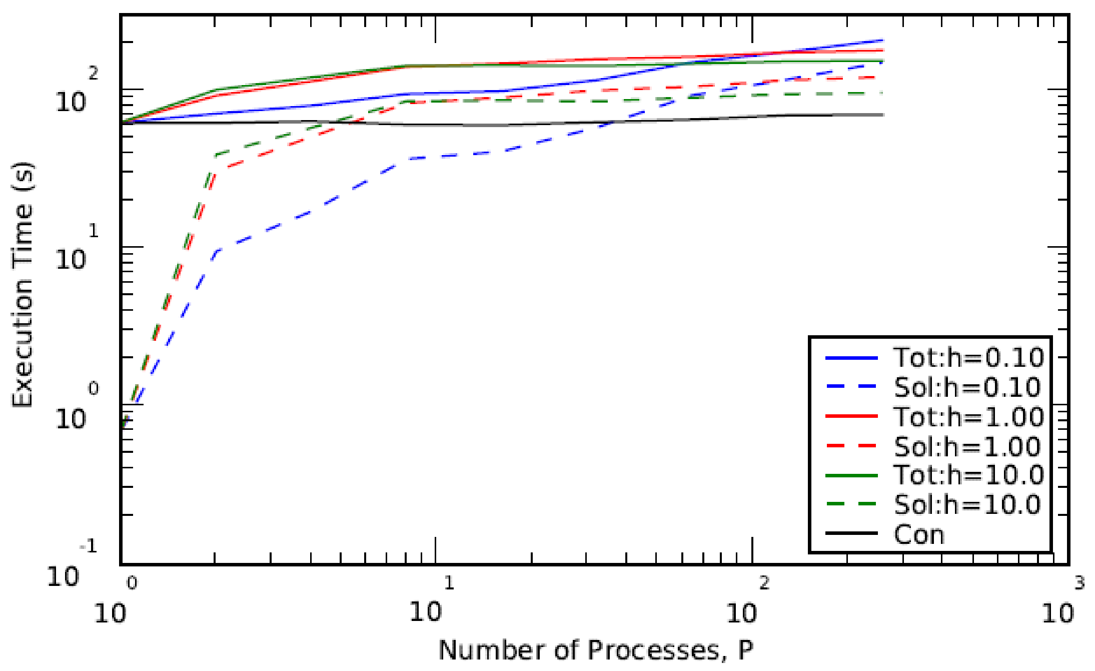


Fig. 3.4.14 YR PBJ weak scaling, Execution Time *vs. P*, 16×16×16 model, *c*=0.99, varying *h*.

$$\epsilon_P \equiv \frac{\tau_{ref}}{\tau_P}\left(\frac{DOF_P}{DOF_{ref}}\right), \ \ \tau = T \times P. \tag{3.4.2}$$

DOF is the number of degrees of freedom (number of unknowns) for parallel runs on *P* PEs; the "*ref*" problem run is typically a problem run with *P* = 1. The wall-clock time is denoted with *T*. For the purposes of all the weak scaling studies performed for this project, the number of unknowns scales linearly with *P*. Employing the *P* = 1 runs as the reference case, Eq. (3.4.2) simplifies, and weak scaling efficiency can be defined as

$$\epsilon_P \equiv \frac{T_1}{T_P}. \tag{3.4.3}$$

This formula appears the same as strong scaling speedup. However, their respective implementations are very different.

Weak scaling efficiency for varying *h* relative to respective single sub-domain PBJ runs is plotted versus *P* for the *c* = 0.9 cases in Fig. 3.4.15. If all parallel runs used a single iteration and no communication, the efficiency would be unity independent of *P*. However, the efficiency drops with increasing *P* as the problem requires more iterations to converge and communication costs are incurred. The two thicker cases have flatter iteration curves, and consequently their parallel efficiency according to Eq. (3.4.3) drops slowly reaching about 60% for *P* = 256. Looking at the *c* = 0.99 efficiencies in Fig. 3.4.16, the thick cases still have flat efficiency curves for high *P*, but they now settle at approximately 40% relative to the serial cases. In both figures the *h* = 0.1 cm case's efficiency slope is more steeply negative and does not change much with increasing *P*. Viewing the data in this manner highlights the importance of accelerating iterative convergence such that fewer iterations are necessary and parallel efficiency can be improved.

This weak scaling study was repeated on the JPF system with up to *P* = 1,024. Knowing that the 16×16×16 base model problem makes the ITMM operator construction time a more dominating component of total execution, the base model was adjusted to 8×8×8 sub-domains. The S_8 quadrature set was selected to refine the angular mesh and to utilize more of the available memory. Varying *c* and *h* in the same manner as before provides insight into how the different architecture and model size (number of cells and angles) affects weak scaling. However, it is expected that the general conclusions will remain the same. That is, the efficiency of a given case may be altered due to the new problem parameters and computing platform, but relative to the other cases of *c* and *h*, the efficiency will exhibit the same behavior observed with the PBJ method.

The iteration count curves are plotted in Fig. 3.4.17 for all six cases of varying *c* and *h*—same values as the study performed on the YR system. The trends are largely the same as those in Figs. 3.4.11 and 3.4.12. Most importantly, these include the flattening of the curves for optically thick cases while the thinnest case has steep growth in iterations up to *P* = 1,024 and that increasing the scattering ratio increases the number of iterations by reducing neutron removal by absorption, thus tightening the coupling among sub-domains. The larger *P* compared to the YR study does not yield a flattening of the optically thin cases' iteration curves. Such a result would have been desirable to suggest that those cases may perform in more massively parallel regimes—i.e., thousands of PEs. The most important difference compared to the study on the

YR system involves the c = 0.99, h = 1.0 cm case. It can be observed in Fig. 3.4.12 that this curve has a slight upward slope. That slope is much more pronounced for this study. The smaller 8×8×8 sub-domains are the likely cause. Because the sub-domains are physically smaller in this model, they demonstrate tighter sub-domain coupling just as thinner cells produce a similar effect. Nevertheless, this problem does not appear for the c = 0.9, h = 1.0 cm case, where the greater absorption counters the effect from reduced sub-domain size.

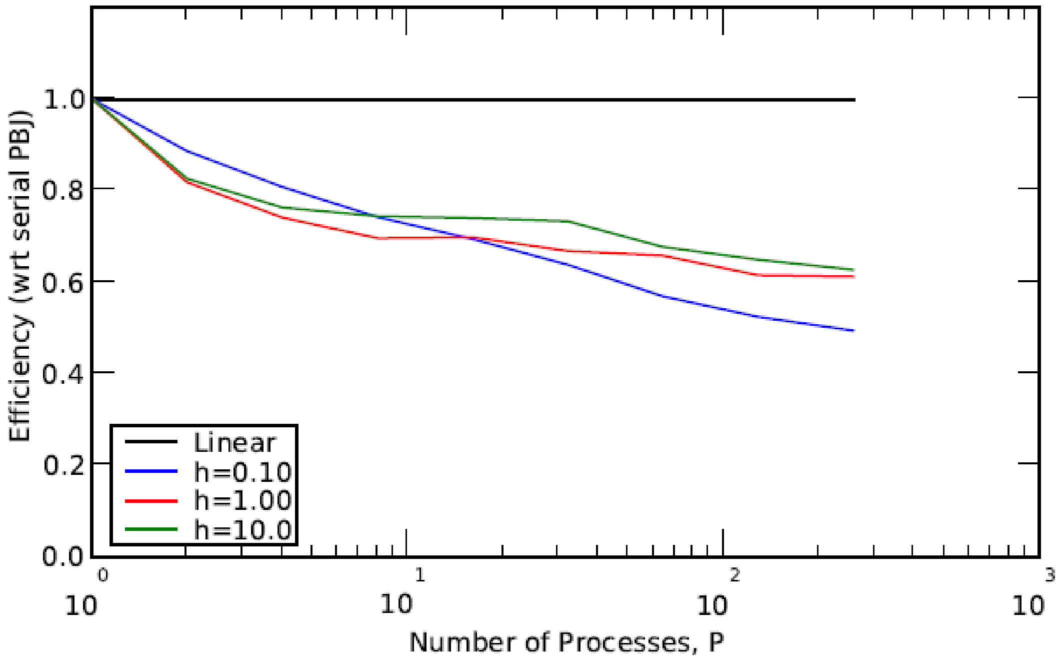


Fig. 3.4.15 YR PBJ weak scaling efficiency relative to serial PBJ runs, c=0.9, varying h.

Figure 3.4.18 shows the execution time for the six cases. As in the YR study, the iterative solution curves follow the corresponding iteration count curves. Even though the angular quadrature order was increased, the large drop in number of cells per sub-domain—4,096 to 512—resulted in a very significant decrease in the ITMM operator construction time. The reduced relative contribution to total execution time means the solution time dominates, and construction time is insignificant for the longest running calculations—c = 0.99 cases, P > 256. For P = 1 cases, the construction time consumes nearly the entirety of the execution time because the time for a single iteration is less than 0.2 s. For larger P, this time increases to ~0.3 s. The slowest growing execution times are expected to have the best scalability for P > 1,024, and these cases remain the optically thick ones due to the decoupling of sub-domains.

Given that the construction time dominates the P = 1 calculations but becomes a small contributor at high P, reduced relative efficiency is an expected consequence. The parallel efficiency curves relative to the serial PBJ runs are shown in Fig. 3.4.19. Very steep drops in relative efficiency are followed by much flatter trends for thicker cases. At ~16%, the c = 0.9, h = 10.0 cm case displays the best efficiency. The thinner cases' efficiencies continue to drop with P. Nevertheless, these results are acceptable. The curves display the same characteristics as before, but the construction time is much smaller. The method is by design supposed to shift computational burden to the independent task of building ITMM operators while seeking a slow growth in the number of iterations. If the construction time is small compared to the iterative solution time, this will skew the efficiency relative to the serial case that converges in a single iteration.

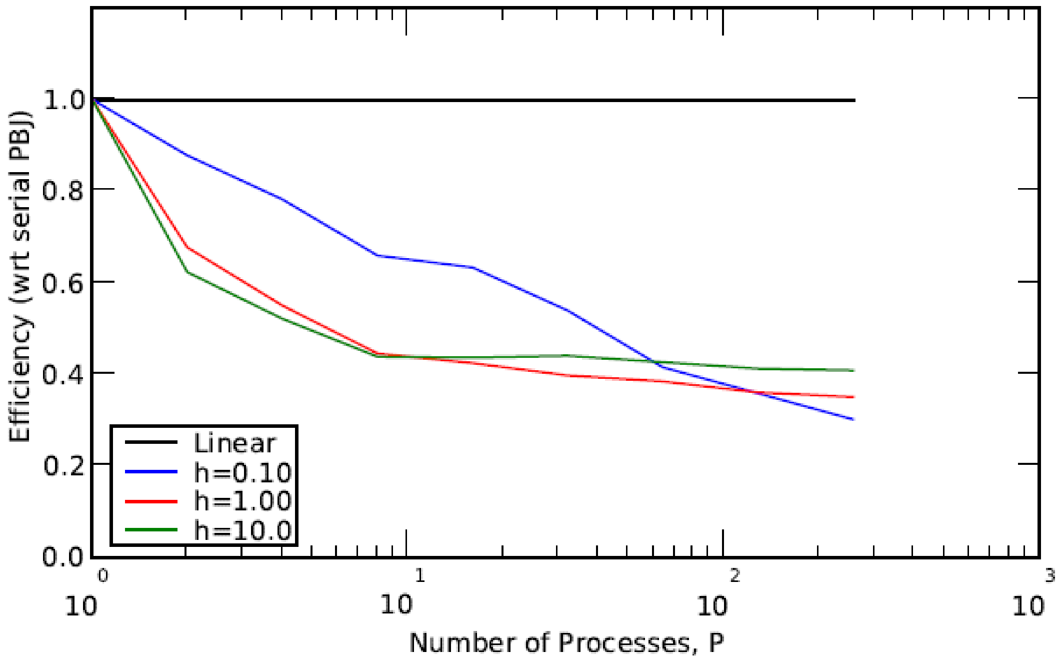


Fig. 3.4.16 YR PBJ weak scaling efficiency relative to serial PBJ runs, *c*=0.99, varying *h*.

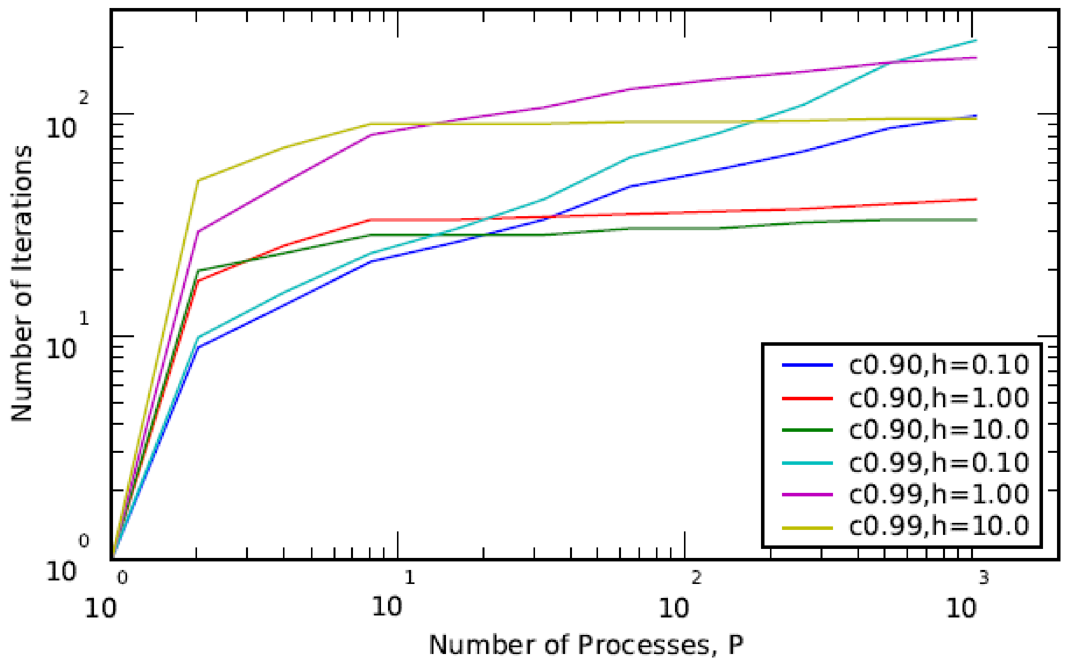


Fig. 3.4.17 JPF PBJ weak scaling study, Iterations *vs. P*, 8×8×8 model, varying *c*, *h*.

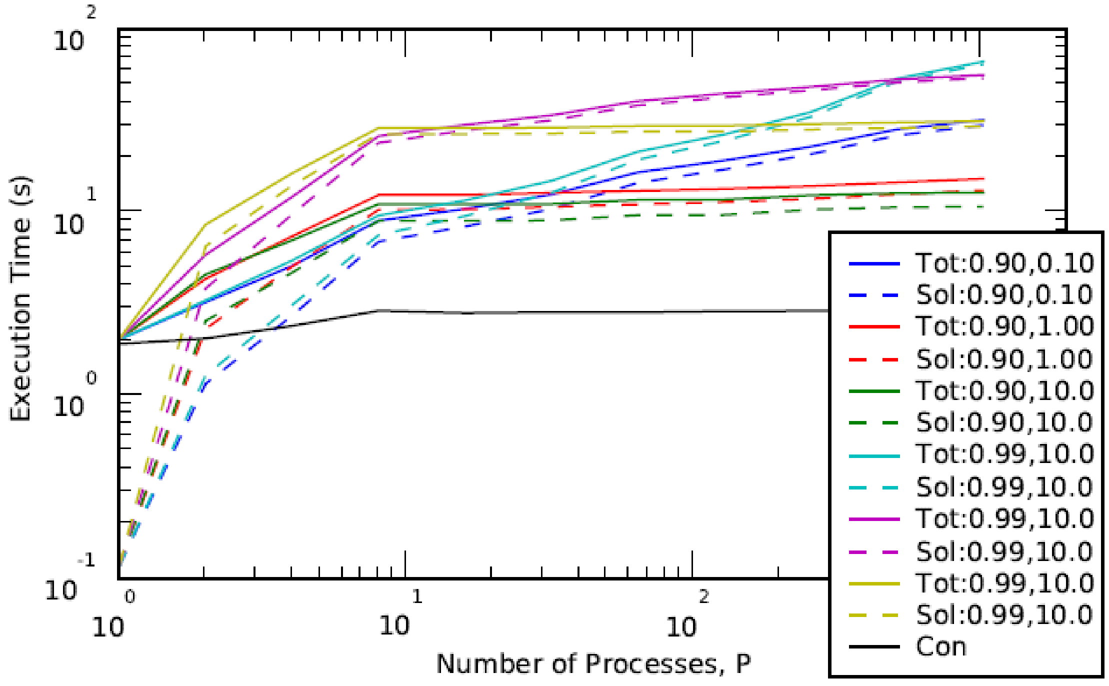


Fig. 3.4.18 JPF PBJ weak scaling, Execution Time *vs. P*, 8×8×8 model, varying *c*, *h*.

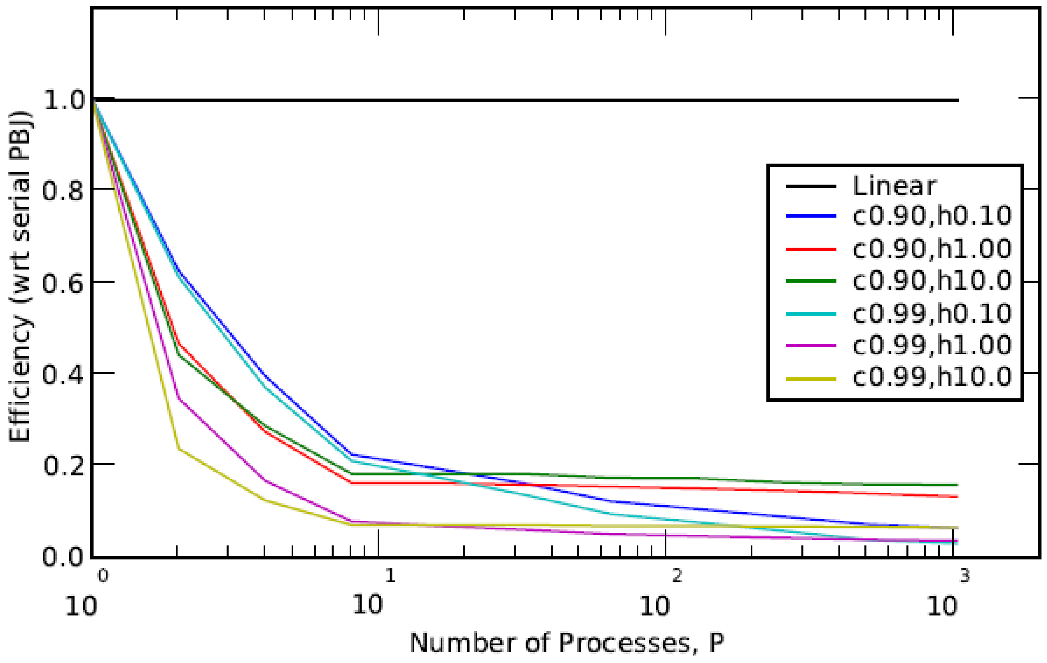


Fig. 3.4.19 JPF PBJ weak scaling efficiency relative to serial PBJ runs, varying *c*, *h*.

3.4.4 PGS Scaling Results

The PGS method is expected to have the advantage of reducing construction time, and assuming the number of sub-domains per PE is not too high, it should maintain a slow growth in the number of global iterations. In this section, the weak scaling studies of the previous section are repeated with a single level of sub-domain division (i.e., two sub-domains per dimension, eight total) on each PE. Then the PGS method is compared to parallel SI via KBA sweeps to show the performance relative to the state-of-the-art. Also the impact of further divisions of sub-domains is investigated with highly massively parallel calculations. Lastly we examine the performance in cases where material heterogeneity plays a significant role in SI iterative convergence.

3.4.4.1 Comparing PBJ and PGS performance

Comparing the PGS and PBJ methods involves evaluating the relative performance amid competing effects. The superlinear reduction in operator sizes leads to smaller memory requirement per PE, faster ITMM construction time, and faster matrix-vector operations for PGS in spite of the fact that each PE handles multiple sub-domains. Conversely, per the results of the strong scaling studies, it is known that for a fixed problem, increasing the number of sub-domains will induce an increase in the number of iterations for convergence. Moreover, the increase in the number sub-domains per PE requires an increase in the number of communications to other PEs as well as substantial copying of arrays among sub-domains on the same PE to transfer data.

Weak scaling studies were performed on the JPF system with an 8×8×8, S_{16} base model. The PGS runs correspond to eight 4×4×4 sub-domains per PE. For this brief comparison, the scattering ratio was held constant for all materials while the optical thickness of each cubic cell was varied by setting h = 0.1, 1.0, and 10.0 cm (and holding σ_t constant).

The iteration counts for c = 0.9 cases are shown in Fig. 3.4.20. As expected, optically thicker cases have flatter iteration count curves. When sub-domains are optically thick, their effects tend to become more localized. This reduces the iteration count because the sub-domains *decouple*, and the influence of the flux iterate in one sub-domain on the flux in other sub-domains quickly diminishes below the specified convergence criterion. The fact that the PGS method demonstrates this behavior as well as PBJ is significant, because it indicates that the PGS method also obeys the decoupling trend that had been previously established for PBJ [7] and [8]. Furthermore, the number of PGS iterations remains below the PBJ counterpart cases indicating, at least for these cases, that the PGS convergence rate is the more dominant effect in comparison with the trend of increasing iteration counts with increasing number of sub-domains.

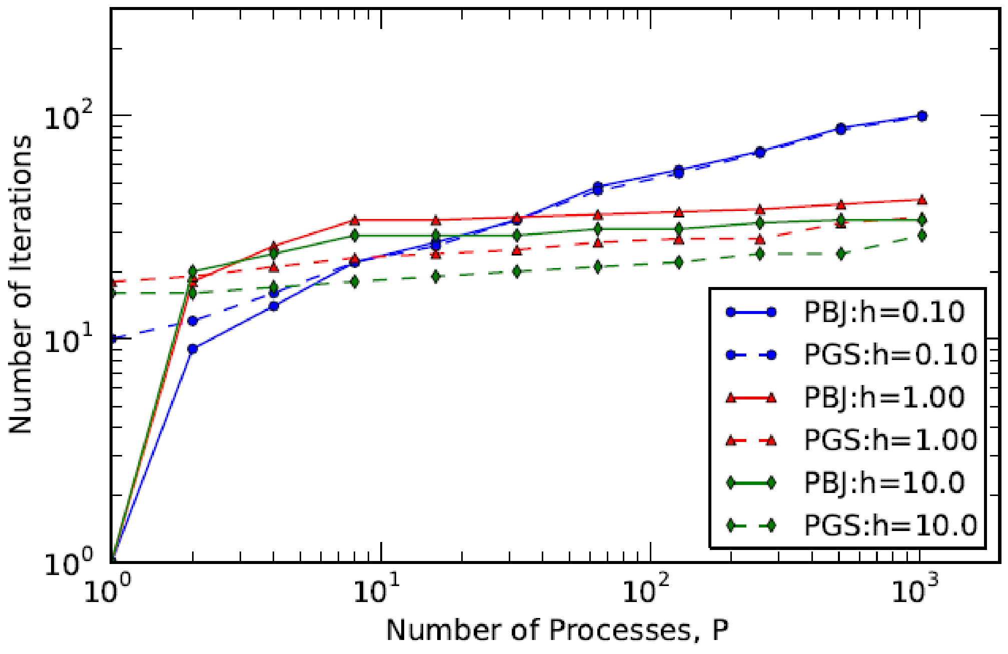


Fig. 3.4.20 JPF PBJ-PGS weak scaling comparison, Iterations *vs*. *P*, *c*=0.90, varying *h*.

The corresponding execution time plots for these cases are presented in Fig. 3.4.21. The savings in the operator construction time is immediately evident (from the two curves with the “Con” label). The superlinear relationship of operator sizes to number of cells clearly outweighs the cost of having to loop over the linear increase in number of sub-domains. The ITMM operator construction time is essentially constant because the number of sub-domains per PE and the sub-domain sizes (number of cells) are constant with increasing P. The remaining curves represent the total execution time. One can deduce from the plots that savings are also realized in the iterative section of the code for these cases. Figure 3.4.21 also shows that the optically thicker PGS cases are still approximately 3–4 times faster than PBJ at P = 1,024. Yet the two optically thicker PGS cases are trending closer toward PBJ execution times because of the slight increase in iteration counts. The results also reinforce the decoupling trend observed in the iteration count curves: namely that optically thick sub-domains have a slow growth in execution time with increasing *P*, and optically thinner sub-domains have a steeper growth.

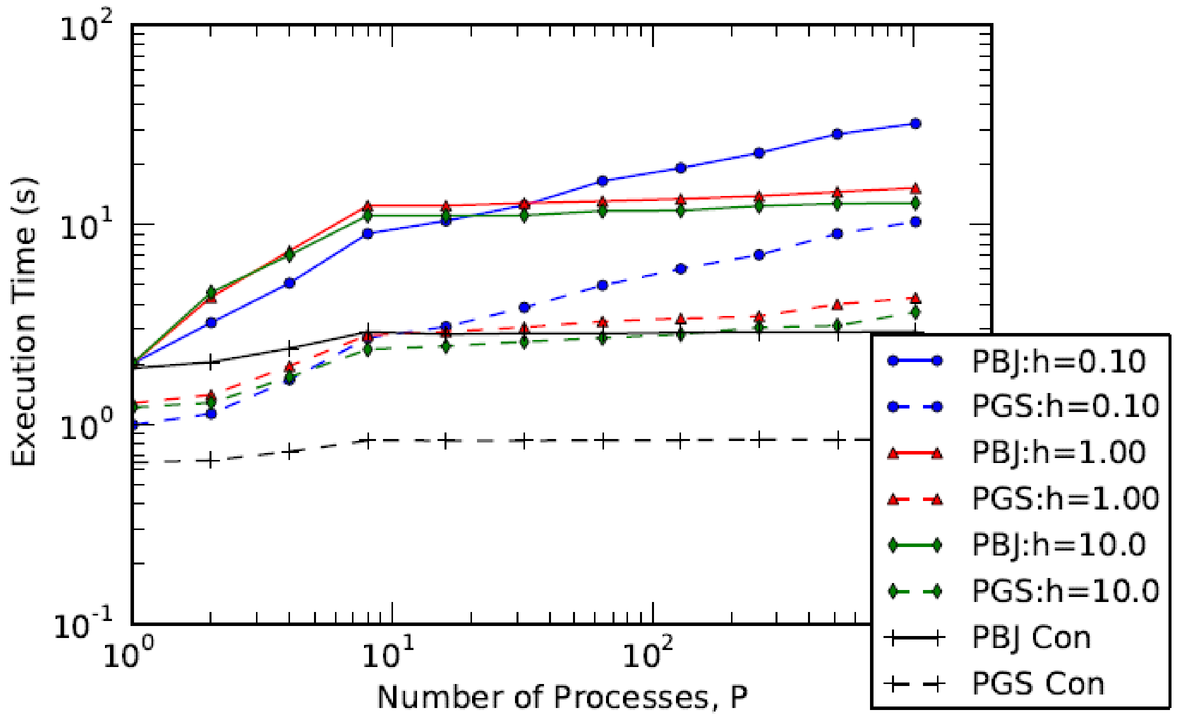


Fig. 3.4.21 PBJ-PGS weak scaling comparison, Execution Time *vs*. *P*, *c*=0.90, varying *h*.

Up to *P* = 1,024, the PGS method exhibits improved performance trends *versus* PBJ due to reduced operator construction times and matrix-vector arithmetic instructions execution times. Although the results presented thus far show a decrease in the number of PGS iterations from PBJ iterations and consistent trends, it remains to be seen if this behavior continues with increasing number of red/black sub-domain divisions. Further divisions of the sub-domains for the PGS approach will lead to higher iteration counts, and the tradeoff between the decreasing work per iteration and the increasing number of iterations is examined via numerical tests presented next.

3.4.4.2 Comparing Levels of PGS Sub-Domain Division in the Very Massively Parallel Regime

Evident in the previous section, PGS offers significant advantages over PBJ, demonstrating comparable scaling trends with reduced execution time. The next task is to examine how increasing the number of red and black sub-domains affects the iteration count and execution time curves *vs P*. Increasing the level of PGS sub-domain division is achieved by successive division of a single sub-domain on a PE (PBJ) into smaller sub-domains. The results in the previous section considered a single division in each dimension to yield eight sub-domains per processor. The base model is modified to a 16×16×16 cells per PBJ subdomain, S_8 problem. For PGS successive divisions are considered such that each PE is assigned eight 8×8×8 (R/B-02), 64 4×4×4 (R/B-04), 512 2×2×2 (R/B-08), and 4,096 1×1×1 (R/B-16) sub-domains. Test cases were performed for a varying scattering ratio, c = 0.90 and 0.99, and varying cell size, h = 0.1 and 1.0 cm. Results are presented for three (c, h) pairs to highlight the effects of increasing scattering ratio and optical thickness: (0.90, 0.1 cm), (0.90, 1.0 cm), and (0.99, 1.0 cm). Computations were performed up to P = 32,768.

The iteration count results for the (0.90, 0.1 cm) case are presented in Fig. 3.4.22. All four levels of PGS division follow the same trend of increasing number of iterations. The curves are shifted upward for increasing number of sub-domains as expected. The execution times for these cases are presented in Fig. 3.4.23. The total execution time ("Tot") is the sum of the iterative solution time ("Sol") and the operator construction time ("Con"). The total execution time for low P shows that the smaller sub-domains perform better, likely caused by the faster operations performed per iteration and the reduced time for ITMM operators' construction. However, for large problems, the construction time becomes increasingly negligible in the total execution time. The larger number of iterations burdens the communication network and the R/B-08 case eclipses R/B-04 and R/B-02 in execution time. The R/B-04 execution time steadily remains below the R/B-02 execution time. These two cases show similar curves with R/B-04 having a slightly faster growth than R/B-02, especially as P is increased to thousands of PEs.

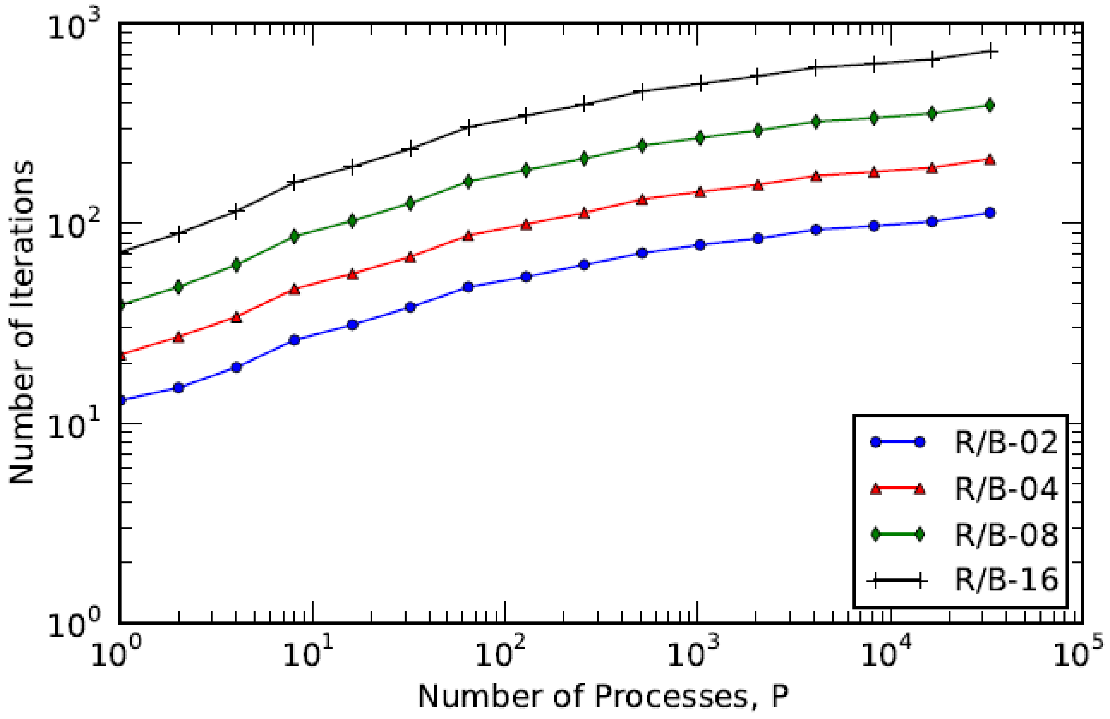


Fig. 3.4.22 JPF R/B division scaling, Iterations *vs*. P, c=0.90, h=0.1 cm.

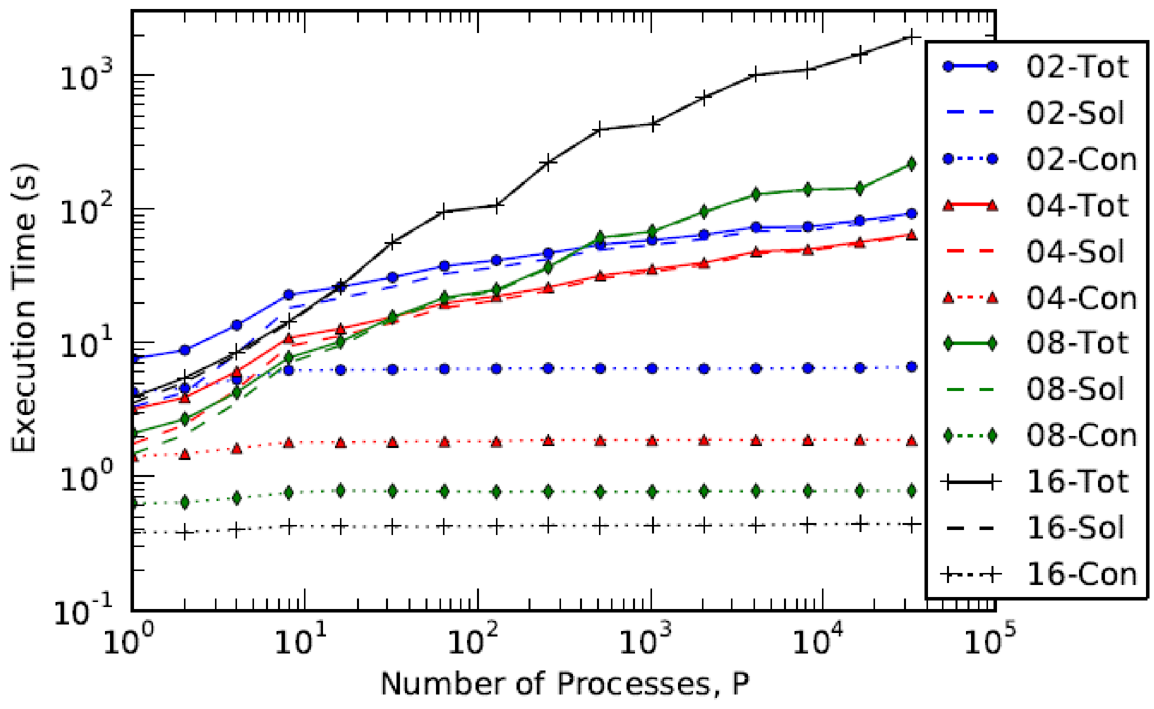


Fig. 3.4.23 JPF R/B division scaling, Execution Time *vs*. *P*, *c*=0.90, *h*=0.1 cm.

Increasing the optical thickness to(0.90, 1.0 cm), the iteration count curves in Fig. 3.4.24, exhibit a less steep slope than their counterparts in Fig. 3.4.22. This is expected due to the decoupling nature of thicker sub-domains. Moreover, Fig. 3.4.24 reveals that increasing optical thickness does not affect the relation among the increasing PGS divisions. The curves all follow the same general trend, albeit with less smooth increases, and the curves do not intersect. In Fig. 3.4.25, the execution time plots are shifted downward due to the decrease in the iteration counts compared to the optically thin case. However, many of the same trends persist, specifically that as *P* is increased to the thousands, R/B-04 executes fastest, and that R/B-08 and R/B-16 have steeper growth in execution time even though the iteration count curves have similar slope to the coarser PGS division.

Increasing the scattering ratio for the (0.99, 1.0 cm) case has the effect of re-tightening the coupling among the sub-domains. As shown in Fig. 3.4.26, the iteration curves are shifted upward, and the higher scattering ratio exhibits faster growth in the number of iterations. However, as in the case of increasing optical thickness, increasing scattering ratio does not impact the behavior that all divisions have very similar iteration count curves, shifted upward with increasing PGS division. In Fig. 3.4.27, R/B-04 remains the fastest PGS division and has an execution time that closely follows the iteration count curve as *P* is increased beyond 1,000.

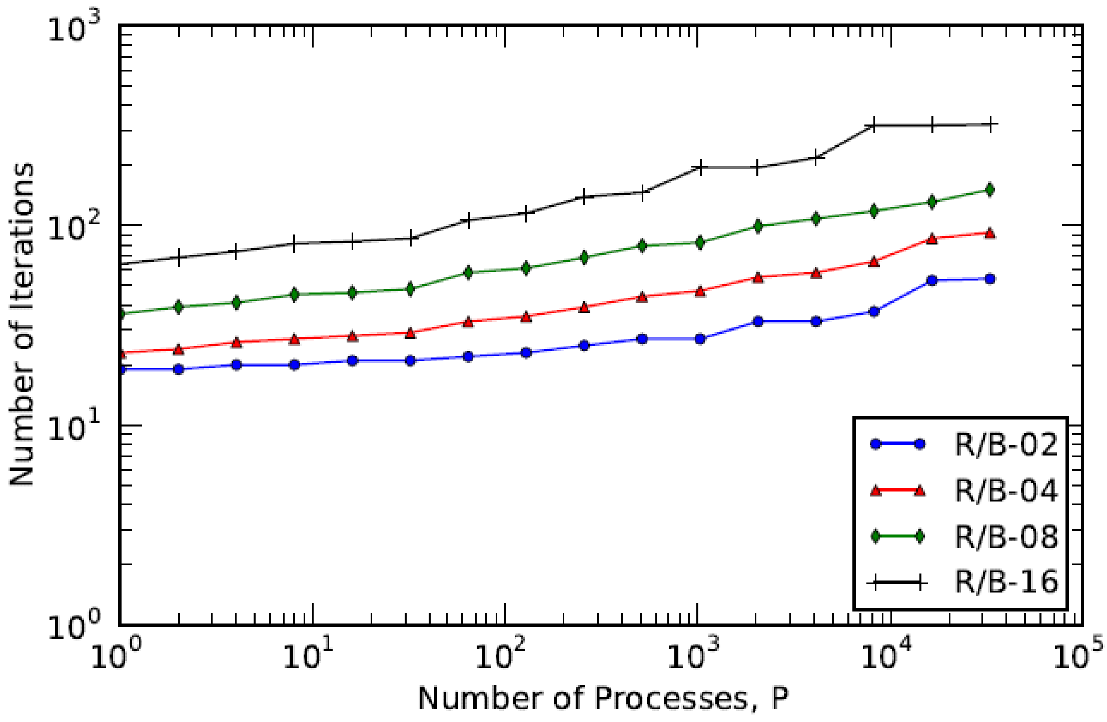


Fig. 3.4.24 JPF R/B division scaling, Iterations *vs*. *P*, *c*=0.90, *h*=1.0 cm.

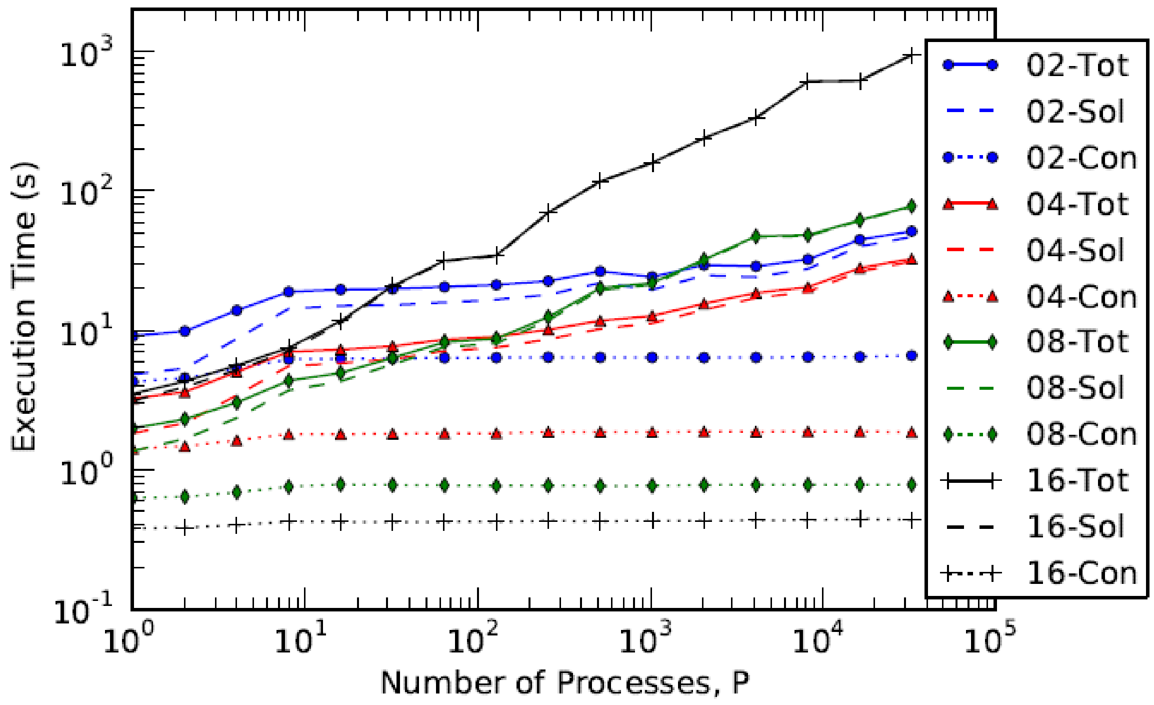


Fig. 3.4.25 JPF R/B division scaling, Execution Time *vs*. *P*, *c*=0.90, *h*=1.0 cm.

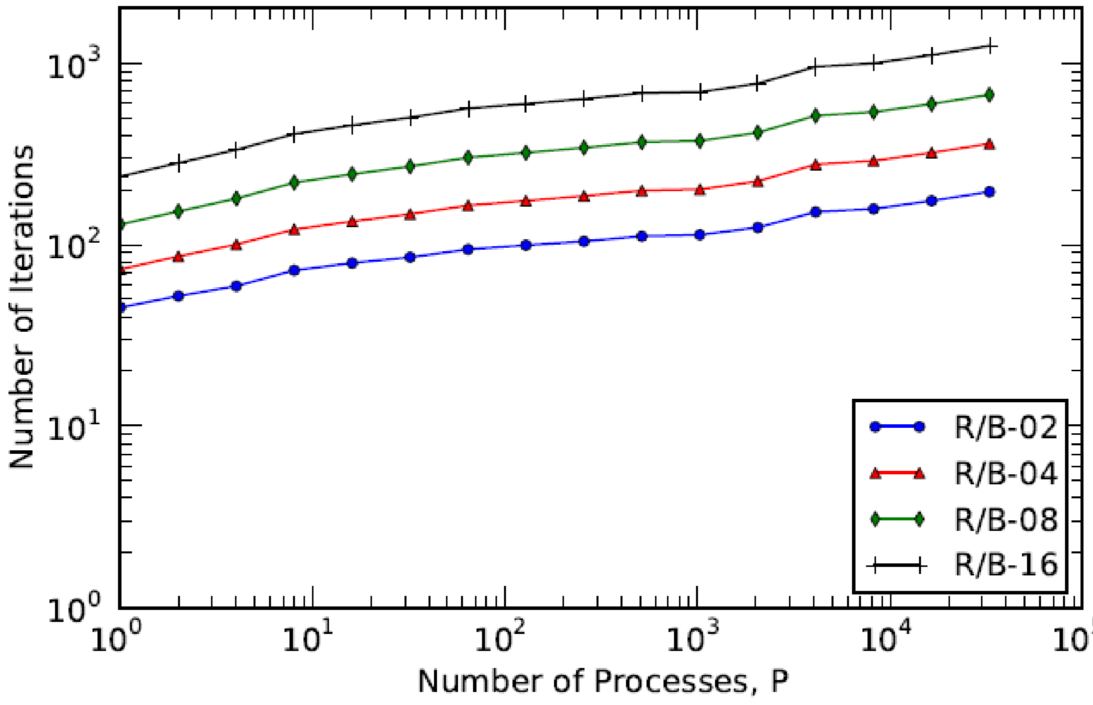


Fig. 3.4.26 JPF R/B division scaling, Iterations *vs*. *P*, *c*=0.99, *h*=1.0 cm.

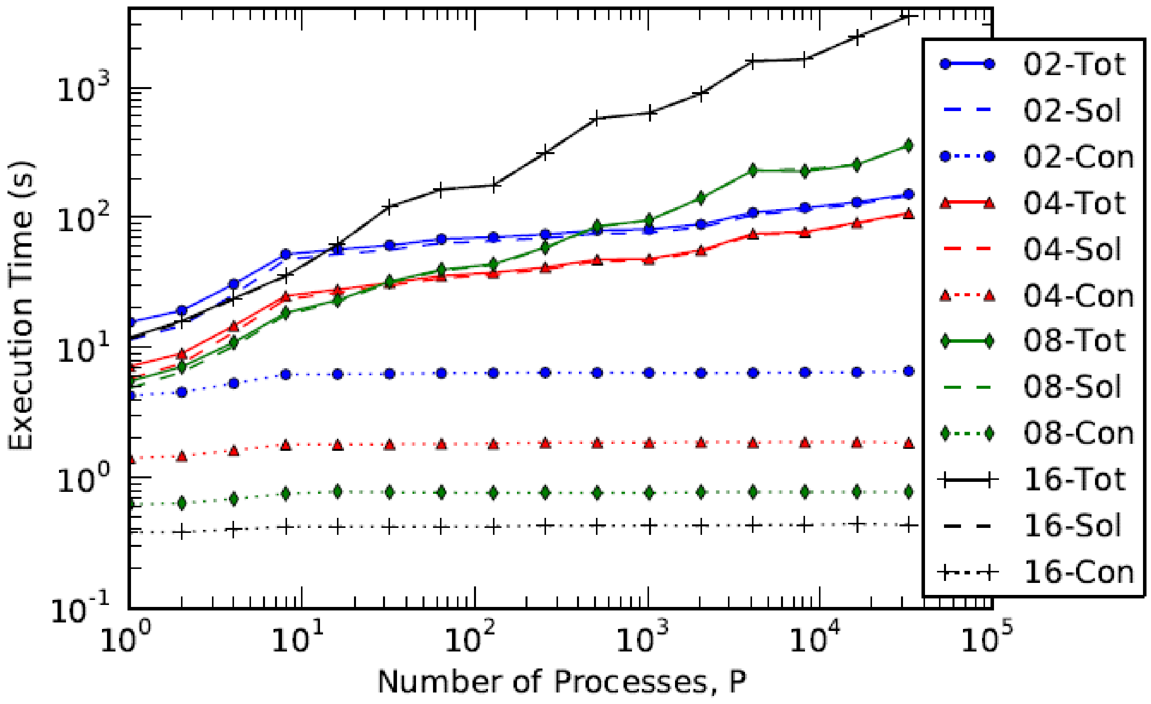


Fig. 3.4.27 JPF R/B division scaling, Execution Time *vs*. *P*, *c*=0.99, *h*=1.0 cm.

The results presented in this section clearly indicate that the R/B-02 and R/B-04 cases scale very similarly. The fact that R/B-04's execution time does not rapidly climb with increasing *P*, like R/B-08 and R/B-16, makes it a very good choice beyond *P* = 32,768. More generally, the results suggest a single division in each dimension, like R/B-02, is not necessarily the most efficient approach, even if it does mean similar or fewer global iterations than PBJ. Moreover, the number of iterations and the burden it places on the system for fast communications is probably the most influential factor in the fast growth of R/B-08 and R/B-16 execution times. If a method is developed to accelerate the ITMM with PGS iterations, more, smaller sub-domains may constitute a more attractive choice because of the faster operator construction time and per iteration matrix-vector operations time.

3.4.4.3 Comparing PGS and KBA performance

Evident from the results above, PGS is a more attractive solution method than PBJ, at least for a modest number of sub-domains per PE. Therefore, the PGS method with the R/B-04 division was selected for a comparison of the parallel ITMM approach against the traditional KBA method for mesh sweep parallelization. The same test configuration as in Sec. 3.4.2 was used with PARTISN and its KBA method and a set of six tests were performed, varying *c* = 0.90 and 0.99 and *h* = 0.1 cm, 1.0 cm, and 10.0 cm.

PARTISN solves the within-group equations with the SI scheme and uses DSA (SI DSA) to improve the iterative convergence rate. The ITMM code, PIDOTS, currently does not include acceleration or preconditioning of the global problem, and this remains an open challenge for improving these methods. For KBA in PARTISN, additional PEs are added in the z- and y-directions before the x-direction to maximize parallel efficiency—i.e., the 2-D spatial decomposition for KBA is done in the y-z plane. [10] This order is consistent with the above tests. Moreover, tests for negative flux fixup were not activated in PARTISN runs to be consistent with PIDOTS, which does not have negative flux fixup. However, every cell is assigned a fixed distributed source to reduce the potential for incurring negative fluxes.

Figures 3.4.28 and 3.4.29 show the iteration and execution time plots, respectively, for the (0.9, 0.1 cm) case. It must be noted that the work per iteration is very different for the three methods presented: PGS, SI, and SI DSA. These plots of iteration counts are used to observe trends in the amount of work to be performed per PE per method. For the present case the PGS and SI cases show a modestly fast growth in the number of iterations with increasing *P*. Yet SI DSA convergence is unsurprisingly rapid and the curve is flat, except for a jump at very high *P*; the acceleration scheme quickly kills diffusive error modes and the small optical thickness makes for a so-called "leaky" system, handled well by the transport sweeps. [19] Up to *P* ~ 400 the ITMM construction time is longer than the entire SI DSA solution time. Moreover, for low *P*, SI and SI DSA exhibit significantly faster execution time than PGS. However, both these curves grow much steeper than the PGS one, needing to perform a mesh sweep every iteration. Additionally, every time the domain is expanded in the x-direction, the efficiency of the KBA method is diminished, as indicated most noticeably by a change in the slope of the SI execution time curve from *P* = 2,048 to 4,096. At *P* ~ 10,000 the SI execution time eclipses that of PGS. However, even at the highest *P* tested, the acceleration by DSA is still effective enough to execute faster than PGS.

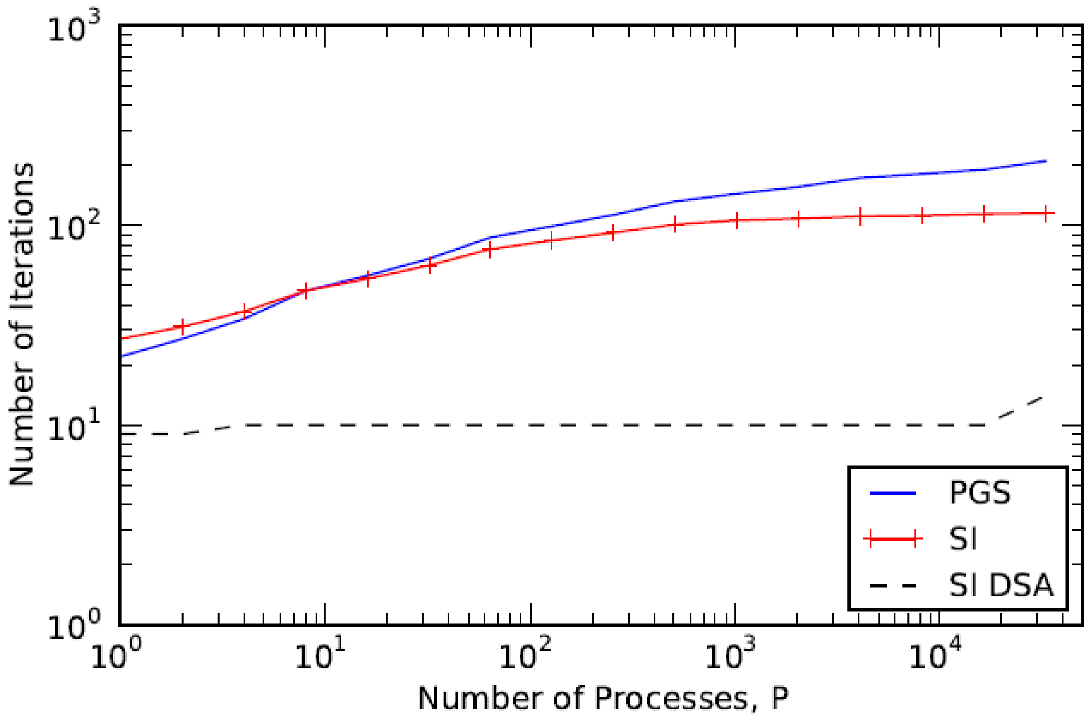


Fig. 3.4.28 JPF PGS-KBA weak scaling comparison, Iterations *vs*. *P*, *c*=0.90, *h*=0.1 cm.

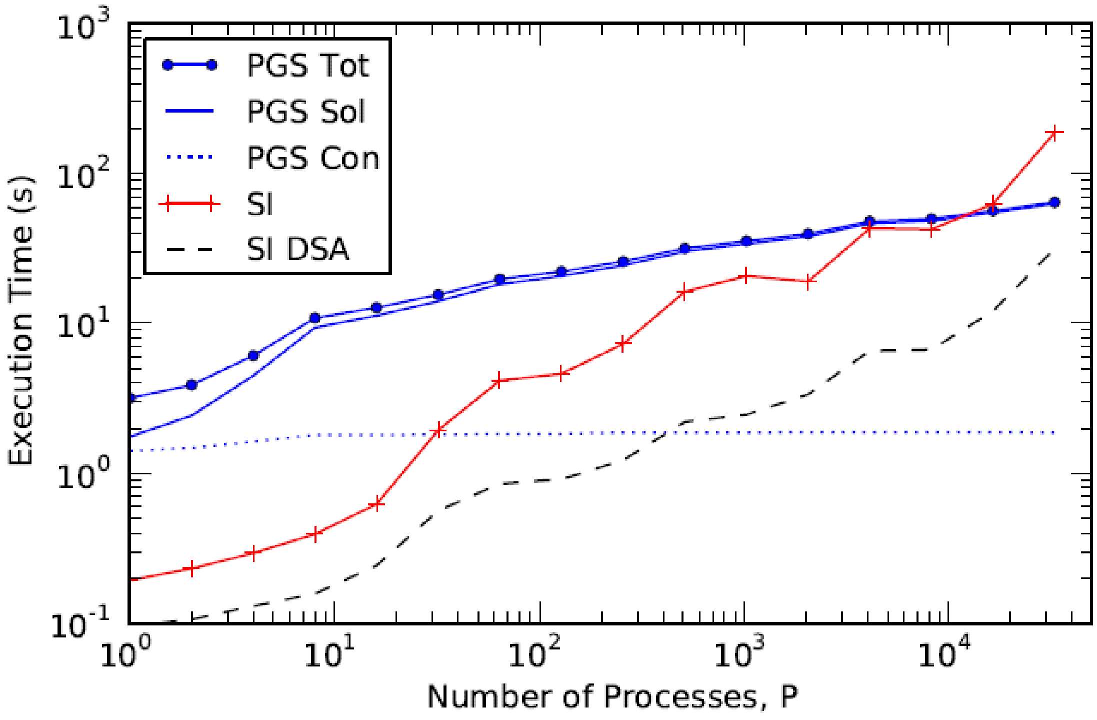


Fig. 3.4.28 JPF PGS-KBA weak scaling comparison, Execution Time *vs*. *P*, *c*=0.90, *h*=0.1 cm.

The results for the (0.9, 1.0 cm) case are given in Figs. 3.4.30 and 3.4.31—iteration counts and execution times, respectively. Now the number of iterations remains relatively flat for SI and SI DSA, but has increased slightly from the *h* = 0.1 case, whereas the number of PGS iterations has decreased from the optically thin case, as expected. The PGS R/B-04 method has a steady growth in iteration count, indicating that the sub-domains are not yet fully decoupled, and some effects are not completely localized. The increasing size of the system has little effect on the necessary number of source iterations for convergence when *P* > 1,000. Nonetheless, the decrease in number of PGS iterations compared to the *h* = 0.1 case reduces its execution time. The PGS execution time curve grows more slowly than SI and SI DSA, intersecting the SI time curve at a smaller value of *P* than the optically thin case. At *P* = 32,768, the SI DSA execution time is only a few seconds shorter than PGS, but has a much steeper up-slope.

Increasing the cell thickness to 10.0 cm has little effect on the PGS iterations, shown in Fig. 3.4.32, maintaining a pretty similar trend to the *h* = 1.0 cm case. Interestingly, the SI curve is essentially unaffected by the increased thickness. However, the SI DSA curve is shifted sharply upward. Although SI DSA is insensitive to *P* (domain size) for this case, the method clearly performs worse for optically thick media. The consequence is easily identifiable in the execution time plots of Fig. 3.4.33. The PGS and SI curves have moved little compared to the *h* = 1.0 cm case, but the SI DSA curve is much closer to SI, evidence of DSA's reduced effectiveness for these problems. Most importantly, beyond P ~ 2,000 the PGS method with R/B-04 division executes faster than SI DSA. This indicates the type of problem where PGS is most competitive with accelerated SI and KBA sweeps: optically thick cases where PGS can take advantage of sub-domain decoupling and avoid the repetitive mesh sweeps.

Increasing the scattering ratio is expected to increase the number of iterations for all the methods as neutron removal by absorption is weakened. Starting with the (0.99, 0.1 cm) case, the numbers of iterations versus *P* are plotted in Fig. 3.4.34. The PGS and SI cases show a quickly growing number of iterations; with the high scattering ratio, cells and sub-domains are tightly coupled. SI DSA convergence is again much more rapid, and the iteration count only begins to grow for *P* > 256. The execution time plots, Fig. 3.4.35, show that for this case the SI method consumes more time than PGS starting at *P* ~ 3,000. Furthermore, although the SI DSA time is nearly 100 times faster than PGS at low *P*, the significantly faster growth in SI DSA execution time cuts this difference down to approximately a factor of two at *P* = 32,768. If the ITMM method is to be competitive with SI DSA for optically thin problems, it is clear that reducing the execution time per iteration will likely be insufficient for most practical problems and some acceleration method to improve the convergence rate is necessary.

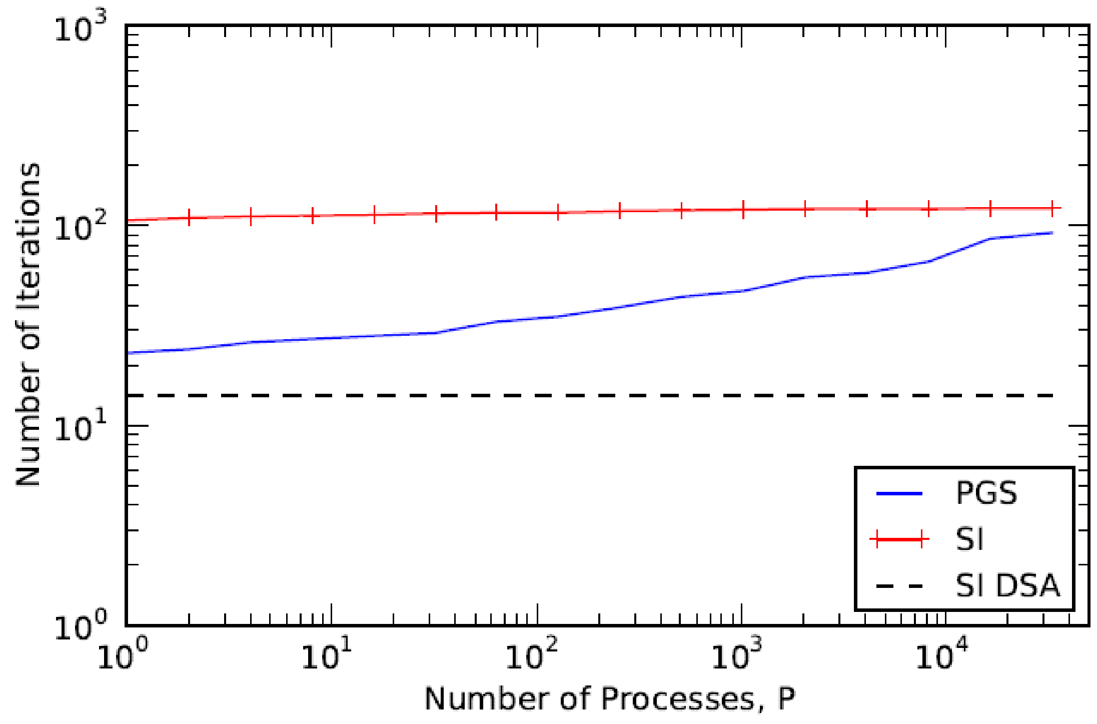


Fig. 3.4.30 JPF PGS-KBA weak scaling comparison, Iterations *vs*. *P*, *c*=0.90, *h*=1.0 cm.

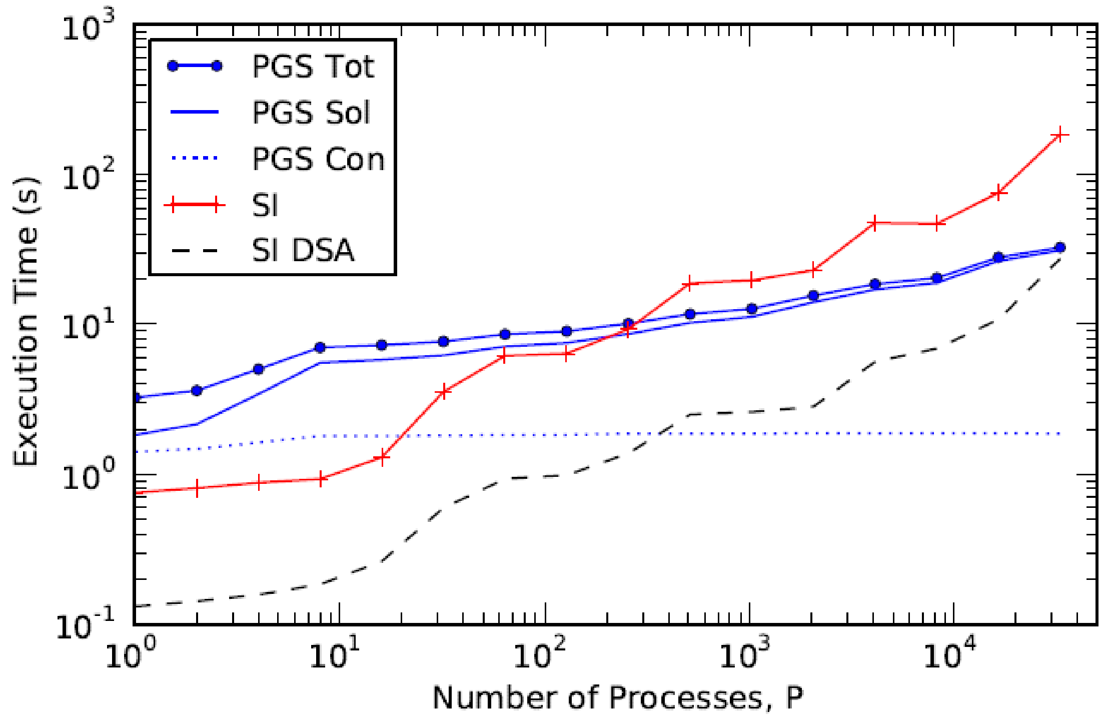


Fig. 3.4.31 JPF PGS-KBA weak scaling comparison, Execution Time *vs*. *P*, *c*=0.90, *h*=1.0 cm.

Compared to the case with a lower scattering ratio, the (0.99, 1.0 cm) case has a large jump, approximately a factor of ten for SI and a factor of 5 for PGS, in the number of iterations, shown in Fig. 3.4.36. PGS also exhibits a trend of steady growth in the number of iterations, while SI and SI DSA are flat for high *P*. The higher scattering ratio is keeping sub-domains more tightly coupled in the PGS method, and the resulting increase in interactions before absorption slows the convergence of the scattering source in SI. SI DSA handles this case well, as the number of iterations has barely increased compared to the *c* = 0.9 case. In Fig. 3.4.37, the execution time plots underscore this observation. The SI DSA time remains much lower than PGS, albeit with a faster rate of growth. The increased scattering ratio does appear to impact SI more negatively than PGS. After KBA efficiency decreases at *P* = 512, the SI execution time is greater than PGS.

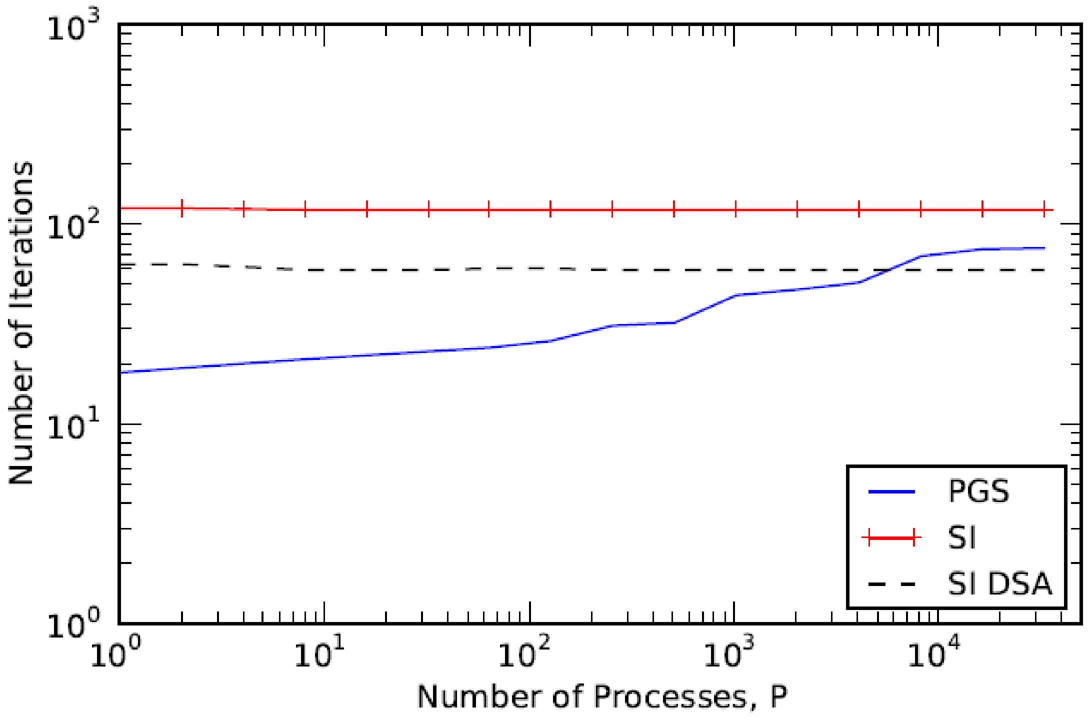


Fig. 3.4.32 JPF PGS-KBA weak scaling comparison, Iterations *vs*. *P*, *c*=0.90, *h*=10.0 cm.

The (0.99, 10.0 cm) case is the most difficult one considered for SI. Increasing optical thickness improves the PGS method's parallel performance because the sub-domains become weakly coupled—transport effects are more localized and fewer iterations are needed. However, the low absorption and high probability of interaction decrease the SI method's ability to quickly converge on the scattering source. Although the SI and SI DSA iteration curves shown in Fig. 3.4.38 are flat, they are both higher than the previous cases considered. The number of PGS iterations is below SI and SI DSA, but the greater amount of work per iteration still results in longer execution times compared to SI DSA until *P* = 2,048. Above this point, the increasing long sweeps and high iteration count for SI DSA grow its execution time longer. The SI and SI DSA time curves continue to grow at a relatively fast rate, whereas the decoupling effect shows excellent scalability for PGS with R/B-04. The execution time plots for this case are depicted in Fig. 3.4.39.

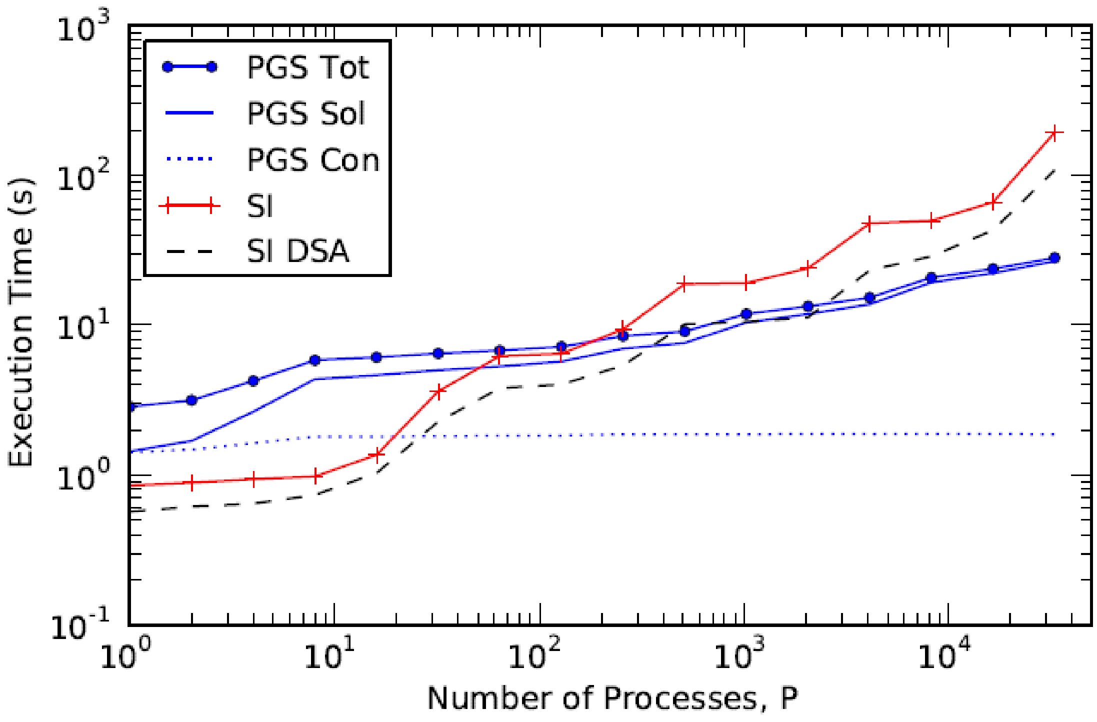


Fig. 3.4.33 JPF PGS-KBA weak scaling comparison, Execution Time *vs*. *P*, *c*=0.90, *h*=10.0 cm.

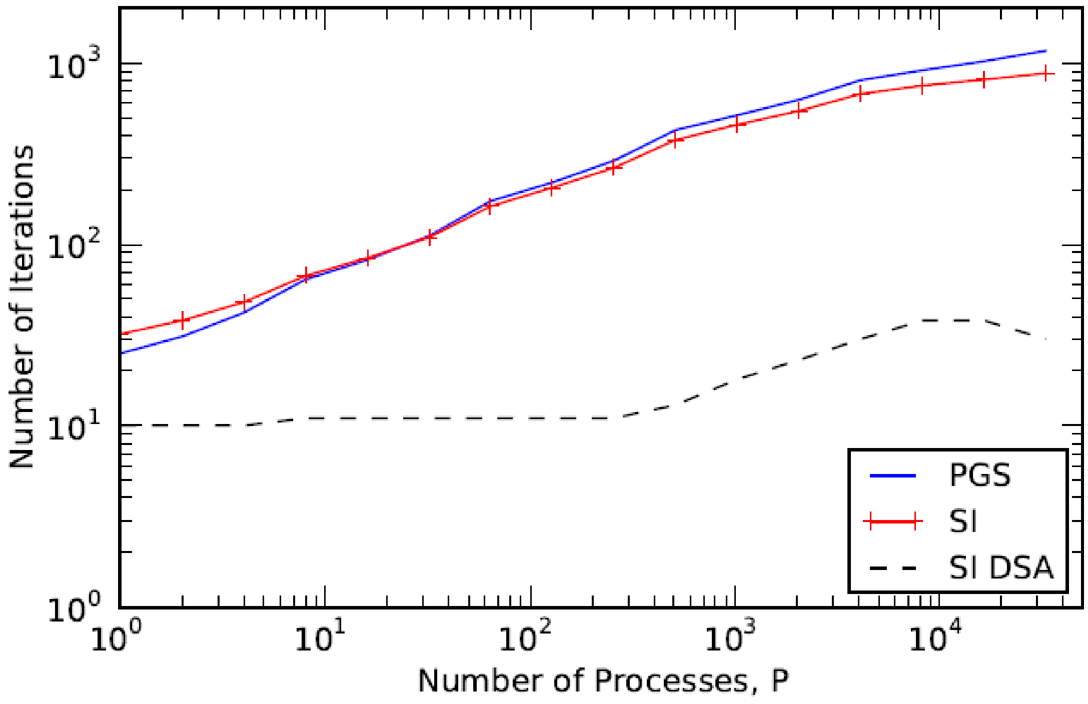


Fig. 3.4.34 JPF PGS-KBA weak scaling comparison, Iterations *vs*. *P*, *c*=0.99, *h*=0.1 cm.

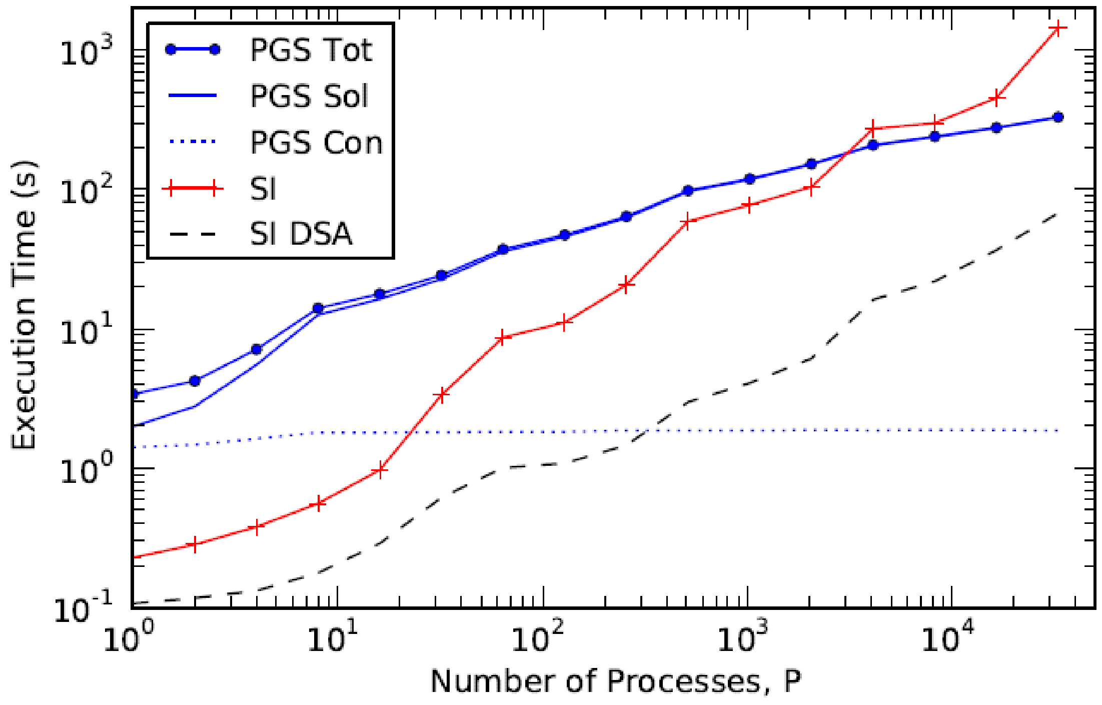


Fig. 3.4.35 JPF PGS-KBA weak scaling comparison, Execution Time *vs*. *P*, *c*=0.99, *h*=0.1 cm.

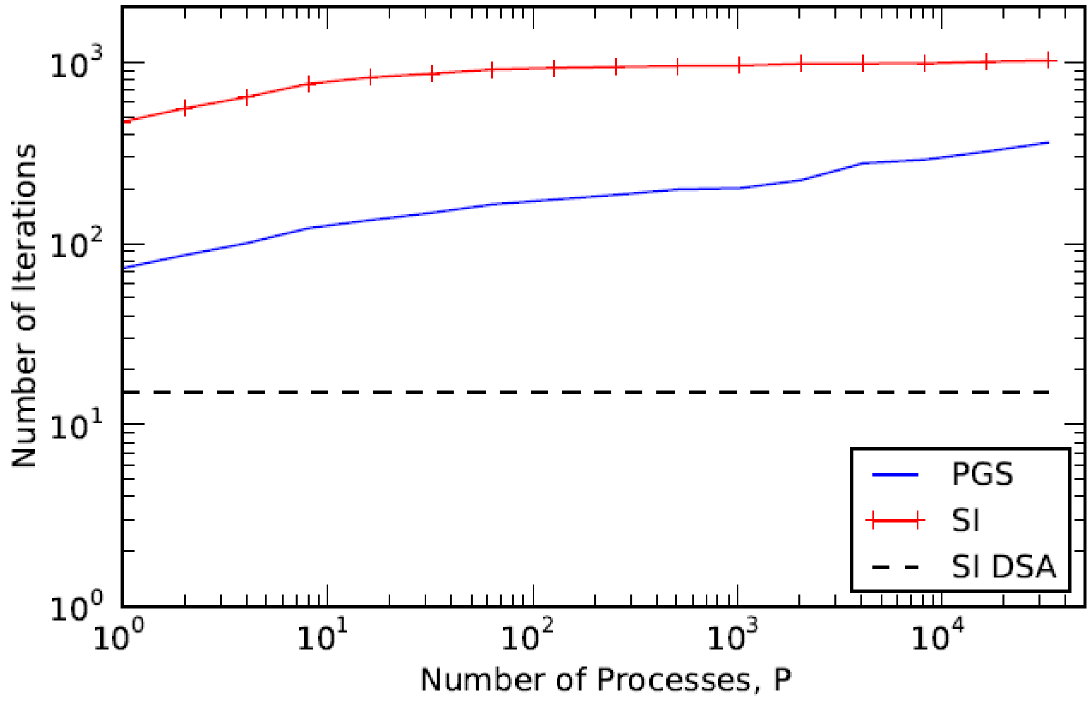


Fig. 3.4.36 JPF PGS-KBA weak scaling comparison, Iterations *vs*. *P*, *c*=0.99, *h*=1.0 cm.

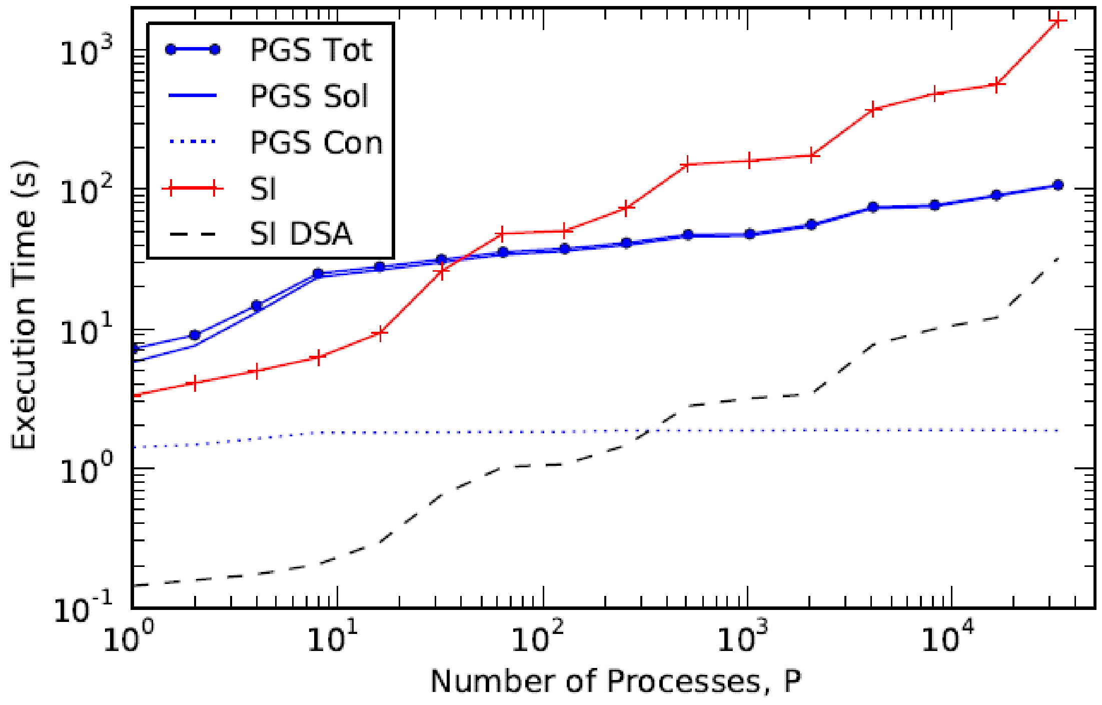


Fig. 3.4.37 JPF PGS-KBA weak scaling comparison, Execution Time *vs*. *P*, *c*=0.99, *h*=1.0 cm.

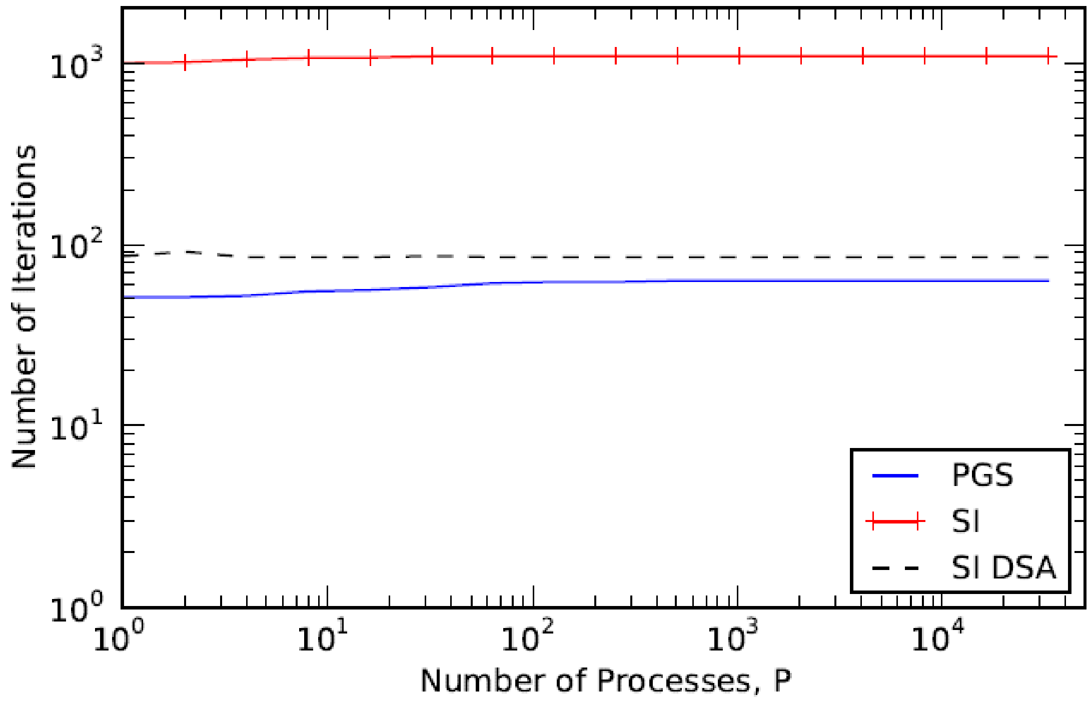


Fig. 3.4.38 JPF PGS-KBA weak scaling comparison, Iterations *vs*. *P*, *c*=0.99, *h*=10.0 cm.

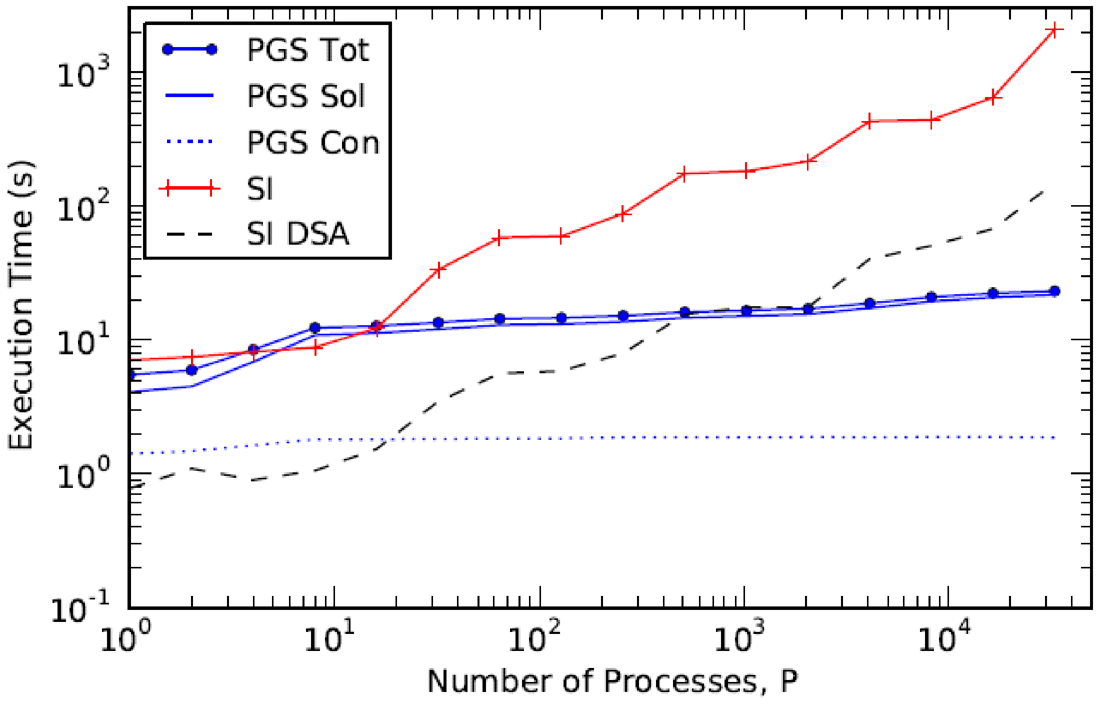


Fig. 3.4.39 JPF PGS-KBA weak scaling comparison, Execution Time *vs*. *P*, *c*=0.99, *h*=10.0 cm.

The results thus far delineate circumstances in which the ITMM with PGS are competitive with SI and SI DSA. Namely, problems with large optical thickness undergo sub-domain decoupling effects that provide PGS an advantage to converge faster. Furthermore, as the problem is scaled to a greater number of cells, the sweeps become increasingly expensive, whereas the ITMM sub-domain sizes remain constant and work per iteration does not change. The parallel solution times of SI and SI DSA continue to grow even when the iteration curves are flat because the size of the domain to be swept continues to grow. However, the large differences that do exist between PGS and SI DSA execution times, especially for optically thin problems, suggest that PGS global convergence must be accelerated to benefit the method in a similar manner as DSA does for SI.

3.4.4.4 Weak scaling studies with the Periodic Heterogeneous Layers configuration

Previous research [29] and [30] has identified that DSA loses its effectiveness to accelerate problems with sharp material discontinuities. Specifically, when a domain is divided into alternating regions with increasingly heterogeneous nuclear properties—i.e., very large and very small optical thicknesses—diffusion-based acceleration techniques fail. This is a significant shortcoming of DSA, but not a completely unexpected one. Azmy [29] noted for this class of problems, the alternating heterogeneity of the system induces highly anisotropic angular flux, and the diffusion approximation, given its assumption of linearly anisotropic angular dependence, is known to break down in such regimes.

A sequence of tests was performed to test the impact that Periodic Heterogeneous Layers (PHL) have on the ITMM global iterations compared to SI and SI DSA. The base model depicted in Fig. 3.4.2 no longer applies to these tests. The base model is now an S_{16}, 8×8×8-cell domain with each cell assigned one of two materials. All cubic-cell dimensions are set to *h* = 1.0 cm and the total cross section is used to scale the optical thickness of the media. The scattering ratio of *c* = 0.9 was set for all materials. The material is homogeneous in the x-y plane but there are eight layers in the z-direction: α-layers 1, 3, 5, and 7 are optically thick, and β-layers 2, 4, 6, and 8 are optically thin. Table 3.4.I lists the total interaction cross sections for the α- and β-layers for the three cases considered. Namely, thickness and thinness are each increased by a factor of *a* away from unity, the homogeneous case.

Table 3.4.I YR PHL total cross sections (or equivalently *mfp* given *h*=1.0 cm).

Case, *a*	$\sigma_{t,\alpha}$ (cm^{-1})	$\sigma_{t,\beta}$ (cm^{-1})
10	10	0.1
100	100	0.01
1000	1000	0.001

The R/B-2 PGS method (i.e. eight 4×4×4 sub-domains per PE) was employed to solve the parallel ITMM problem for comparison with PARTISN's KBA-based parallel SI and SI DSA schemes. It should be clear that the sub-domains are composed of both α and β materials. The ITMM has the distinct advantage of computing explicit coupling among cells within a sub-domain, providing it with a competitive edge in problems with sharp material discontinuities.

Cases were run in parallel on JPF up to *P* = 4,096. Due to time constraints, cases with higher *P* were impractical to be performed for inclusion in this report. The scaling trends for higher *P* are still highly desired, but the presented results do provide a good indication of the comparative performance between PGS and SI. Unlike the previous tests, where the base model was stretched as the problem grew, now additional sub-domains *are* periodic repetitions of the base. A virtual process Cartesian topology is created as before, and PEs are added to this topology one dimension at a time, cycling through the dimensions. Subdomains are added in the direction of additional PEs. Therefore, only when the number of z-processes is increased are additional PHL layers modeled. The cyclic order is still $z \rightarrow y \rightarrow x$. Vacuum boundary conditions were applied on all external boundaries, and every cell is supplied a unit volumetric source.

Starting with *a* = 10, the iteration curves versus *P*, shown in Fig. 3.4.40, illustrate that for this level of heterogeneity the PGS, and SI iteration counts remain largely unchanged. However, the SI DSA method undergoes a large jump in the number of iterations from *P* = 16 to 32. This corresponds to an increase in the y-dimension, possibly indicating a significant decrease in the impact that the boundary conditions have on the DSA algorithm. The execution times are plotted in Fig. 3.4.41. The SI DSA method starts out much faster than the other methods but its execution time increases rapidly to consume approximately the same amount of time as SI. Even though SI DSA iteration counts are less than SI, the two methods use the same amount of time approximately, because SI DSA iterations are more expensive. The PGS method meanwhile exhibits good scaling properties and no discernable problems caused by the material heterogeneity. It outperforms the parallelized SI on as few as 32 processes.

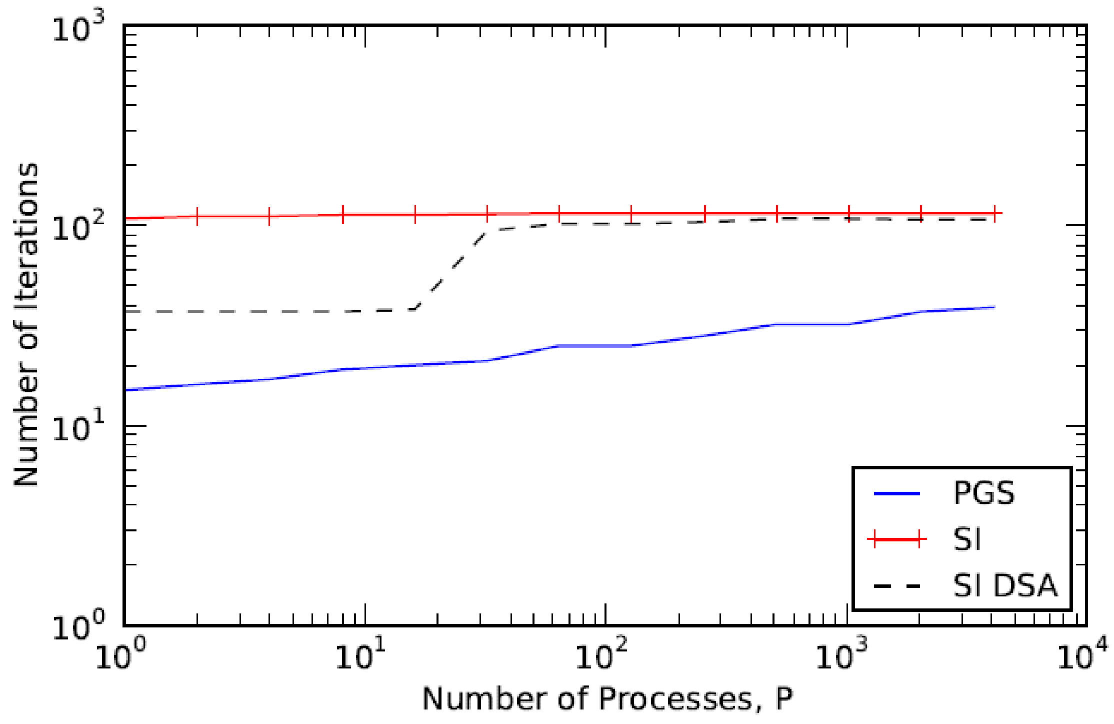


Fig. 3.4.40 JPF PHL scaling study, Iterations *vs*. *P*, *c*=0.90, *a*=10.

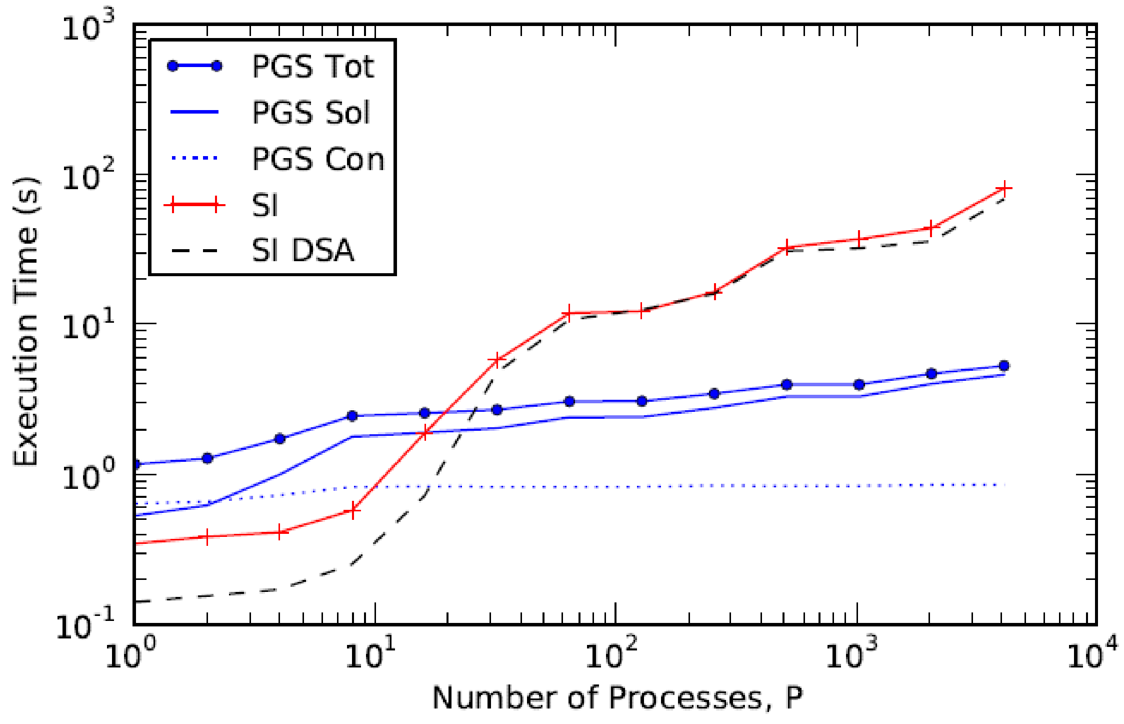


Fig. 3.4.41 JPF PHL scaling study, Execution Time *vs*. *P*, *c*=0.90, *a*=10.

The *a* = 100 case demonstrates the impact of increasing the heterogeneity. In Fig. 3.4.42, the PGS iteration curve grows much steeper, indicating the difficulty in resolving the interface angular flux solution with the increased heterogeneity. The SI and SI DSA curves are shifted upward as well, but it is clear that SI DSA is ineffective in accelerating SI. DSA turns itself off due to negative source terms caused by the large optical thicknesses and the disabling of negative flux fixup. While simple tests suggest for some cases that enabling negative flux fixup can help reduce the number of iterations, it comes at a very high cost, potentially outweighing the benefits. The need for fixup was avoided in this study from the outset to reduce differences among the compared algorithms since it is difficult to incorporate in ITMM-type methods that are based on scalar flux variables. Observing the execution time plots in Fig. 3.4.43, it is clear that SI does not scale as well as PGS in the presence of sharp material discontinuities. Additionally, the ITMM parallel solution method continues to be less affected by the anisotropic effects resulting from the nature of the PHL configuration.

The iteration curves for the *a* = 1,000 case are given in Fig. 3.4.44. Again DSA turns itself off hence the SI and SI DSA iteration curves are identical. Interestingly, increasing *a* from 100 to 1,000 has little impact on the iteration curves for all methods. Perhaps the anisotropic effects are already strongly present for *a* = 100 and further increases do little to impact how the methods perform. For both values of *a*, the SI curves do not grow monotonically. The more complex nature of the problem (i.e., the heterogeneities and the influence of the boundary conditions for different regions) seems to impact the iteration counts in somewhat unexpected ways. The same conclusions can be drawn for the execution time curves, shown in Fig. 3.4.45. Again, the PGS method shows considerably better scaling properties compared to SI.

Most real-world nuclear engineering problems encountered are models of nuclear systems where sharp, periodic material heterogeneity is not present, and DSA effectively annihilates diffusive error modes of the SI scheme. However, for more complicated problems where diffusive error modes are not dominant, the DSA does little to accelerate SI, which itself is observed to be a poorer solution strategy than PGS for growing PHL problem size and *P*. On the other hand, the ITMM seems to resolve more quickly the highly anisotropic nature of the flux via explicit coupling between very different materials within each sub-domain. Moreover, the PGS iterative global solution method does not introduce difficulties that would hinder convergence. Although these problems are encountered far less frequently in neutronics, this study has revealed a class of problems where the ITMM has definite advantages. Moreover, applied to thermal radiative transport problems, where material opacity discontinuities are larger and isotropic scattering is more prevalent, the ITMM may be found to be more beneficially applicable so long as its scaling properties are similar to those observed here.

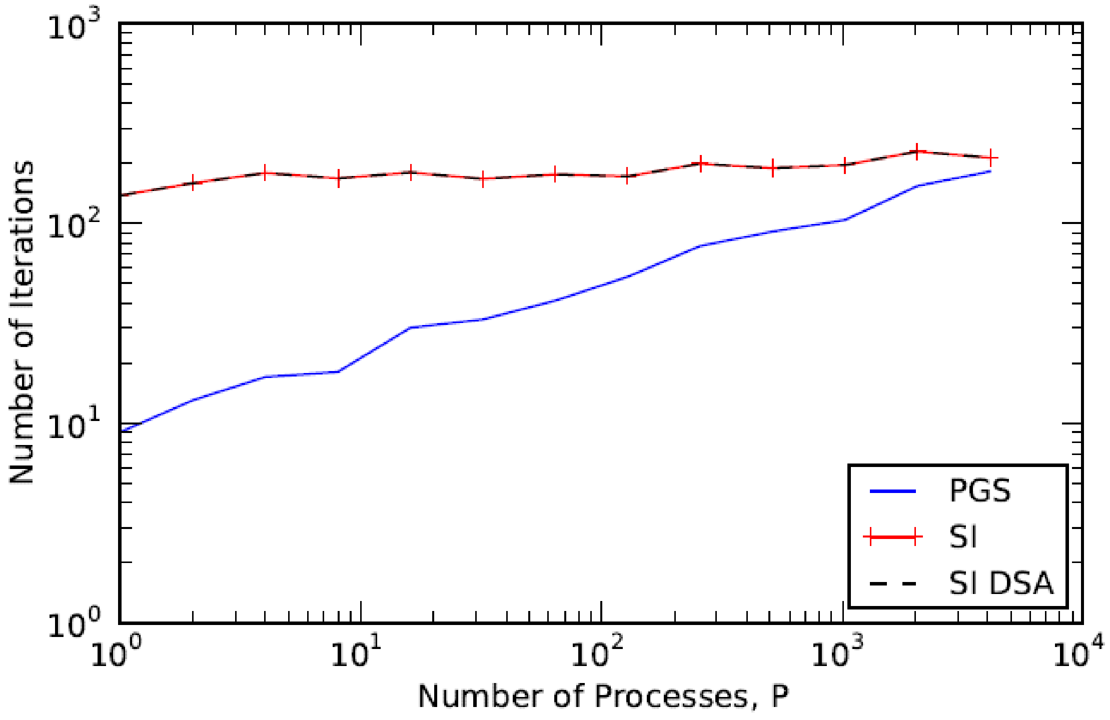


Fig. 3.4.42 JPF PHL scaling study, Iterations *vs*. *P*, *c*=0.90, *a*=100.

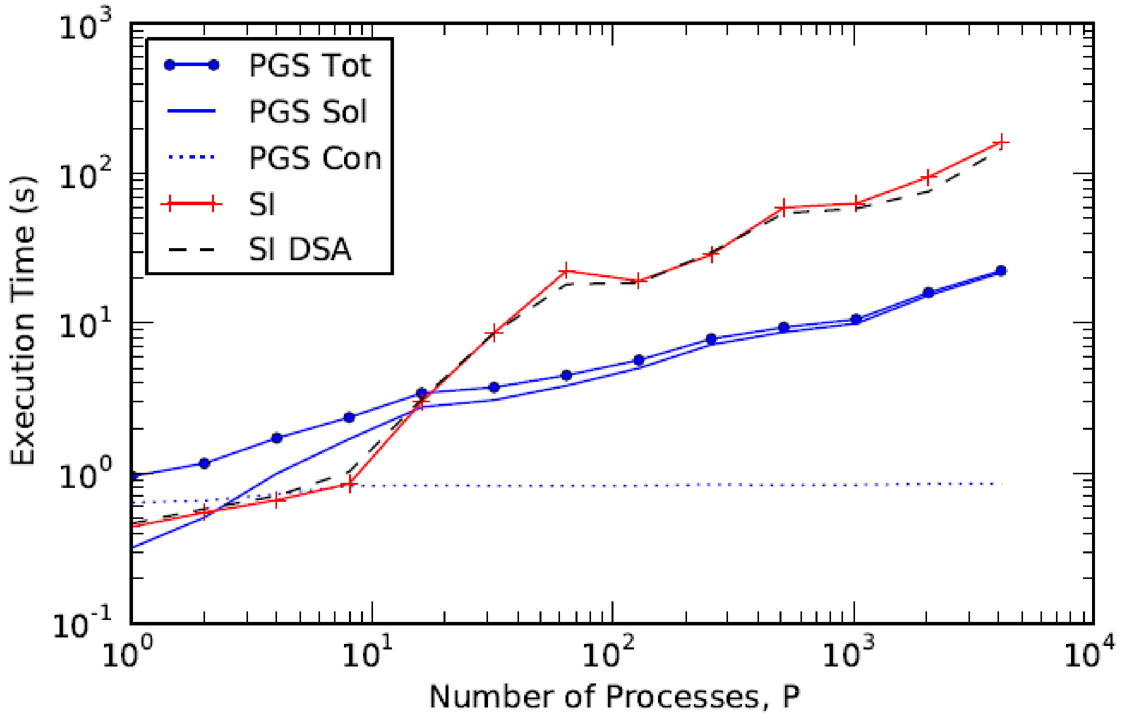


Fig. 3.4.43 JPF PHL scaling study, Execution Time *vs*. *P*, *c*=0.90, *a*=100.

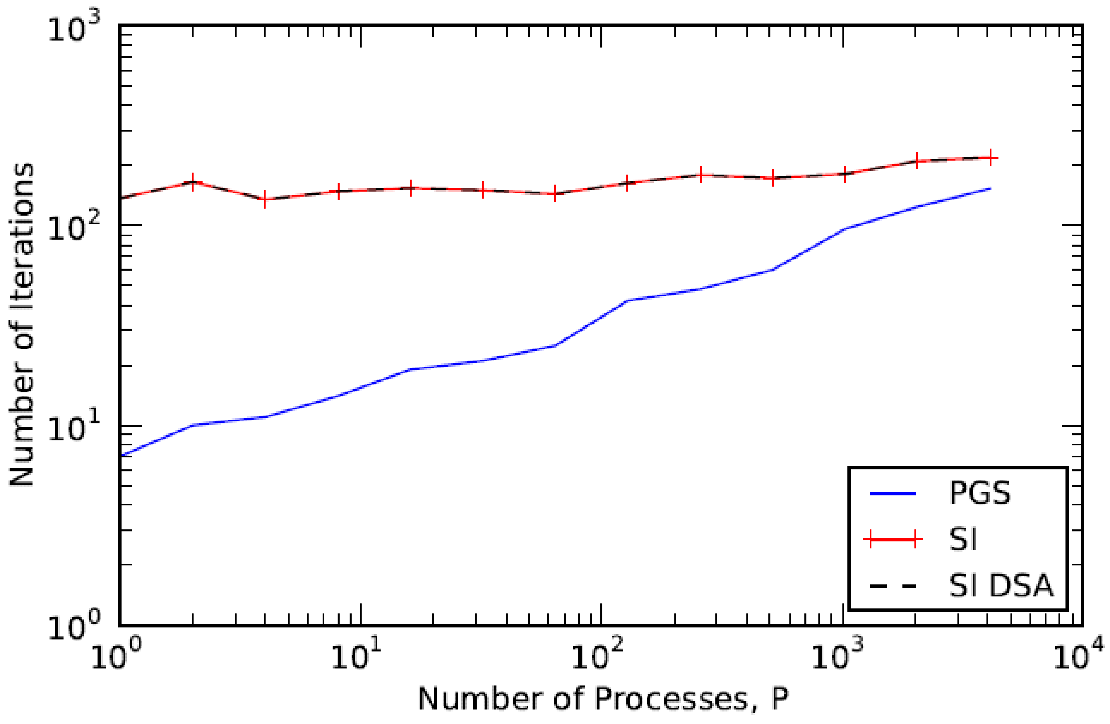


Fig. 3.4.44 JPF PHL scaling study, Iterations *vs*. *P*, *c*=0.90, *a*=1000.

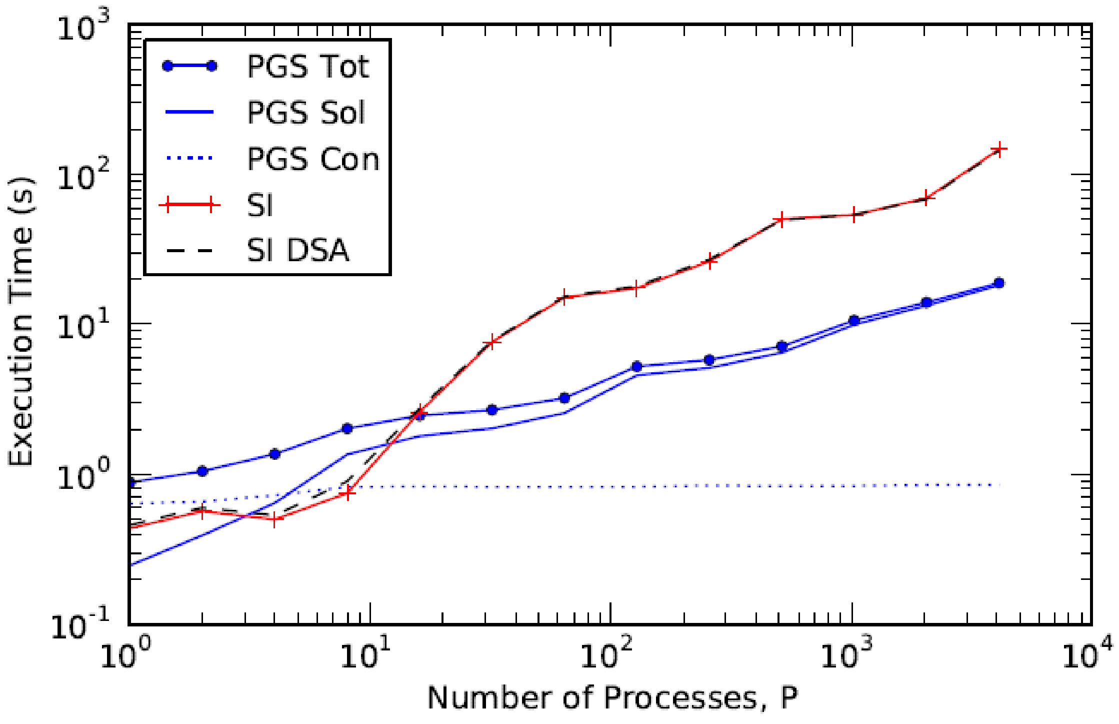


Fig. 3.4.45 JPF PHL scaling study, Execution Time *vs*. *P*, *c*=0.90, *a*=1000.

3.5 Task 5 – Evaluate options for solving local system

In the PSD approach [19] the overall region is divided into several sub-domains, as shown in Fig. 3.5.1. Each sub-domain is treated as an independent problem; each has its own set of ITMM operators and boundary conditions. The scalar flux is computed for all cells in the sub-domain according to Eq. (3.1.12) with methods described in greater detail in the following section. Using this information and the boundary conditions, all the outgoing angular fluxes are computed at the boundaries of the sub-domain with Eq. (3.1.14). The outgoing angular flux is 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 scalar flux in all sub-domains is achieved.

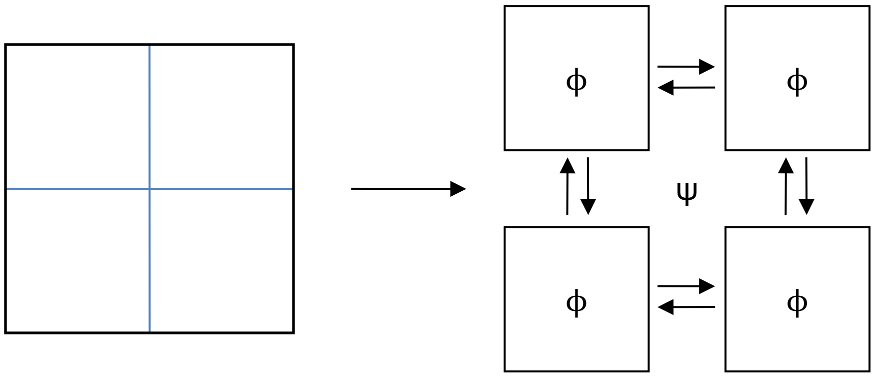


Fig. 3.5.1 Spatial decomposition of a global domain into multiple sub-domains.

This iterative procedure is *not* an inner iteration scheme because it does not iterate on the scattering source. The critical aspect of this iterative method is that the quantities being iterated on are the angular fluxes at sub-domain boundaries, not the scalar fluxes within the sub-domain. The scalar fluxes serve as a computationally inexpensive intermediate step in the computation of updated outgoing boundary angular fluxes and as a convenient tool for evaluating the convergence of the global system. By substituting Eq. (3.1.12) into Eq. (3.1.14) one can remove the scalar flux from the solution process entirely, only computing it when necessary for reaction rate computations. However, the large expenses of inverting the $(\boldsymbol{I}-\boldsymbol{J}_{\phi})$ matrix, N^3, and performing matrix-matrix multiplications with $\boldsymbol{K}_{\phi}$ and $\boldsymbol{J}_{\phi}$, $N^{11/3}$, make the retention of the scalar flux computation during the iterative procedure a computationally efficient option.

All four integral transport matrix operators are necessary for this approach. Constructing the ITMM operators per sub-domain is computationally intensive. However, because the involved operators are completely uncoupled across sub-domains and are pre-computed only once, their construction is highly concurrent. Effectively, the goal of the new algorithm is to shift the computational burden from the solution stage that is sequential in the mesh sweep algorithm to the fully parallel computation of the ITMM operators for each sub-domain. Moreover, because the operations per iteration are much simpler than the mesh sweep, the grind time may potentially decrease, addressing the primary obstacle to drastically improving parallel performance of solution algorithms for transport methods.

To compute the scalar flux distribution for the sub-domain, one must account for the dependence of the fixed (isotropic) source and the expectedly anisotropic boundary conditions, Eq. (3.1.12). The top half of Fig. 3.5.2 illustrates the computation of the scalar flux. The bottom half illustrates the computation of the outgoing angular flux on the faces of the boundary cells from the computed scalar flux spatial moments, source, and boundary conditions, Eq. (3.1.14). Depending on the size of the sub-domains, the operators may still involve large matrices. But due to the rapid growth of the operators with respect to increasing number of cells, sub-domains' cumulative storage demand may be less than the requirement for a single domain, depending on the competing effects: fewer overall cells and increasing number of sub-domain boundary cells.

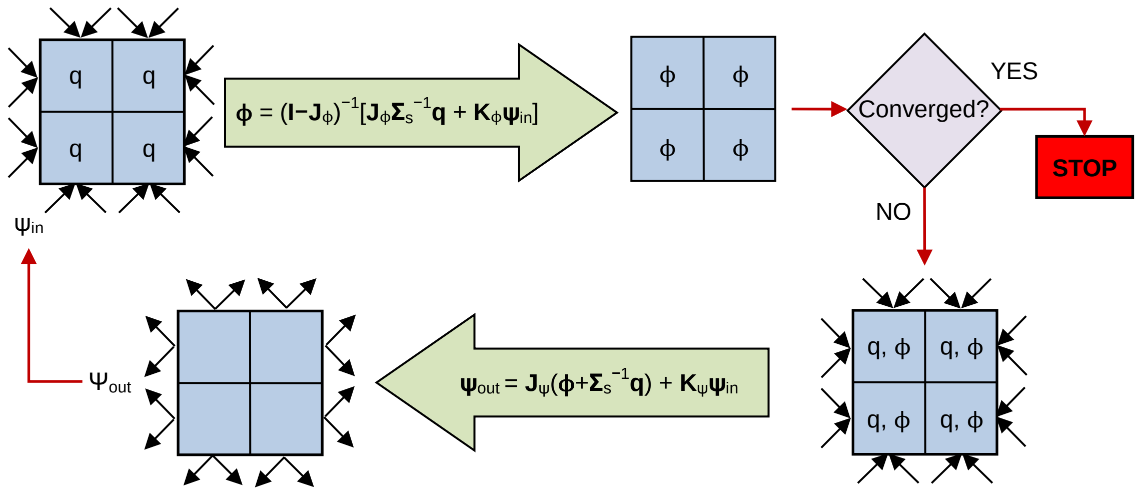

q
q
q
q
$\phi = (I-J_\phi)^{-1}[J_\phi\Sigma_s^{-1}q + K_\phi\psi_{in}]$
ϕ
ϕ
ϕ
ϕ
Converged?
YES
STOP
NO
ψ_{in}
ψ_{out}
$\psi_{out} = J_\psi(\phi+\Sigma_s^{-1}q) + K_\psi\psi_{in}$
q, ϕ
q, ϕ
q, ϕ
q, ϕ

Fig. 3.5.2 Schematic of the PSD algorithm.

3.5.1 Global Solution Methods

The global solution process described above is representative of an iterative approach to invert a large, blocked, sparse matrix system of linear equations. The right hand side is composed of fixed source and boundary condition terms; the solution vector is composed of the cell average scalar fluxes and outgoing angular fluxes at sub-domain boundaries; and the coefficient matrix is composed of the ITMM operators. Because the problem is decomposed, the matrix and vectors are blocked, grouped by sub-domain.

As an illustration, consider the 1-D problem in Fig. 3.5.3. Two sub-domains share a single interface and have fixed boundary conditions on opposing sides. The solution vector is composed of the scalar flux sub-vectors and the outgoing/incoming angular fluxes exchanged on the common interface. The terms are colored to illustrate the distributed ownership, and the superscripts "+" and "−" refer to the particle flow in the positive and negative directions, respectively. Note the similar superscripts applied to the ITMM operators in Eq. (3.5.1), indicating direction-specific portions of the full matrix.

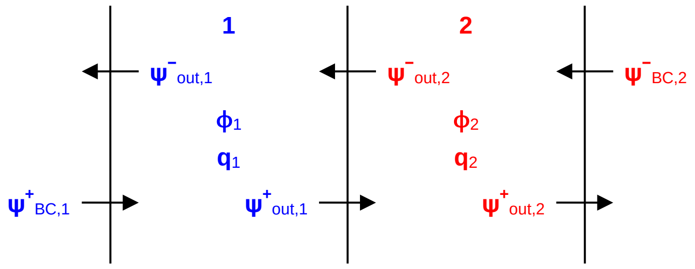


Fig. 3.5.3 Example problem to illustrate the global problem's system of equations.

The system of equations can then be written for the global problem,

$$\begin{bmatrix} (\boldsymbol{I}-\boldsymbol{J}_{\phi 1}) & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\boldsymbol{K}_{\phi 1}^{-} & \mathbf{0} \\ \mathbf{0} & (\boldsymbol{I}-\boldsymbol{J}_{\phi 2}) & \mathbf{0} & -\boldsymbol{K}_{\phi 2}^{+} & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{J}_{\psi 1}^{-} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & -\boldsymbol{K}_{\psi 1}^{-} & \mathbf{0} \\ -\boldsymbol{J}_{\psi 1}^{+} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{J}_{\psi 2}^{-} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{J}_{\psi 2}^{+} & \mathbf{0} & -\boldsymbol{K}_{\psi 2}^{+} & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi}_1 \\ \boldsymbol{\phi}_2 \\ \boldsymbol{\psi}_{out,1}^{-} \\ \boldsymbol{\psi}_{out,1}^{+} \\ \boldsymbol{\psi}_{out,2}^{-} \\ \boldsymbol{\psi}_{out,2}^{+} \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_{\phi 1}\boldsymbol{\Sigma}_{s,1}^{-1}\boldsymbol{q}_1 + \boldsymbol{K}_{\phi 1}^{+}\boldsymbol{\psi}_{BC,1}^{+} \\ \boldsymbol{J}_{\phi 2}\boldsymbol{\Sigma}_{s,2}^{-1}\boldsymbol{q}_2 + \boldsymbol{K}_{\phi 2}^{-}\boldsymbol{\psi}_{BC,2}^{-} \\ \boldsymbol{J}_{\psi 1}^{-}\boldsymbol{\Sigma}_{s,1}^{-1}\boldsymbol{q}_1 \\ \boldsymbol{J}_{\psi 1}^{+}\boldsymbol{\Sigma}_{s,1}^{-1}\boldsymbol{q}_1 + \boldsymbol{K}_{\psi 1}^{+}\boldsymbol{\psi}_{BC,1}^{+} \\ \boldsymbol{J}_{\psi 2}^{-}\boldsymbol{\Sigma}_{s,2}^{-1}\boldsymbol{q}_2 + \boldsymbol{K}_{\psi 2}^{-}\boldsymbol{\psi}_{BC,2}^{-} \\ \boldsymbol{J}_{\psi 2}^{+}\boldsymbol{\Sigma}_{s,2}^{-1}\boldsymbol{q}_2 \end{bmatrix}, \quad (3.5.1)$$

where the subscript "BC" refers to the fixed, non-iterated, boundary condition. The parallel solution of the system relies on the exchange of angular flux at the shared boundary, i.e. $\boldsymbol{\psi}_{out,1}^{+}$ and $\boldsymbol{\psi}_{out,2}^{-}$. After the exchange of information, outgoing angular fluxes become incoming angular fluxes to be operated on by $\boldsymbol{K}_{\phi}$, $\boldsymbol{J}_{\psi}$, and $\boldsymbol{K}_{\psi}$.

Viewing the global problem in this more formal manner allows us to consider efficient means of solving it. Due to the large size of the global coefficient matrix, direct solution techniques have not been considered. Iterative methods are far more suitable for the global solution. Three methods have been employed and tested for this project: parallel block Jacobi, parallel block red-black Gauss-Seidel, and parallel generalized minimal residual with restarts. All three methods are asynchronous, changing the amount and order of work with varying decomposition of the global domain.

The parallel block Jacobi (PBJ) solution algorithm is the direct implementation of the method already outlined. Knowing the previous iterate of the solution vector, the global coefficient matrix acts on the values of the outgoing (incoming, once communicated) angular flux and scalar flux. Because the global matrix is sparse, each processor has the necessary matrix elements—the ITMM operators—and via communication with neighboring sub-domains receives the needed sub-vectors of the total solution vector. Thus, the operator action is represented by Eqs. (3.1.12) and (3.1.14). Furthermore, because all processes use information only from the previous iterate of $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_{out}$ to compute a new iterate the process is a block Jacobi fixed-point iteration.

The most straightforward improvement to the Jacobi iterative method is the Gauss-Seidel iterative method, wherein one uses the most current information when computing new values. This partially occurs in the PBJ algorithm when sub-domains use the new scalar flux value in the computation of outgoing angular fluxes. However, the global problem is solved with iterations on the angular flux; the scalar flux merely serves as a convenient, intermediate variable. In the PBJ algorithm the angular flux always lags because no improved information exists until communication occurs. The parallel red-black Gauss-Seidel (PGS) algorithm can improve on the number of iterations by applying an alternating red-black color-set to the sub-domains. During a PGS iteration, the red sub-domains solve for $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_{out}$ using information from the previous black iteration. This first half of the iteration is followed by sending outgoing angular fluxes to black sub-domains. The PGS iteration is completed when black sub-domains use the new incoming angular flux to compute their own $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_{out}$.

A clear drawback to direct implementation of PGS is processor idleness. By limiting one sub-domain per processor, the PGS algorithm will have half the processors idle during the entirety of the solution phase as black (red) processors wait for red (black) processors to individually evaluate Eqs. (3.1.12) and (3.1.14). To avoid this problem we give processors ownership of multiple sub-domains, as illustrated for 2D configurations in Fig. 3.5.4. In the figure the global problem is bound by the blue box and each processor has two red and two black sub-domains. Angular fluxes at sub-domain boundaries are exchanged either via message-passing within the blue box or via vector copying within individual processor boxes. Processors are assigned at least two sub-domains per dimension to eliminate idleness when alternating between red and black halves of the iteration.

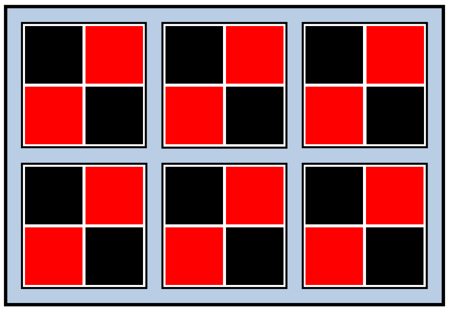


Figure 3.5.4 2D Illustration of the red-black color set used when assigning multiple sub-domains per processor in the PGS scheme.

For a given problem size, implementing the PGS strategy requires splitting a single sub-domain into multiple ones. To start the problem each processor loops over all the sub-domains they own to construct ITMM operators for each. During the solution phase, processors loop over red sub-domains to solve for $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_{out}$, send (or copy) $\boldsymbol{\psi}_{out}$ to neighboring processors in the topology (or to owned black sub-domains), receive incoming angular fluxes for the black sub-domains, and compute $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_{out}$ for black sub-domains.

Fortunately the effect of additional matrix operations involved in looping over all sub-domains during construction and *per* iteration is compensated by the super-linear reduction in size of the ITMM operators as discussed before. Because the operators' sizes decrease at a faster rate than sub-domains can be added, dividing a sub-domain of fixed size (number of cells) into progressively smaller sub-domains will result in shorter computational time devoted to constructing all the ITMM operators and performing the matrix-vector multiplications associated with computing $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_{out}$ per iteration.

On the other hand, numerical tests indicate that increasing the number of sub-domains for a given problem is often accompanied by an increase in the number of iterations to reach convergence, and at best does not require more iterations. Because the Gauss-Seidel method has approximately twice the convergence rate as the Jacobi method, the PGS algorithm mitigates the effect of increasing number of iterations—i.e. compared to having the same number of sub-domains and employing PBJ. In general assigning multiple sub-domains per processor involves competing effects on the total execution time: reduced cumulative work during construction and per iteration and additional work due to increased number of iterations. The PGS algorithm is the most readily available technique to take advantage of the former while addressing the consequences of the latter. Moreover, the optimal number of sub-domains per processor is problem-dependent as these tradeoffs are affected by all elements that influence the number of iterations, including the number of processors, problem size, optical thickness, and scattering ratio.

The multiple sub-domains per processor scheme can be further improved by accelerating the PGS algorithm with a parallel block successive over-relaxation (PSOR) algorithm. However, because any changes to the number of sub-domains results in a new problem, a generic SOR weight cannot be assigned that is optimal for all problems or even guarantees improved spectral properties compared to PGS. Analysis necessary to develop a formula that specifies the optimal SOR weight as a function of relevant problem parameters has not been completed in this project. Therefore, the PSOR algorithm has not been fully tested and used for rigorous comparison with parallel SI as PBJ and PGS have.

Solving the set of equations representing the global problem—exemplified by Eq. (3.5.1)—is not limited to the stationary iterative methods just described; a Krylov subspace solver should be applicable to these problems. Absent the necessary analysis, the global system of equations cannot be generically characterized as positive definite. Furthermore, an algorithm to make the coefficient matrix symmetric has not been identified. As a result the most adequate Krylov solver for general problems is the GMRES algorithm. Although full description and analysis of the GMRES algorithm is outside the scope of this project, a brief description of GMRES and its parallel implementation for the global problem shall be provided. Commented pseudocode for the GMRES method is provided in Algorithm 3.5.1. The reader may also wish to consult References [21] and [31] for complete derivation, description, and convergence analysis of GMRES.

Consider a linear system of equations with a coefficient matrix $\boldsymbol{A}$, solution $\boldsymbol{x}$, RHS $\boldsymbol{b}$, and residual $\boldsymbol{r}$:

$$\begin{gathered}\boldsymbol{Ax} = \boldsymbol{b},\\ \boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}.\end{gathered} \tag{3.5.2}$$

Algorithm 3.5.1 GMRES(m) method with classical Gram-Schmidt orthogonalization.

```
1.  $\boldsymbol{x}^{(0)} \equiv \boldsymbol{x}_0 =$ initial guess
2.  $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
3.  for $k = 0, 1, \dots$ do
4.      $\boldsymbol{v}_1 = \boldsymbol{r}/\|\boldsymbol{r}\|_2$                      {first column of basis}
5.      $\bar{\boldsymbol{s}} = \|\boldsymbol{r}\|_2\, \boldsymbol{e}_1$                    {map norm to basis vector}
6.      for $l = 1, 2, \dots$ do
7.          for $j = 1, 2, \dots, l$ do            {Gram-Schmidt orthogonalization}
8.              $h_{jl} = \langle \boldsymbol{A}\boldsymbol{v}_l, \boldsymbol{v}_j \rangle$                          {inner product}
9.          end for
10.         for $j = 1, 2, \dots, l$ do
11.             $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{v}_l - h_{jl}\boldsymbol{v}_j$
12.         end for
13.         $h_{l+1,l} = \|\boldsymbol{w}\|_2$                    {sub-diagonal element results
14.         $\boldsymbol{v}_{l+1} = \boldsymbol{w}/h_{l+1,l}$                        in Hessenberg matrix}
15.         Use Givens rotation to annihilate sub-diagonal element of $\boldsymbol{h}_l$
16.         Apply rotation to $\bar{\boldsymbol{s}}$                              {Transform least
17.         $|s_{l+1}|$ is equivalent to $\|\boldsymbol{r}\|_2$                       squares problem}
18.         if $|s_{l+1}| \leq$ convergence criterion
19.             Neglect row $l+1$ of $\bar{\boldsymbol{H}}$ and $\bar{\boldsymbol{s}} \rightarrow \boldsymbol{H}$ and $\boldsymbol{s}$
20.             Solve upper triangular $\boldsymbol{H}\boldsymbol{y} = \boldsymbol{s}$           {$\boldsymbol{y}$ that minimizes $\|\boldsymbol{r}\|_2$}
21.             $\boldsymbol{x}^{(km+l)} = \boldsymbol{x}^{(km)} + y_1\boldsymbol{v}_1 + \cdots + y_l\boldsymbol{v}_l$
22.             exit
23.         end if
24.     end for
25.     if $|s_{m+1}| \leq$ convergence criterion: exit
26.     Neglect row $m+1$ of $\bar{\boldsymbol{H}}$ and $\bar{\boldsymbol{s}} \rightarrow \boldsymbol{H}$ and $\boldsymbol{s}$
27.     Solve upper triangular $\boldsymbol{H}\boldsymbol{y} = \boldsymbol{s}$
28.     $\boldsymbol{x}^{((k+1)m)} = \boldsymbol{x}^{(km)} + y_1\boldsymbol{v}_1 + \cdots + y_m\boldsymbol{v}_m$
29.     $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{((k+1)m)}$
```

The GMRES method begins with an initial guess $\boldsymbol{x}^{(0)}$ and corresponding residual $\boldsymbol{r}^{(0)}$. Often, a suitable initial guess of the solution is simply zero, making the residual equal to $\boldsymbol{b}$.

The GMRES method employs the Arnoldi method (lines 4–14 of the algorithm) to build a basis $\boldsymbol{V}_l$ with l orthonormal columns and a corresponding Hessenberg matrix $\bar{\boldsymbol{H}}_l$ for the Krylov subspace $\mathrm{span}\{\boldsymbol{r}^{(0)}, \boldsymbol{A}\boldsymbol{r}^{(0)}, \boldsymbol{A}^2\boldsymbol{r}^{(0)}, \dots, \boldsymbol{A}^{l-1}\boldsymbol{r}^{(0)}\}$ and results in the relation

$$\boldsymbol{A}\boldsymbol{V}_l = \boldsymbol{V}_{l+1}\bar{\boldsymbol{H}}_l. \tag{3.5.3}$$

Krylov subspace methods are a broad class of projection methods that seek solutions of the form

$$\boldsymbol{x}^{(l)} = \boldsymbol{x}^{(0)} + y_1\boldsymbol{v}_1 + \cdots + y_l\boldsymbol{v}_l, \tag{3.5.4}$$

where the subscript is an index of an element of a yet undefined l-vector $\boldsymbol{y}$ or of a column of $\boldsymbol{V}_l$. Accordingly, at line 4 of Algorithm 3.5.1, the first column of the basis is simply set to the normalized residual vector. Using Eqs. (3.5.3) and (3.5.4), one can re-express the residual,

$$\begin{aligned} \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(l)} &= \boldsymbol{b} - \boldsymbol{A}\left(\boldsymbol{x}^{(0)} + \boldsymbol{V}_l\boldsymbol{y}\right) \\ &= \boldsymbol{r}^{(0)} - \boldsymbol{A}\boldsymbol{V}_l\boldsymbol{y} \\ &= \|\boldsymbol{r}\|_2\boldsymbol{v}_1 - \boldsymbol{V}_{l+1}\overline{\boldsymbol{H}}_l\boldsymbol{y} \\ &= \boldsymbol{V}_{l+1}(\|\boldsymbol{r}\|_2\boldsymbol{e}_1 - \overline{\boldsymbol{H}}_l\boldsymbol{y}). \end{aligned} \tag{3.5.5}$$

At line 5 of Algorithm 3.5.1, the first parenthetical term of the final relation is stored in the vector $\overline{\boldsymbol{s}}$.

Orthogonalization techniques for the Arnoldi method include classical Gram-Schmidt, modified Gram-Schmidt, and Householder transformations. Although classical Gram-Schmidt orthogonalization is known to be prone to cancellation errors in finite arithmetic, it has favorable properties related to parallelism. Moreover, test cases have not shown problems arising from cancellation, which would typically require a costly reorthogonalization. Therefore the Arnoldi process in Algorithm 3.5.1 is classical Gram-Schmidt without reorthogonalization.

Evident from Eq. (3.5.5) $\boldsymbol{y}$ should be chosen in a manner that minimizes $\|\overline{\boldsymbol{s}} - \overline{\boldsymbol{H}}_l\boldsymbol{y}\|_2$, thereby also minimizing $\|\boldsymbol{r}\|_2$. Because $\overline{\boldsymbol{H}}_l$ is a Hessenberg matrix, the minimization of $\|\overline{\boldsymbol{s}} - \overline{\boldsymbol{H}}_l\boldsymbol{y}\|_2$ with $\boldsymbol{y}$ is a least-squares problem. To solve the least-squares problem, Givens rotations are used to annihilate the $\overline{\boldsymbol{H}}_l$ sub-diagonal, modifying $\overline{\boldsymbol{s}}$ in the process. Ignoring the last row leaves an upper triangular system of equations $\boldsymbol{H}\boldsymbol{y} = \boldsymbol{s}$ to be solved with backward substitution. However, most importantly, as shown in Reference [21] the last element of $\boldsymbol{s}$ is equivalent to the residual norm, and therefore it is used to check convergence before the solution is updated. This process accounts for lines 15–20 of Algorithm 3.5.1.

Krylov-subspace methods are known to produce the exact solution, assuming exact arithmetic, in n steps, where n is the length of $\boldsymbol{x}$. However, sufficiently accurate approximate solutions are often computed in far fewer steps. A convergence criterion is applied to determine a solution's adequacy, and as a result the GMRES algorithm serves as an efficient iterative solution method. Moreover, because storing all basis vectors $\boldsymbol{v}_l$ can become prohibitively large, the algorithm is typically *restarted* every m steps. Evident in Algorithm 3.5.1, the restarted algorithm, GMRES(m), begins building a new basis every m steps, beginning with the current $\boldsymbol{r}$.

The GMRES(m) method was parallelized and applied to the global problem previously described and illustrated simply by Figure 3.5.3 and Eq. (3.5.1). The solution process is outlined in Algorithm 3.5.2. The full global coefficient matrix $\boldsymbol{A}$ is not constructed, but its nonzero entries are sub-matrices distributed among all the participating processes, i.e. the ITMM operators. Further, the solution vector $\boldsymbol{x}$ is divided into sub-vectors and distributed as $\boldsymbol{\phi}$ and $\boldsymbol{\psi}_{out}$.

Algorithm 3.5.2 PGMRES(m) method for solving the ITMM global problem.

```
1.  Set local $\boldsymbol{x}^{(0)} \equiv \boldsymbol{x}_0 =$ initial guess
2.  Compute local $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
3.  Compute local $\|\boldsymbol{r}\|_2$
4.  **all_reduce_sum** of $\|\boldsymbol{r}\|_2$                      {parallel reduction and broadcast}
5.  **for** $k = 0, 1, \dots$ **do**
6.      $\boldsymbol{v}_1 = \boldsymbol{r}/\|\boldsymbol{r}\|_2$                                {$\boldsymbol{V}$ is distributed}
7.      $\bar{\boldsymbol{s}} = \|\boldsymbol{r}\|_2\, \boldsymbol{e}_1$                          {each process has full $\bar{\boldsymbol{s}}$}
8.      **for** $l = 1, 2, \dots$ **do**
9.          **communicate** $\boldsymbol{v}_{l,\psi,out} \rightarrow \boldsymbol{v}_{l,\psi,in}$            {send/receive vector}
10.         Compute local $\boldsymbol{A}\boldsymbol{v}_l \rightarrow \boldsymbol{w}$           {ITMM operators $\times$ flux vectors}
11.         **for** $j = 1, 2, \dots, l$ **do**                          {$\boldsymbol{w}$ is distributed}
12.             $h_{jl} = \langle \boldsymbol{w}, \boldsymbol{v}_j \rangle$                           {inner product}
13.         **end for**
14.         **all_reduce_sum** of $\boldsymbol{h}_l$                 {All processes will have full $\bar{\boldsymbol{H}}$}
15.         **for** $j = 1, 2, \dots, l$ **do**
16.             $\boldsymbol{w} = \boldsymbol{w} - h_{jl}\boldsymbol{v}_j$
17.         **end for**
18.         Compute local $\|\boldsymbol{w}\|_2$
19.         **all_reduce_sum** of $\|\boldsymbol{w}\|_2$
20.         $h_{l+1,l} = \|\boldsymbol{w}\|_2$
21.         $\boldsymbol{v}_{l+1} = \boldsymbol{w}/h_{l+1,l}$
22.         Every process has full $\bar{\boldsymbol{H}}$; all apply Givens rotations
23.         Apply rotations to $\bar{\boldsymbol{s}}$
24.         $|s_{l+1}|$ is equivalent to $\|\boldsymbol{r}\|_2$            {all processes have same value}
25.         **if** $|s_{l+1}| \leq$ convergence criterion
26.             Neglect row $l + 1$ of $\bar{\boldsymbol{H}}$ and $\bar{\boldsymbol{s}} \rightarrow \boldsymbol{H}$ and $\boldsymbol{s}$
27.             Solve upper triangular $\boldsymbol{H}\boldsymbol{y} = \boldsymbol{s}$              {$\boldsymbol{y}$ that minimizes $\|\boldsymbol{r}\|_2$}
28.             Compute local $\boldsymbol{x}^{(km+l)} = \boldsymbol{x}^{(km)} + y_1\boldsymbol{v}_1 + \cdots + y_l\boldsymbol{v}_l$
29.             **exit**
30.         **end if**
31.     **end for**
32.     **if** $|s_{m+1}| \leq$ convergence criterion: **exit**
33.     Neglect row $m + 1$ of $\bar{\boldsymbol{H}}$ and $\bar{\boldsymbol{s}} \rightarrow \boldsymbol{H}$ and $\boldsymbol{s}$
34.     Solve upper triangular $\boldsymbol{H}\boldsymbol{y} = \boldsymbol{s}$
35.     Compute local $\boldsymbol{x}^{((k+1)m)} = \boldsymbol{x}^{(km)} + y_1\boldsymbol{v}_1 + \cdots + y_m\boldsymbol{v}_m$
36.     **communicate** $\boldsymbol{\psi}_{out} \rightarrow \boldsymbol{\psi}_{in}$
37.     Compute local $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{((k+1)m)}$
```

The parallel GMRES(m) method, PGMRES(m), relies on the distributed nature of the ITMM operators and flux vectors to perform the matrix-vector multiplications. Because of this data distribution, the columns of $\boldsymbol{V}$ and the vector $\boldsymbol{w}$ will similarly be distributed. In Algorithm 3.5.2 the initial guess is set to zero, which was the only guess employed in test problems. Because the residual is distributed, computing the 2-norm is a two-step process: first each process computes the local contribution and, second a global all-reduction is used to sum all the pieces and broadcast the result to all processors (lines 3–4). The matrix-vector multiplication of line 10 requires the incoming data at the boundaries, i.e. the outgoing data from adjacent sub-domains. Therefore, at line 9 sub-vectors of the current $\boldsymbol{V}$ column are exchanged just as angular flux vectors were exchanged in the stationary methods.

The Arnoldi process, lines 10–21, is performed in parallel with several global communications necessary for completion. With locally owned sub-vectors of $\boldsymbol{w}$ and $\boldsymbol{v}_l$ each process can compute its own contribution to elements of $\overline{\boldsymbol{H}}$. Another global all-reduction sums/broadcasts the elements of the column vector $\boldsymbol{h}_l$. Sub-vectors of $\boldsymbol{w}$ are updated locally and another global all-reduction is needed to compute its 2-norm and subsequently h_{l+1} , and $\boldsymbol{v}_{l+1}$. Orthogonalization with the modified Gram-Schmidt technique has also been implemented but requires costly global all-reductions for each individual element of $\overline{\boldsymbol{H}}$, making it unsuitable for problems meant for massively parallel architectures. Classical Gram-Schmidt orthogonalization allows for a less expensive all-reduction of a vector. The benefit of giving each processor the entire $\overline{\boldsymbol{H}}$ matrix becomes apparent when solving the least squares problem, lines 22–27. Each process can simultaneously check the global residual norm for convergence, solve for $\boldsymbol{y}$, and update its part of $\boldsymbol{x}$ without communication. Alternative algorithms have been developed for parallelizing the Arnoldi process and solving the least squares problem. However, the method in Algorithm 3.5.2 is a straightforward approach that introduces a modest number of additional global communications compared to the stationary iterative approaches.

A more dominant factor in the performance of the PGMRES(m) method is implementation and effectiveness of a preconditioner. Preconditioners are designed to improve the spectral properties of $\boldsymbol{A}$, making the system converge in fewer iterations. A preconditioner applied to Algorithm 3.5.2 would appear in lines 2, 10, and 37, where in each case the inverse of the preconditioner would be (left) multiplied with the RHS shown in the algorithm. The only preconditioner investigated for this work was the inverse diagonal block preconditioner: a diagonal block matrix where each block is the inverse of the corresponding diagonal block of $\boldsymbol{A}$. Examining Eq. 3.5.1, in the case of the ITMM global problem, this preconditioner is the inverse of $\boldsymbol{J}_\phi$ for the scalar flux equations and the identity matrix $\boldsymbol{I}$ for the angular flux equations. Additional costs associated with computing and storing the inverse of $\boldsymbol{J}_\phi$ are considerable but provide a simple improvement to the method. Less costly and/or more effective preconditioners may exist but have not been explored so far.

3.5.2 Local Solution Methods

Solving the global problem involves iterating on the sub-domains' boundary angular fluxes. Solving the local problem involves computing these angular flux values along outgoing directions on each of the subdomain boundaries. Because the scalar flux is a useful value that aides the computational process, it is additionally computed locally as an intermediate variable during each global iteration. Considering Eq. (3.1.14), solving for an updated outward angular flux at a sub-domain's boundaries comprises a set of matrix-vector multiplications involving a previous iterate of the cell-averaged scalar fluxes, distributed source, and incoming angular flux at the sub-domain's boundaries. Programming techniques that seek to optimize these necessary multiplication operations have been evaluated.

In contrast computing a new iterate of the scalar flux per Eq. (3.1.12) requires the solution of a system of equations for which many methods are available. Consequently, this section is dedicated primarily to the discussion of computing new iterates of the scalar flux. Specifically, each local system of scalar flux equations may be solved with direct inversion of $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$, with factorization of $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$, or with stationary or non-stationary iterative methods.

Direct inversion of $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$, and repeated reuse of the inverse operator in all global iterations, is the most straightforward approach to solving Eq. (3.1.12). It shifts a large fraction of computational burden from work performed per global iteration to work performed during ITMM operator construction. Recall that the $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ is an $N \times N$ matrix, where N is the number of computational cells. The act of inverting this matrix requires computational effort of order N^3. To further reduce the work per iteration, after inversion the product $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)^{-1}\boldsymbol{K}_{\phi}$ is computed and stored in $\boldsymbol{K}_{\phi}$. Because the dimensions of $\boldsymbol{K}_{\phi}$ depend on N and N_t, the number of angles, the matrix multiplications are performed at a cost that grows like $N^{8/3}$ and like N_t. About an additional N^2 multiplications and additions are performed per global iteration to compute the first RHS term of Eq. (3.1.12). Similarly computing the second RHS term has costs that grow like $N^{5/3}$ and N_t operations. Due to the size of the vectors and the dense nature of the matrices, the work per iteration cannot be reduced. However, the prohibitively large costs associated with computing $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)^{-1}$ and $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)^{-1}\boldsymbol{K}_{\phi}$ make this method a poor choice for solving for the scalar flux within each subdomain per global iteration.

Alternatively the $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ matrix can be factorized at a considerable reduction in ITMM operator construction time compared to direct inversion. Decomposing a matrix $\boldsymbol{A}$ into lower $\boldsymbol{L}$ and upper $\boldsymbol{U}$ components ($\boldsymbol{A} \equiv \boldsymbol{LU}$) can be done via Gaussian elimination at approximately one-third the cost to invert it. Cholesky ($\boldsymbol{LL}^T$) factorization can be applied with half the amount of work as $\boldsymbol{LU}$ factorization if the matrix is symmetric—or is made symmetric as discussed in [19]—and positive definite. Although the cost of factorizing $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ grows rapidly with N, sub-domain sizes are typically kept small due to limitations on available memory, and this method avoids the more costly $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)^{-1}\boldsymbol{K}_{\phi}$ multiplication. The work to solve the factorized system is similar to that of the inverted matrix system. In Eq. (3.1.12) a RHS source term can be computed from the distributed source and incoming boundary angular fluxes (known after communication), leaving $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ operating on the solution vector on the LHS of the relation.

Stationary iterative methods are also applicable to the solution of the local scalar flux. Again $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ is a LHS coefficient matrix and a RHS source vector is computed from known quantities. The Jacobi, Gauss-Seidel, and SOR iterative methods [21] and [32] can be used to solve the linear system. Without an analytic formula to determine the optimum SOR weight the method relies on relaxation weights chosen empirically. Because each sub-domain is different, one should not expect the optimal relaxation weight to be identical for all sub-domains. However, as currently implemented, all sub-domains are given the same SOR weight for local solution.

Recall that $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ can be made symmetric easily using readily available factors: cell volume, σ_s, λ (AHOT-N order), and l (anisotropic scattering order). Moreover assume that $\left(\boldsymbol{I}-\boldsymbol{J}_{\phi}\right)$ is positive definite. These two characteristics imply that the conjugate gradient (CG) method can be used to solve Eq. (3.1.12). The CG method can be derived from the Lanczos method for symmetric matrices. Like the Arnoldi method, the Lanczos method builds an orthogonal basis for the Krylov subspace of a matrix $\boldsymbol{A}$. Whereas the Arnoldi method requires all previous columns of the basis $\boldsymbol{V}$, the Lanczos method builds the basis with a three-term recurrence. Complete derivations of the Lanczos and CG methods are provided in Reference [21].

For a generic $\boldsymbol{Ax} = \boldsymbol{b}$ system the CG method is outlined in Algorithm 3.5.3. The method is posed as an iterative procedure, although the system is solved exactly in n steps, where n is the length of $\boldsymbol{x}$, similar to GMRES. The Krylov subspace in the algorithm is built by the vectors $\boldsymbol{p}$. The $\boldsymbol{p}$-vectors are called search directions. The algorithm begins by initializing the solution, residual, and current search direction (lines 1–3). A scalar search parameter α is computed at line 5 that ensures orthogonality among residual vectors. Using this parameter and the current search direction, $\boldsymbol{x}$ and $\boldsymbol{r}$ are updated (lines 6–7). A second scalar parameter β, related to the three-term recurrence of the $\boldsymbol{p}$-vectors, is computed and subsequently used to compute the new search direction (lines 8–9).

Algorithm 3.5.3 Conjugate Gradient Method.

```
1.   $\boldsymbol{x}^{(0)} \equiv \boldsymbol{x}_0 = \text{initial guess}$
2.   $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
3.   $\boldsymbol{p}_0 = \boldsymbol{r}^{(0)}$
4.   **for** $l = 0, 1, \ldots$ **do**
5.       $\alpha_l = \langle \boldsymbol{r}^{(l)}, \boldsymbol{r}^{(l)} \rangle / \langle \boldsymbol{A}\boldsymbol{p}_l, \boldsymbol{p}_l \rangle$
6.       $\boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} + \alpha_l \boldsymbol{p}_l$
7.       $\boldsymbol{r}^{(l+1)} = \boldsymbol{r}^{(l)} + \alpha_l \boldsymbol{A}\boldsymbol{p}_l$
8.       $\beta_l = \langle \boldsymbol{r}^{(l+1)}, \boldsymbol{r}^{(l+1)} \rangle / \langle \boldsymbol{r}^{(l)}, \boldsymbol{r}^{(l)} \rangle$
9.       $\boldsymbol{p}_{l+1} = \boldsymbol{r}^{(l+1)} + \beta_l \boldsymbol{p}_l$
10.  **end for**
```

The iterative convergence speed of CG method is improved via the application of a preconditioner. In Algorithm 3.5.3 a preconditioner is employed at lines 3 and 9, multiplying with the residual vector. For this project only preconditioners based on the stationary iterative methods were investigated because they increase the storage requirement only by a relatively small amount—a single vector of equal length to each sub-domain's scalar flux vector—and because they pose a relatively low cost to compute during ITMM operator construction and to apply in each CG iteration. Because $(\boldsymbol{I} - \boldsymbol{J}_\phi)$ is made symmetric, it can be decomposed into

$$\boldsymbol{I} - \boldsymbol{J}_\phi = \boldsymbol{D} + \boldsymbol{L} + \boldsymbol{L}^T. \tag{3.5.6}$$

Three preconditioners $\boldsymbol{M}$ based on this decomposition are the Jacobi (jac), symmetric Gauss-Seidel (sgs), and symmetric SOR (sor) preconditioners:

$$\begin{aligned} \boldsymbol{M}_{jac} &= \boldsymbol{D}, \\ \boldsymbol{M}_{sgs} &= (\boldsymbol{D} + \boldsymbol{L})\boldsymbol{D}^{-1}(\boldsymbol{D} + \boldsymbol{L})^T, and \\ \boldsymbol{M}_{sor} &= \frac{1}{2-\omega}\left(\frac{1}{\omega}\boldsymbol{D} + \boldsymbol{L}\right)\left(\frac{1}{\omega}\boldsymbol{D}\right)^{-1}\left(\frac{1}{\omega}\boldsymbol{D} + \boldsymbol{L}\right)^T. \end{aligned} \tag{3.5.7}$$

An expression to compute an optimal SOR weight factor ω per sub-domain has not been formulated to date.

The number of operations to solve for the local scalar flux solution per local iteration were comparable to the cost to solve the pre-factorized system of equations, growing like N^2. Because the $(\boldsymbol{I}-\boldsymbol{J}_{\phi})$ matrix is generally full, operations cannot be spared as they may be if the matrix was sparse. As a result the direct method with stored factorization of $(\boldsymbol{I}-\boldsymbol{J}_{\phi})$ is competitive in the measure of execution time per global iteration.

For all iterative methods described, computing the local scalar flux to within the convergence criterion may not be necessary. In fact computing locally converged scalar flux with poor estimates of the correct local boundary conditions, especially for early global iterations, can be an expensive waste of computer resources. Fortunately, the number of local iterations per global iteration is easily controlled. However, fixing the number of local iterations has been found to increase the total number of global iterations in many cases. This is a consequence of using poorer approximations of the scalar flux when computing new iterates of the outgoing angular flux. The direct solution methods produce the exact solution scalar flux according to the current global iterate of the incoming angular flux boundary conditions, thus providing a floor for the number of global iterations. Ultimately the goal of fixing the number of local iterations is to minimize the cumulative number of local iterations consumed over all global iterations.

Lastly local source iterations can be performed per group by sweeping across each sub-domain in all directions in the quadrature set using the group distributed source and incoming angular flux at the sub-domain boundaries from the previous global iteration. This method has been applied by Yavuz and Larsen [7] and Warsa, *et al.* [12]. Like other iterative methods, the number of source iterations performed per global iteration can be limited. This method does not require the ITMM operators, resorting to mesh sweeps whose evasion was a motivating factor for this project. Therefore, this local solution method is noted for completeness but has not been employed in test cases.

3.5.3 Remarks

Research on the PSD approach involved investigation of both local and global solutions. First, within the PBJ global framework, the different local methods were tested on a series of problems to measure the computational costs—execution time and memory. Numerical testing summarized in this report established the fact that solving for the local scalar flux was generally performed most efficiently via the stored factorization technique. When the sub-domains are small in terms of number of cells, the cost to factorize $(\boldsymbol{I}-\boldsymbol{J}_{\phi})$ is small and is overcome by faster per iteration execution time. Second, the various global solution methods were considered. The goals of the varying methods were to reduce, relative to PBJ base cases, the ITMM operator construction time, the solution time per iteration, and the number of iterations.

3.6 Task 6 – Extend operators to anisotropic scattering

Thus far only isotropic scattering has been considered, although most real-world problems must account for some degree of anisotropy in the scattering source. The ITMM formalism has been extended to permit anisotropic scattering. Additional terms will be added to the RHS of Eq. (3.1.4) that correspond to angular moments of the flux accompanying a polynomial expansion of the scattering cross section. The flow of the differential mesh sweep will not change, but the ITMM operators and the $\boldsymbol{\phi}$ solution vector will become larger to account for the additional unknowns.

In this section, the ITMM operators first will be extended to anisotropic scattering for the general 3-D case to demonstrate how the expansion affects the system of equations per cell and thus the construction of the ITMM operators. This expansion will be performed with the standard spherical harmonics approach. The equations defining the anisotropic scattering source for 1-D and 2-D geometries are simplifications of the 3-D case. Subsequently, the new scattering source will be substituted into the per-cell system of equations such that a new $\boldsymbol{\Gamma}$ matrix can be formed and utilized in the differential mesh sweep. Then parallel performance results for test cases with anisotropic scattering will be presented.

3.6.1 The General Three-Dimensional Case

To begin the derivation of the 3-D within-group equations with anisotropic scattering, first reconsider the expansion,

$$q_s(\hat{\Omega}) = \sum_{l=0}^{\infty} \frac{(2l+1)}{4\pi} \sigma_{sl} \int_{4\pi} d\hat{\Omega}'\, P_l(\hat{\Omega}' \cdot \hat{\Omega}) \psi(\hat{\Omega}'). \tag{3.6.1}$$

The temporal variable is neglected for the steady-state case and the spatial and energy variables are suppressed for brevity. The $P_l(\hat{\Omega}' \cdot \hat{\Omega})$ function can be rewritten in terms use of the polar directional cosine μ and the azimuthal angle ω, [25]

$$P_l(\hat{\Omega}' \cdot \hat{\Omega}) = P_l(\mu)P_l(\mu') + 2\sum_{m=1}^{l} \frac{(l-m)!}{(l+m)!} P_l^m(\mu) P_l^m(\mu') \cos[m(\omega-\omega')]. \tag{3.6.2}$$

$P_l^m(\mu)$ is known as the associated Legendre function; expressions for the first several (integer) values of l and m are readily available, and recursive relationships are available for higher orders. The cosine term in Eq. (3.6.2) is easily modified using the trigonometric identity,

$$\cos(m\omega - m\omega') = \cos(m\omega)\cos(m\omega') + \sin(m\omega)\sin(m\omega'). \tag{3.6.3}$$

When Eqs. (3.6.2) and (3.6.3) are substituted into Eq. (3.6.1) and the angular integral is decomposed into a double integral form, the scattering source becomes, [25]

$$\begin{aligned} q_s(\hat{\Omega}) &= \sum_{l=0}^{\infty} \frac{(2l+1)}{4\pi} \sigma_{sl} \Bigg\{ P_l(\mu) \int_{-1}^{1} d\mu' \int_0^{2\pi} d\omega'\, \psi(\mu',\omega') P_l(\mu') \\ &+ 2\sum_{m=1}^{l} \frac{(l-m)!}{(l+m)!} \Bigg[P_l^m(\mu)\cos(m\omega) \int_{-1}^{1} d\mu' \int_0^{2\pi} d\omega'\, \psi(\mu',\omega') P_l^m(\mu') \cos(m\omega') \\ &+ P_l^m(\mu)\sin(m\omega) \int_{-1}^{1} d\mu' \int_0^{2\pi} d\omega'\, \psi(\mu',\omega') P_l^m(\mu') \sin(m\omega') \Bigg] \Bigg\}. \end{aligned} \tag{3.6.4}$$

Next, the equivalence between Legendre polynomials and associated Legendre functions for $m = 0$—$P_l = P_l^0$—is applied. The leading factors of the summations are split into two square root factors and moved next to the Legendre functions:

$$\begin{aligned} q_s(\hat{\Omega}) &= \sum_{l=0}^{\infty} \sigma_{sl} \Bigg\{ \sqrt{C_{l0}} P_l^0(\mu) \cos(0\omega) \int_{-1}^{1} d\mu' \int_0^{2\pi} d\omega' \, \psi(\mu',\omega') \sqrt{C_{l0}} P_l^0(\mu) \cos(0\omega') \\ &+ 2 \sum_{m=1}^{l} \Bigg[\sqrt{C_{lm}} P_l^m(\mu) \cos(m\omega) \int_{-1}^{1} d\mu' \int_0^{2\pi} d\omega' \, \psi(\mu',\omega') \sqrt{C_{lm}} P_l^m(\mu') \cos(m\omega') \\ &+ \sqrt{C_{lm}} P_l^m(\mu) \sin(m\omega) \int_{-1}^{1} d\mu' \int_0^{2\pi} d\omega' \, \psi(\mu',\omega') \sqrt{C_{lm}} P_l^m(\mu') \sin(m\omega') \Bigg] \Bigg\}. \end{aligned} \quad (3.6.5)$$

The constants C_{lm} are defined as

$$C_{lm} \equiv \frac{(2l+1)(l-m)!}{4\pi(l+m)!}. \quad (3.6.6)$$

A useful approach is to define even and odd angular moments from even and odd spherical harmonics. Observing the terms in Eq. (3.6.5) and using Eq. (3.6.6), even (e) and odd (o) spherical harmonics are, respectively, [25]

$$Y_{lm}^e(\hat{\Omega}) = Y_{lm}^e(\mu,\omega) = \sqrt{C_{lm}} P_l^m(\mu) \cos(m\omega) \quad (3.6.7.a)$$

and

$$Y_{lm}^o(\hat{\Omega}) = Y_{lm}^o(\mu,\omega) = \sqrt{C_{lm}} P_l^m(\mu) \sin(m\omega). \quad (3.6.7.b)$$

Even angular moments of the flux are defined with Eq. (3.6.7.a),

$$\varphi_l^m = \int_{-1}^{1} d\mu \int_0^{2\pi} d\omega \, Y_{lm}^e(\mu,\omega) \psi(\mu,\omega), \quad (3.6.8.a)$$

and odd angular moments are defined with Eq. (3.6.7.b),

$$\vartheta_l^m = \int_{-1}^{1} d\mu \int_0^{2\pi} d\omega \, Y_{lm}^o(\mu,\omega) \psi(\mu,\omega). \quad (3.6.8.b)$$

Substituting the definitions given by Eqs. (3.6.7) and (3.6.8) into Eq. (3.6.5) finally yields a more compact form for the anisotropic scattering source:

$$q_s = \sum_{l=0}^{\infty} \sigma_{sl} \left\{ Y_{l0}^{e}(\hat{\Omega})\varphi_l^0 + 2\sum_{m=1}^{l} \left[Y_{lm}^{e}(\hat{\Omega})\varphi_l^m + Y_{lm}^{o}(\hat{\Omega})\vartheta_l^m \right] \right\}. \tag{3.6.9}$$

With a slight rearrangement and expanding the *m* summation to include 0, the source is written in the more compact form, which will be employed in formulating the ITMM operators,

$$q_s = \sum_{l=0}^{\infty} \sum_{m=0}^{l} \sigma_{sl}\,(2-\delta_{m0})\left[Y_{lm}^{e}(\hat{\Omega})\varphi_l^m + Y_{lm}^{o}(\hat{\Omega})\vartheta_l^m \right]. \tag{3.6.10}$$

Note, the infinite series in *l* is typically truncated to some small, finite value *L*. The anisotropic scattering source expressed by Eq. (3.6.10) applies to 3-D geometries. The anisotropic scattering source of 2-D geometries can be expressed, due to symmetry, with only the even angular moments of the flux. The term is simplified even further for 1-D geometries, where only an expansion in Legendre polynomials is necessary.

3.6.2 Anisotropic Scattering in the ITMM

With Eq. (3.6.10), the system of equations to solve per cell can be reposed as:

$$\begin{bmatrix} \sigma_{tijk} & |\xi_n|/\Delta z_k & |\eta_n|/\Delta y_j & |\mu_n|/\Delta x_i \\ 1 & -0.5 & 0 & 0 \\ 1 & 0 & -0.5 & 0 \\ 1 & 0 & 0 & -0.5 \end{bmatrix} \begin{bmatrix} \psi_{nijk} \\ \psi_{n,i,j,k_{out}} \\ \psi_{n,i,j_{out},k} \\ \psi_{n,i_{out},j,k} \end{bmatrix} =$$

$$\begin{bmatrix} Y_{00}^e & Y_{10}^e & 2Y_{11}^e & 2Y_{11}^o & \cdots & 2Y_{LL}^o & |\xi_n|/\Delta z_k & |\eta_n|/\Delta y_j & |\mu_n|/\Delta x_i \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} \sigma_{so}\varphi_0^0 + q_{00}^e \\ \sigma_{s1}\varphi_1^0 + q_{10}^e \\ \sigma_{s1}\varphi_1^1 + q_{11}^e \\ \sigma_{s1}\vartheta_1^1 + q_{11}^o \\ \vdots \\ \sigma_{sL}\vartheta_L^L + q_{LL}^o \\ \psi_{n,i,j,k_{in}} \\ \psi_{n,i,j_{in},k} \\ \psi_{n,i_{in},j,k} \end{bmatrix}. \tag{3.6.11}$$

The fixed source is brought into the anisotropic summations with even/odd definitions similar to those presented for the flux. For many problems, distributed sources are frequently assumed isotropic—that is only the q_{00}^e moment is nonzero. However, for multigroup problems, anisotropic downscattering sources will be accumulated at higher energy groups and used as fixed sources for lower energy groups, making such treatment convenient.

The LHS coefficient matrix of Eq. (3.6.11) is always a four-by-four matrix, independent of the order of anisotropy L considered. Therefore, it can be inverted analytically, and the expressions for the sixteen elements can be implemented directly, saving computational time associated with costly matrix inversion. The RHS coefficient matrix is dependent on the order of anisotropy; its size is $4 \times [(L+1)^2 + 3]$. Moreover, inverting the LHS matrix and left-multiplying the inverse with the RHS matrix, results in the anisotropic scattering operator $\boldsymbol{\Gamma}$, also of size $4 \times [(L+1)^2+3]$:

$$\begin{bmatrix} \psi_{nijk} \\ \psi_{n,i,j,k_{out}} \\ \psi_{n,i,j_{out},k} \\ \psi_{n,i_{out},j,k} \end{bmatrix} = \begin{bmatrix} \gamma^{a00e} & \cdots & \gamma^{aLLo} & \gamma^{axy} & \gamma^{axz} & \gamma^{ayz} \\ \gamma^{xy00e} & \cdots & \gamma^{xyLLo} & \gamma^{xyxy} & \gamma^{xyxz} & \gamma^{xyyz} \\ \gamma^{xz00e} & \cdots & \gamma^{xzLLo} & \gamma^{xzxy} & \gamma^{xzxz} & \gamma^{xzyz} \\ \gamma^{yz00e} & \cdots & \gamma^{yzLLo} & \gamma^{yzxy} & \gamma^{yzxz} & \gamma^{yzyz} \end{bmatrix}_{nijk} \begin{bmatrix} \sigma_{so}\varphi_0^0 + q_{00}^e \\ \vdots \\ \sigma_{sL}\vartheta_L^L + q_{LL}^o \\ \psi_{n,i,j,k_{in}} \\ \psi_{n,i,j_{in},k} \\ \psi_{n,i_{in},j,k} \end{bmatrix}. \quad (3.6.12)$$

The operator equations that define the ITMM operators in the differential mesh sweep and the earlier description of that sweep do not change with the inclusion of anisotropic scattering. Instead, the differential mesh sweep is performed in the same way as the isotropic case, except that coupling involves $(L+1)^2$ angular moments ϕ_l^m (decomposed into φ_{lm}^e and ϑ_{lm}^o) instead of just the scalar flux ϕ,

$$\frac{\partial \phi_{i,j,k,l,m}^v}{\partial \phi_{i',j',k',l',m'}^p} = a_{(i,j,k,l,m)(i',j',k',l',m')} c_{i',j',k',l'} \equiv j_{\phi(i,j,k,l,m)(i',j',k',l',m')}. \quad (3.6.13)$$

The scattering ratio $c_{i',j',k',l'}$ has a dependence on the anisotropic order because scattering cross sections are defined for the range of moments l' in 0 to L. In the presence of anisotropic scattering, $\boldsymbol{J}_\phi$ is a $IJK(L+1)^2 \times IJK(L+1)^2$ square matrix.

The self-coupling expression is no longer a scalar. It is represented by the $(L+1)^2$-vector,

$$\frac{\partial \psi_{n,i,j,k}}{\partial \phi_{i',j',k',l',m'}^p} = c_{ijkl}\gamma_{nijk}^{alm}, \quad (3.6.14)$$

Even and odd moments again have been combined for brevity, including in γ elements.

The summation over the angular quadrature must be modified per Eqs. (3.6.8.a) and (3.6.8.b). Modification will result in derivatives of even and odd angular moments of the flux instead of the scalar flux only. The angular quadrature does not change in the presence of the anisotropic treatment, so elements of $\boldsymbol{J}_\phi$ can be computed in a straightforward way for the even order components

$$j_{\phi nijklm}^e = \sum_{n=1}^{N_t} w_n \, Y_{lm}^e(\mu_n, \omega_n) \frac{\partial \psi_{n,i,j,k}}{\partial \varphi_{i',j',k',l',m'}^p} \quad (3.6.15.a)$$

and the odd moment components

$$j^{o}_{\phi nijklm} = \sum_{n=1}^{N_t} w_n \, Y^{o}_{lm}(\mu_n, \omega_n) \frac{\partial \psi_{n,i,j,k}}{\partial \vartheta^{p}_{i',j',k',l',m'}} \qquad (3.6.15.b)$$

The order of the moments per cell is consistent with the order of terms in the system of equations, Eq. (3.6.11).

Additional details on the construction of the ITMM operators in the presence of anisotropic scattering are provided in [19]. Ultimately these operators are used according to Eqs. (3.1.12) and (3.1.14) to solve for the angular moments of the flux and the outgoing angular flux at system boundaries, respectively.

3.6.3 Parallel Performance with Anisotropic Scattering

The anisotropic scattering formalism described above has been implemented in a special version of PIDOTS, using the PBJ global solution method. The DD spatial discretization is still used, meaning that the flux is expanded in a series of angular moments only (no higher spatial moments utilized). The results in the remainder of this section serve two purposes: 1) to demonstrate the proof of principle that the ITMM can be implemented with anisotropic scattering and 2) to determine if such implementation has a significant effect on the conclusions drawn from the results of isotropic scattering tests. Serial SI runs were performed for comparison and to ensure that the ITMM produces the correct converged solution; no mismatches outside the convergence criterion were observed between PBJ and SI.

Strong and weak scaling studies with *L* = 0, 1, and 3 were performed on the JPF system up to *P* = 1,024. The base model described in Sec. 3.4.2 has been used for all numerical experiments, featuring four different materials and four source strengths. However, to properly model a problem with anisotropic scattering, the model must be updated to account for all angular moments of the flux. First, the source strengths listed in Fig. 3.4.2 are assigned to the zero-th angular moment, i.e., the source is isotropic. Fixed sources of higher angular moments were not considered. Second, the scattering cross section input was augmented to account for higher scattering moments, σ_{sl}, being considered. Only a single set of cross sections were used for these experiments. Scattering cross section moments were restricted in two ways: $\sigma_{s0} \leq \sigma_t$ and $\sum_{l=1}^{L} \sigma_{sl} \leq \sigma_{s0}$. Table 3.6.I shows all materials' cross sections used for these tests. Note that the total cross sections have not changed from the values employed in the isotropic scattering cases. Lastly, all cell dimensions were set to *h* = 0.1 cm in order to examine the method's behavior with the more challenging, optically thin setting.

Table 3.6.I Anisotropic scattering scaling studies' cross section data.

Material	σ_t (cm^{-1})	σ_{s0} (cm^{-1})	σ_{s1} (cm^{-1})	σ_{s2} (cm^{-1})	σ_{s3} (cm^{-1})
1	2.000	1.800	0.600	0.100	0.200
2	1.000	0.900	0.300	0.050	0.100
3	0.500	0.450	0.150	0.025	0.050
4	1.500	1.350	0.450	0.075	0.150

3.6.3.1 Strong Scaling

Strong scaling studies with anisotropic scattering were performed up to P = 1,024. An 8×8×16 base model was formed by taking the cubic base model of Fig. 3.4.2 and doubling/stretching it in the z-direction. Additional processes were added to the topology one dimension at a time in the $z \rightarrow y \rightarrow x$ order used previously. For L = 0 and 1 cases, the S_8 angular quadrature set was selected. To conserve memory, the L = 3 case was provided only an S_4 quadrature.

The iteration counts for all three values of L are shown in Fig. 3.6.1. The sloping curves correspond to the PBJ results, and the flat curves correspond to serial SI results. Note the SI iteration counts are exactly the same, 17, for the L = 1 and 3 cases and 18 for the L = 0, isotropic case. Likewise, the order of anisotropy has little effect on the iteration count curves of the PBJ method. Because the problem is optically thin, all PBJ iteration curves climb steadily as sub-domains remain tightly coupled.

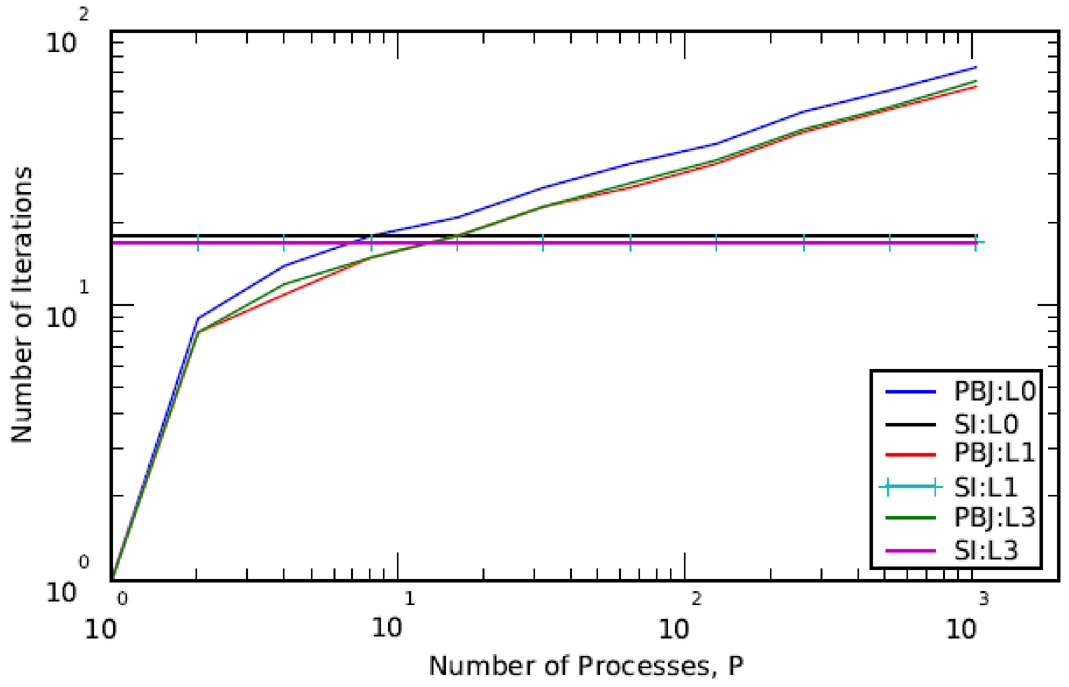


Fig. 3.6.1 JPF Anisotropic scattering, strong scaling, Iterations *vs*. P, h=0.1 cm, varying L.

The execution times for the L = 0 case are given in Fig. 3.6.2. Total (Tot) execution time is the sum of the iterative PBJ solution (Sol) time and the ITMM operator construction (Con) time. The anisotropic scattering has little influence on how the ITMM with PBJ behaves in strong scaling. Namely, for small P, operator construction dominates the execution, and when P is large, this time decreases and the execution time is almost entirely spent in the iterative solution phase. Moreover, the optimal sub-domain size occurs at 16 cells per sub-domain (P = 64). The test problem counts as a small, leaky system, and serial SI completes execution in a very short time.

The execution times for the *L* = 1 case, shown in Fig. 3.6.3, reveal a stronger shift upward for SI than PBJ. This is not surprising, because the computation of the higher order scattering source is required every iteration. This penalty has a smaller impact on the ITMM, which performs only a single mesh sweep and subsequently needs only fast matrix-vector operations per global iteration. When the PBJ method's execution time reaches a minimum at *P* = 128, it is approximately 7.5 times faster than the serial SI method.

Lastly, the execution times for the *L* = 3 case are given in Fig. 3.6.4. The shift upward by increasing the order of anisotropy from *L* = 1 is much smaller than the transition from *L* = 0 to *L* = 1. The time consumed per iteration has increased by a factor only slightly greater than one. Instead, the much larger construction time of the ITMM operators unequivocally indicates the weakness of the ITMM with anisotropic scattering for small *P*. However, the PBJ execution time drops quickly and reaches a minimum at *P* = 128 and is nearly a full decade faster than serial SI.

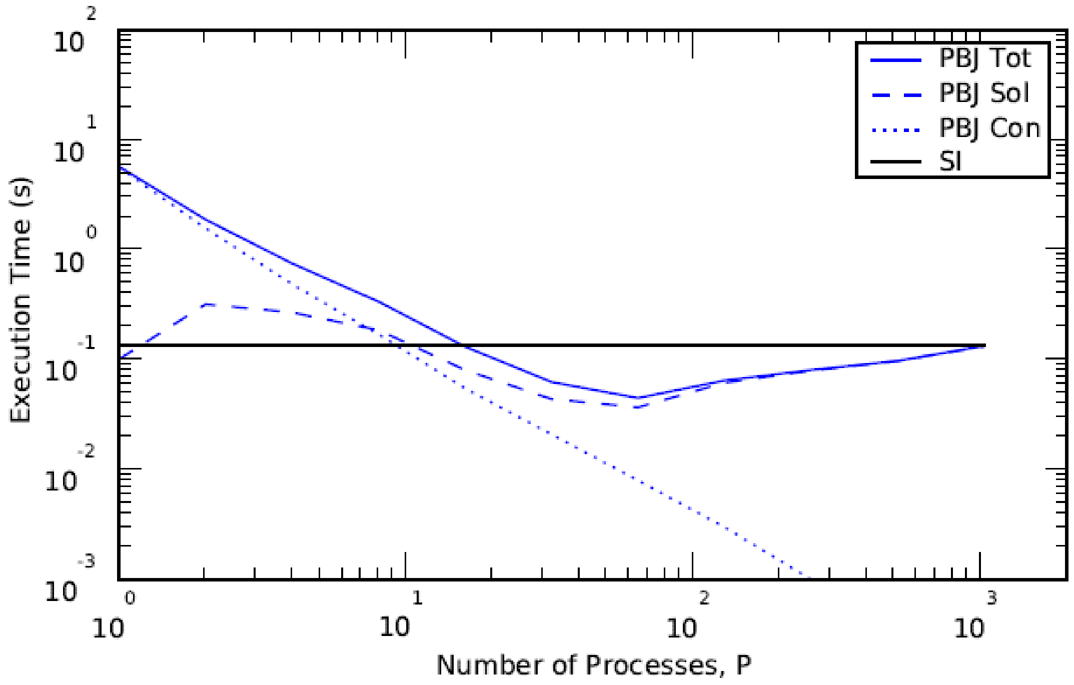


Fig. 3.6.2 JPF Anisotropic scattering, strong scaling, Execution Time *vs*. *P*, *h*=0.1 cm, *L*=0.

These strong scaling results are mostly consistent with the isotropic scattering results reported above. As the order of anisotropy is increased, the execution time per iteration increases slightly. However, the increase in construction time is much larger, which is expected due to the increase in operators' sizes. The results seem to show that sub-domains of either 8 or 16 cells provide the optimal blend of shortest construction time and fewest iteration count.

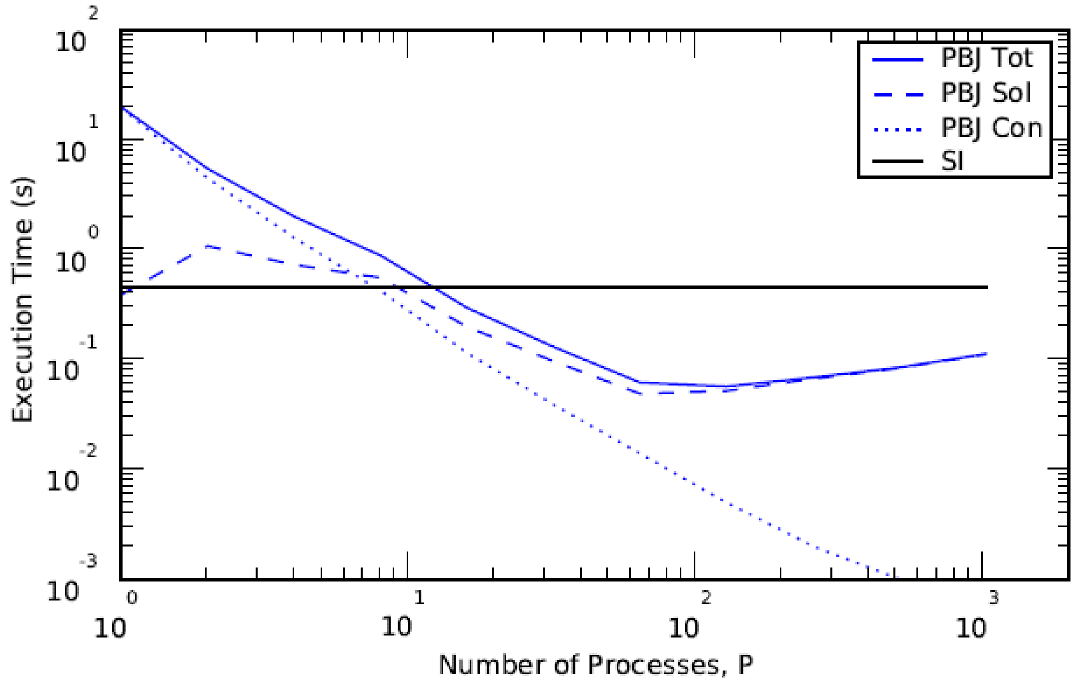


Fig. 3.6.3 JPF Anisotropic scattering, strong scaling, Execution Time *vs*. *P*, *h*=0.1 cm, *L*=1.

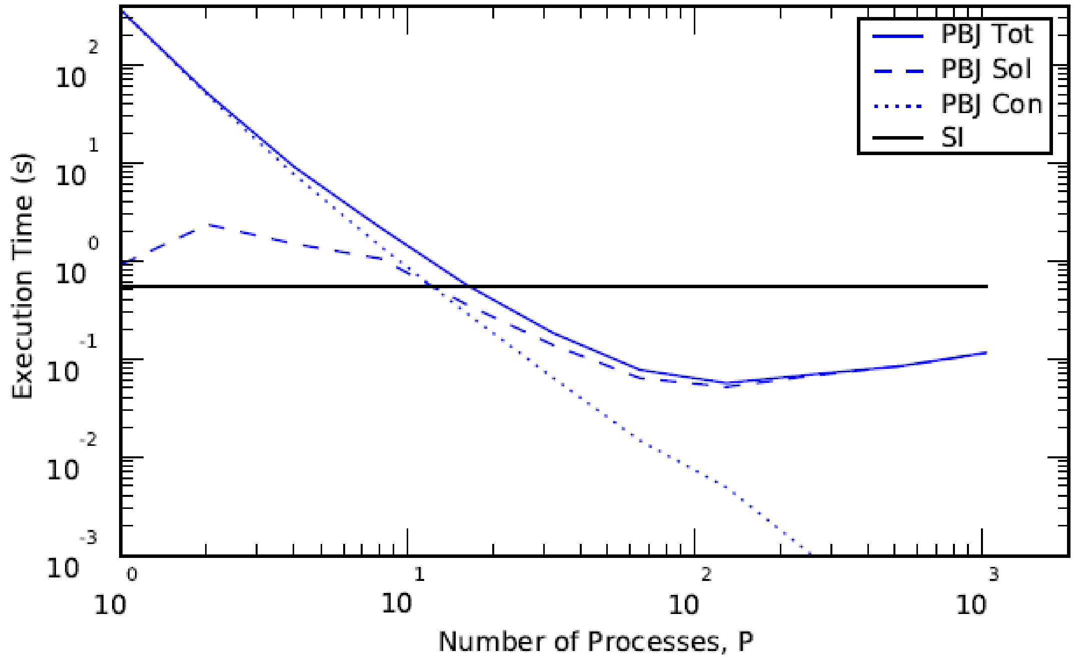


Fig. 3.6.4 JPF Anisotropic scattering, strong scaling, Execution Time *vs*. *P*, *h*=0.1 cm, *L*=3.

3.6.3.2 Weak Scaling

The weak scaling studies were performed up to P = 1,024 using the 8×8×8 base model of Fig. 3.4.2. Additional processes were added to the topology one dimension at a time in the $z \rightarrow y \rightarrow x$ order and the base model domain was stretched in the affected direction. The S_8 quadrature set was used for the L = 0 and 1 cases, and the S_4 quadrature set was used for the L = 3 case. Again, all cell dimensions were set to h = 0.1 cm. No comparisons with SI were made for these cases. Results are meant to purely illustrate the impact anisotropic scattering has on the ITMM with PBJ.

The iteration counts for the three values of L are shown in Fig. 3.6.5. The trend from strong scaling results seems to persist here too: increasing order of anisotropy has minimal effect on the PBJ iteration counts. In fact, the L = 1 and L = 3 curves nearly exactly match and consume fewer iterations at each value of P than the isotropic scattering case. All curves steadily rise with increasing P, because the cells are optically thin, and the sub-domains are not sufficiently decoupled.

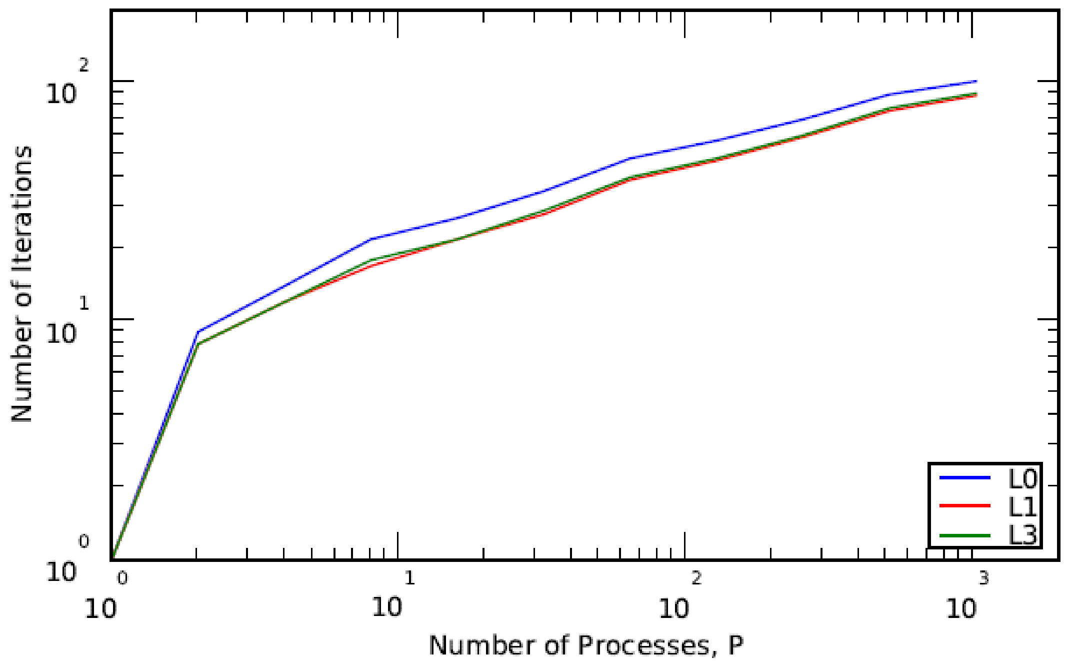


Fig. 3.6.5 JPF Anisotropic scattering, weak scaling, Iterations *vs*. *P*, *h*=0.1 cm, varying *L*.

The execution times for the L = 0 case are presented in Fig. 3.6.6. This plot is similar to the PBJ weak scaling figures with isotropic scattering for the optically thin cases. As the iteration counts increase with larger problem size, the iterative solution consumes more time.

Execution times are all shifted upward in Fig. 3.6.7 for the L = 1 case. Although the iterations decreased in Fig. 3.6.5, the operators and vectors are larger to account for the first angular moment of the flux. The construction time has increased by about a factor of three from the isotropic case. This results in the iterative solution time remaining the dominant consumer of execution time.

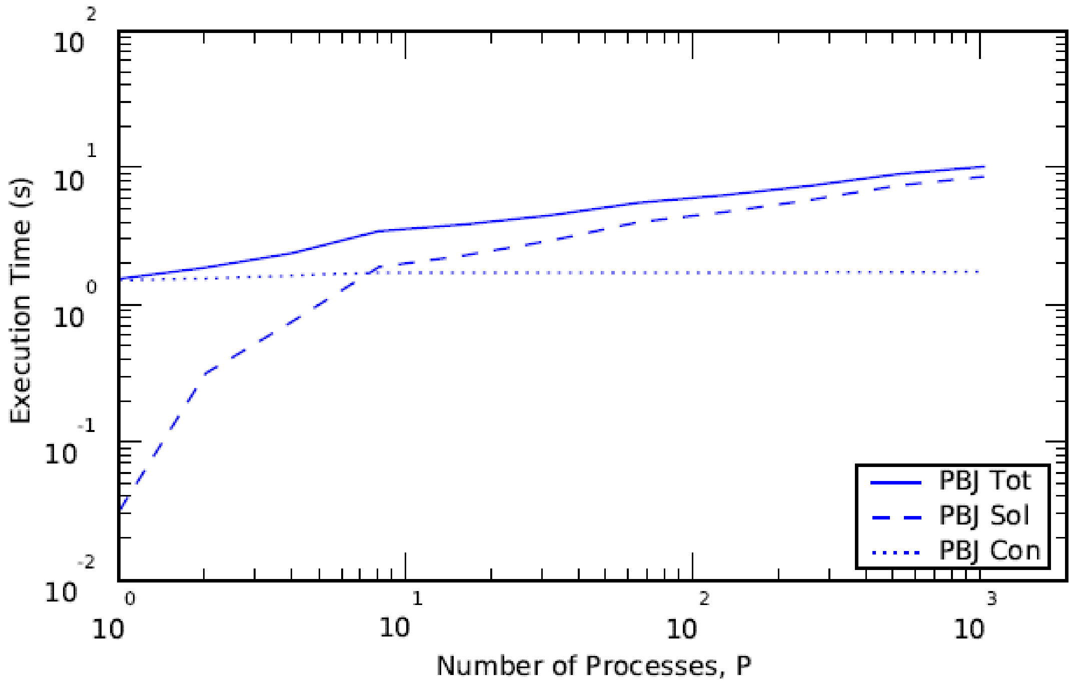


Fig. 3.6.6 JPF Anisotropic scattering, weak scaling, Execution Time *vs*. *P*, *h*=0.1 cm, *L*=0.

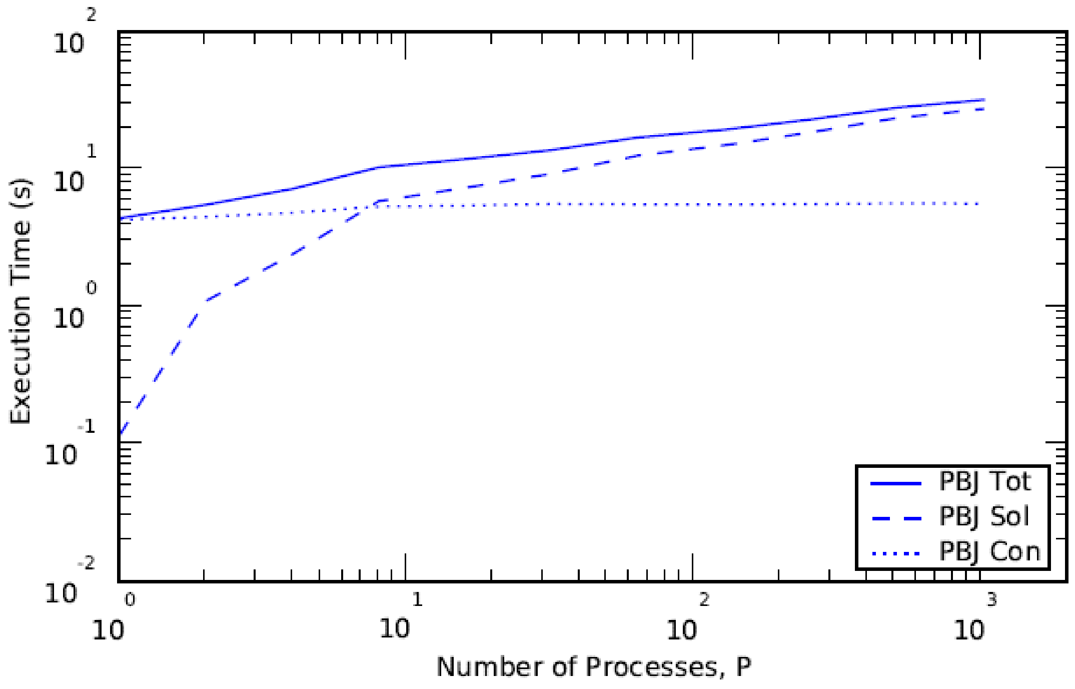


Fig. 3.6.7 JPF Anisotropic scattering, weak scaling, Execution Time *vs*. *P*, *h*=0.1 cm, *L*=1.

Increasing the anisotropy order to *L* = 3 drastically increases the construction time. In Fig. 3.6.8, it is evident that the construction time dominates total execution up to ~256 processes. Nevertheless, as *P* continues to increase, the iterative solution time will eventually dominate, as evident for *P* = 1,024, because of the growing number of iterations.

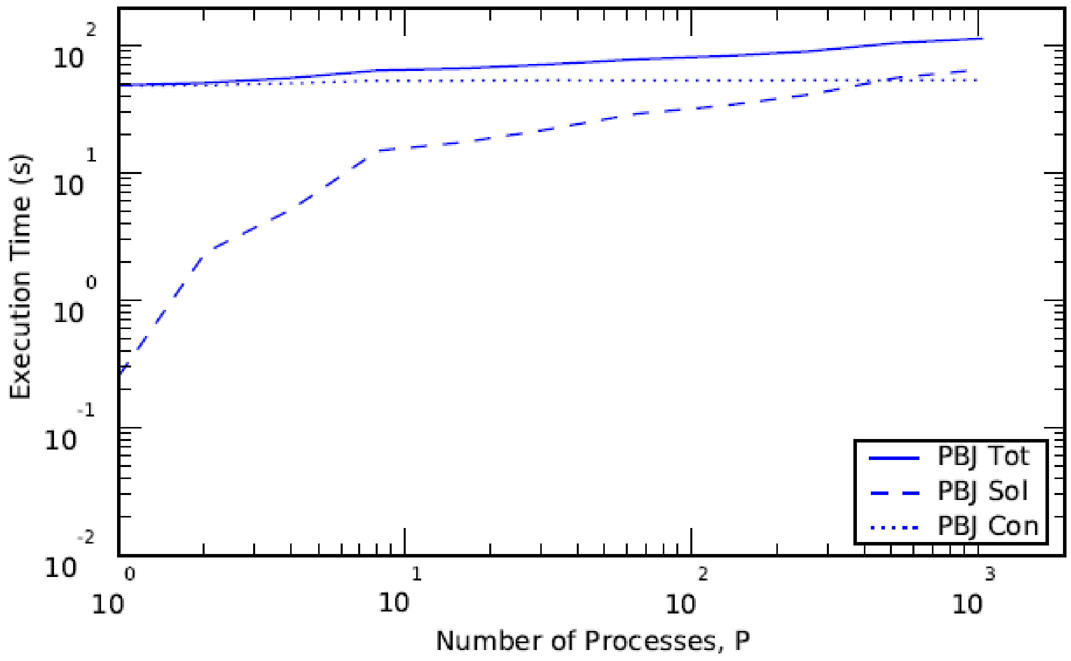


Fig. 3.6.8 JPF Anisotropic scattering, weak scaling, Execution Time *vs*. *P*, *h*=0.1 cm, *L*=3.

3.7 Task 7 – Investigate pre-conditioners

The above results have demonstrated the importance of minimizing the ITMM operators' construction time and the time per iteration. This is done most effectively by simply reducing the size of the operators and assigning each PE multiple sub-domains, i.e., the PGS approach. However, sub-domain sizes are bounded from below by a single-cell size, and increasing the number of sub-domains results in increasing number of global iterations, and consequently longer execution times.

The next challenge is to reduce the number of iterations till convergence. That is, assuming the operator construction time and time per iteration has been effectively minimized, the only way to improve the performance and scalability of the ITMM is to improve the convergence rate. Evident in the above results, the problems with large optical thickness tend to flatten in number of iterations and execution time grows rather slowly. Sub-domain decoupling does not manifest itself in the optically thin regime and execution time grows much more rapidly, making the method both poorly scalable and uncompetitive with KBA-based parallelization.

In this task a spatial multigrid method (SMG) is presented as an attempt to accelerate PBJ iterations. PBJ was selected instead of PGS or PGMRES (as a preconditioner) because it is the simplest method to consider and was assumed to be less likely to introduce programming challenges (e.g., indexing sub-domains of different grids) to SMG development. Moreover, it was presumed that if an effective and robust SMG method was not possible to develop for PBJ, then one would not be possible to develop for the other global solution methods. Correctness of this presumption has not been verified, and applying SMG to the other methods remains a potential topic for future research. Test results indicate the strengths and weaknesses of SMG relative to PBJ, and conclusions are drawn, albeit absent formal spectral analysis. Such analysis, which is a key goal of future work, will clearly indicate how the optical thickness and scattering ratio affect convergence rates and hopefully indicate means to improve the SMG method beyond its current status.

3.7.1 Parallel Spatial Multigrid with the ITMM

The ITMM with the PBJ global solution has been modified to incorporate a spatial multigrid *V*-cycle scheme. Within a sub-domain the ITMM operators couple all cells to one another and to the sub-domain's boundaries. PBJ iterations carry information about particle spatial distribution across the global domain, coupling cells in different sub-domains. An SMG method is intended to accelerate this process by combining sub-domains, thus permitting an explicit expression for the coupling among cells in different sub-domains on the finest grid, albeit with the additional approximation of larger spatial cells.

The starting point for this description is a homogeneous domain with vacuum boundary conditions. Moreover, uniform grid spacing is assumed, although variable mesh dimensions do not force a deviation from the remainder of this discussion. The scheme has been developed with weak scaling in mind; comments regarding the SMG method applied to strong scaling are reserved for the conclusions at the end of the chapter. A homogeneous domain is selected to avoid the problem of mixing cross sections of different materials assigned to multiple cells in the fine mesh that comprise a single cell in the coarse mesh, a process known as *homogenization*. When multiple materials are homogenized, nuclear data must be combined in a manner that best preserves the response to be expected from the heterogeneous system. Typically, weighting cross sections by cells' scalar flux and volume (flux-volume weighting) is done to capture the effects each material has on the distribution. Unfortunately in the *V*-cycle SMG scheme the flux is different each time the grid is re-evaluated, and the homogenization would have to be repeated. Moreover, volume weighting alone was not considered because it is not believed to be accurate enough. Rather, the scheme is as yet unprepared to model heterogeneous domains unless the varying materials are never combined together into a single cell during the coarsening process. That is, the coarsest grid can have multiple materials, assuming that the preceding coarsening steps were always performed over homogeneous regions of the domain. Reflective boundary conditions create an additional iterative burden in the PBJ-SMG and are avoided to focus on the comparative convergence between PBJ and PBJ with SMG.

The global problem is decomposed into P sub-domains on P PEs. Coarsening each sub-domain individually will not provide significant benefit to convergence. A simple example has been designed to illustrate this point. A homogeneous problem with the scattering ratio c = 0.99 and on fixed P = 64 PEs has been run with cubic cell dimension varied as h = 0.1 cm, 1.0 cm, and 10.0 cm. The base case of 8×8×8 cells per sub-domain was coarsened to 4×4×4, 2×2×2, and 1×1×1 cells per sub-domain; each time the sub-domain is coarsened, cell volume increases by a factor of eight. Table 3.7.I shows the change in the number of PBJ iterations. For optically thin domains coarsening the sub-domain has essentially no effect on convergence. Coarsening a sub-domain to a single cell does lead to fewer iterations in the optically thick case, h = 10.0 cm, but the gain is not significant enough to outweigh the expected benefits of coarsening over sub-domain boundaries immediately.

Table 3.7.I Number of PBJ iterations during coarsening for homogeneous, c = 0.99 test problem.

		h (cm)	
Case	0.1	1.0	10.0
8×8×8	71	124	90
4×4×4	70	122	84
2×2×2	70	119	91
1×1×1	69	108	73

With Table 3.7.I in mind, it is assumed that real improvement in convergence will come as a result of coarsening across sub-domain boundaries. This is not surprising because it explicitly couples regions of the global domain that previously were not directly coupled but shared information via iteration on the interfacial angular fluxes. Therefore, the SMG scheme's coarsening parameter, denoted κ, refers to the number of sub-domains across which to coarsen in each direction.

In the PBJ framework, the number of sub-domains is equivalent to the number of processes, and more specifically, the sub-domain mesh coincides with the virtual mesh process topology, $P_x \times P_y \times P_z$. Starting at the finest grid, P_x, P_y, and P_z are successively coarsened by the coarsening parameter κ. Using P_x as an example, at the finest grid, κ x-direction sub-domains (over κ processes) are combined into a single sub-domain, leaving $P_x / \kappa \rightarrow P_x$ sub-domains remaining in the coarser grid's x-direction. Analogous treatment is applied to the y- and z-directions. κ will necessarily change as grids become coarser because fewer processes are left in each dimension over which to coarsen. Ultimately, the sub-domains (and processes) are reduced from a $P_x \times P_y \times P_z$ mesh to a 1×1×1 mesh. Different values for κ were tested for their impact on the SMG scheme before settling on a maximum value of κ = 8. That is, so long as P_x, P_y, or P_z equals 8 or more, then κ = 8. As grids become coarser, the sizes of P_x, P_y, and P_z decrease. The test problems employed in this task are specifically designed to always have the number of sub-domains in each direction equal to a power of two. Thus, when either P_x, P_y, or P_z is less than 8, κ equals 4 or 2 or 1, depending on the value of P_x, P_y, or P_z.

The SMG scheme was designed to keep the sub-domains at the same size (number of cells) at all grid refinement levels. This allows for greater predictability in the amount of memory needed at each grid and in the execution time of ITMM operator construction and matrix-vector operations at each grid. The cells in each sub-domain are also arranged in a Cartesian mesh, sized $I \times J \times K$. The values of I, J, and K relative to κ have implications on the coarsening at each grid. Using the x-direction as an example again, when $I = \kappa$, the κ sub-domains in the x-direction are each coarsened to a single cell within a new sub-domain of I cells. If $I > \kappa$, the κ sub-domains are each reduced to I/κ cells. If $I < \kappa$, multiple sub-domains must be coarsened together to form a single cell on the coarser grid. This lattermost case creates additional complications during restriction and interpolation and was consequently avoided during formal testing of the SMG scheme.

To help illustrate the preceding description, Fig. 3.7.1 shows a 2-D example of coarsening over a 2×2 sub-domain mesh (solid lines), where each sub-domain has 2×2 cells (dashed lines). In the illustration, κ = 2. The fine grid on the left is coarsened and results in the coarse grid depicted on the right, which maintains the physical size of the fine grid, but does so with only a single sub-domain. Depicted in Fig. 3.7.1, each sub-domain's space on the fine grid has been coarsened and is modeled by a single cell on the coarse grid, and the coarse grid's sub-domain is the same size, in terms of number of cells, as all four sub-domains on the fine grid.

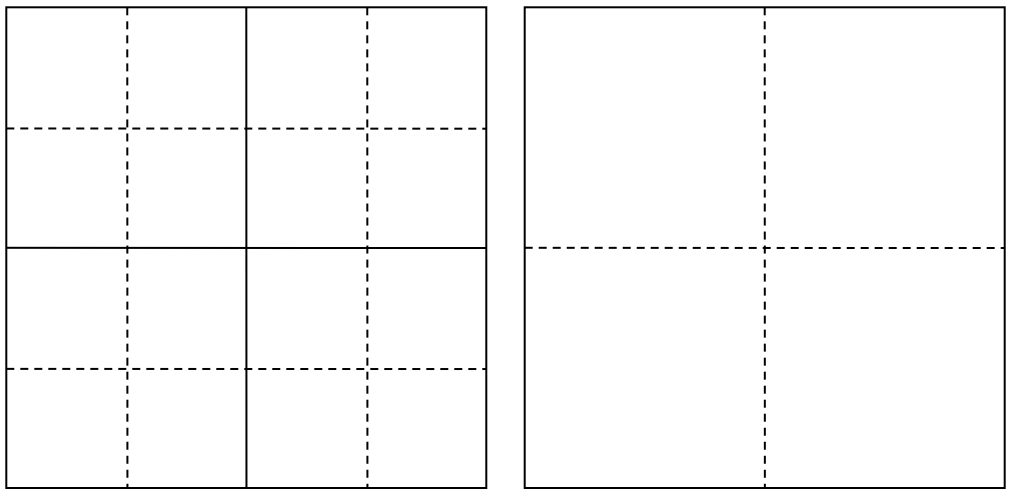


Fig. 3.7.1 2-D Illustration of PBJ-SMG coarsening with 4 processes and 4 cells per process.

Figure 3.7.1 also shows the process topology numbering on the fine grid and the coarse grid. The important feature to notice is that the coarse grid uses the same processes as the fine grid. Alternatively, a new virtual process topology could have been formed for each grid modeled during the *V*-cycle. However, for this work, SMG was implemented by using a single topology and assigning successively coarser grids to PEs already participating at the fine grid level. Eventually, a single process has the coarsest (sub-)domain that it solves directly.

Following standard multi-grid notation, the scalar flux solution vector for each grid *a* is stored in the array $\boldsymbol{\phi}^{ah}$, which has been indexed to account for all the grids. Likewise, $\boldsymbol{\psi}_{out}^{ah}$ represents the angular flux solution vector for each grid. At the finest-level grid, $\boldsymbol{\phi}^{h}$ and $\boldsymbol{\psi}_{out}^{h}$ comprise the actual solution. At the coarsened grids, $\boldsymbol{\phi}^{ah}$ and $\boldsymbol{\psi}_{out}^{ah}$ comprise solutions of the recursively formed residual equations.

During the flow of a single *V*-cycle, the residual must be restricted. The easiest method—and the one selected here—is to average the combined residual elements. If the total problem solution on a grid *a* is comprised of the vectors $\boldsymbol{\phi}^{ah}$ and $\boldsymbol{\psi}_{out}^{ah}$, then the residual is best expressed as $\boldsymbol{r}_{\phi}^{ah}$ and $\boldsymbol{r}_{\psi}^{ah}$. The $\boldsymbol{r}_{\phi}^{ah}$ and $\boldsymbol{r}_{\psi}^{ah}$ vectors are computed after multiple PBJ iterations. The $\boldsymbol{r}_{\phi}^{ah}$ vector is restricted by computing the average value from the elements that correspond to cells that are being coarsened together. For example, the bottom, left four cells in the fine grid of Fig. 3.7.1 are coarsened together to form the single bottom, left cell of the coarse grid. Therefore, the residuals corresponding to those four cells are averaged together during restriction. Likewise, the $\boldsymbol{r}_{\psi}^{ah}$ vector is restricted by computing the average value from the elements that correspond to faces (3-D) that are being coarsened together. The processes that will continue to the next coarser grid receive a message containing these average values from each of the processes of the current grid. The receiving process reorders the data for use as the source function of the system of equations representing the coarse grid.

The SMG scheme is currently implemented to compute an approximation of the error of $\boldsymbol{\phi}^{ah}$ only to correct the solution, not the $\boldsymbol{\psi}_{out}^{ah}$ error. The $\boldsymbol{\psi}_{in}^{ah}$ vector is saved at the completion of the iterations of each grid. After the system is solved exactly at the coarsest grid *X*, $\boldsymbol{\phi}^{Xh}$ is corrected for the next finest grid *a* without performing a full computation of $\boldsymbol{\phi}^{ah}$ according to the residual equation form of Eq. (3.1.12). Using the corrected $\boldsymbol{\phi}^{ah}$ and the saved $\boldsymbol{\psi}_{in}^{ah}$, a new $\boldsymbol{\psi}_{out}^{ah}$ is computed and the iterations continue. This process is performed for each grid until the finest grid is reached, whereby the actual solutions are being updated, and $\boldsymbol{\phi}^{h}$ can be checked for convergence.

Considering that only $\boldsymbol{\phi}^{ah}$ is corrected, the interpolation must only account for that vector. The easiest way to interpolate is to apply the correction from the coarse grid's cells to the appropriate fine grid cells equally, without bias for the actual flux distribution. This approach has been found to be ineffective. Instead, after the elements of $\boldsymbol{r}_{\phi}^{ah}$ are averaged over cells coarsened together, the elements of $\boldsymbol{\phi}^{ah}$ are also averaged. Then each element of $\boldsymbol{\phi}^{ah}$ that contributes to the average value of the coarse cell is divided by the average and the ratio is stored. During interpolation, the coarse-cell correction value is multiplied by this ratio for each fine-cell comprising the coarse-cell, thus weighting the correction according to which finer cells have the highest flux and contribute most to the coarse cell's flux value. This interpolation by weighted-average of $\boldsymbol{\phi}^{ah}$ has been found to improve the SMG scheme significantly compared to no weighting or instead weighting by $\boldsymbol{r}_{\phi}^{ah}$.

The SMG algorithm requires a new coefficient matrix for each grid refinement level in the *V*-cycle. Per the description provided in this report for PBJ, the coefficient matrix is in fact a large, sparse, block matrix whose elements are the ITMM operators. Given its size and the complex nature of the restriction and interpolation operators above, the Galerkin condition was not deemed applicable as the method by which to compute the coarse grids' coefficient matrices. Alternatively, this coefficient matrix is recomposed for each grid by performing a new differential mesh sweep over the coarsened system. Processes that advance to a coarser grid sweep over the coarsened system and construct the ITMM operators for the physically larger cells. The ITMM operators are indexed by grid.

The overall goal of this SMG scheme is to transmit information about transport effects more quickly over the entire domain. The typical PBJ method transfers information each iteration, coupling highly separated sub-domains and cells via interfacial angular fluxes. However, this SMG scheme couples cells from different sub-domains together explicitly through one or more stages of grid coarsening. This has the potential of introducing harmful truncation errors that will not accelerate the convergence or, even worse, that make the entire system diverge. Yet, if successful the method will allow for the direct computation of an approximation of the scalar flux error that can be used to improve the solution at lower wall-clock execution times.

Two disadvantages of this SMG scheme should be noted. First, because only certain processes advance to coarser grids, most processes will suffer significant times of idleness, and all but the root process will suffer some time of zero workload. Furthermore, the processes assigned to multiple grids will necessarily have a greater memory burden. This drawback was accounted for when sizing problems for numerical experimentation by determining the maximum number of grids to which a single PE would be assigned and multiplying this number with an estimate of the sizes of the ITMM operators.

3.7.2 SMG Numerical Tests and Results

Numerical experiments have been performed using the V-cycle SMG scheme described in the previous section. All test problems have a homogeneous material map, a unit volumetric source in every cell, and a uniform mesh of cubic cells with varying edge-length h. Weak scaling tests were performed on the JPF system up to P = 4,096 with c = 0.9 and 0.99 and h = 0.1 cm, 1.0 cm, and 10.0 cm. The S_8 quadrature set was used. Table 3.7.II presents the process topology of the weak scaling problems. The $z \rightarrow y \rightarrow x$ order was used for expanding the domain. Following the aforementioned description of the coarsening factor κ, Table 3.7.II also lists the number of grids modeled. Note that when the number of processes in some dimension d, P_d, is 8 or more, κ = 8; when P_d = 4, κ = 4; when P_d = 2, κ = 2; and when P_d = 1, κ = 1 (i.e., there is no further coarsening to be performed). As has been the case with previous testing, execution timing results are presented as an average from multiple runs that are assumed to exhibit typical system performance.

Table 3.7.II PBJ-SMG testing process topology and number of V-cycle grids.

P_x	P_y	P_z	Maximum κ	Number of Grids
1	1	1	1	1
1	1	2	2	2
1	2	2	2	2
2	2	2	2	2
2	2	4	4	2
2	4	4	4	2
4	4	4	4	2
4	8	8	8	2
8	8	8	8	2
8	8	16	8	3
8	16	16	8	3
16	16	16	8	3

Preliminary testing showed benefits of minimizing the work performed at the coarse grid. Most of the computational burden is spent at the fine grid with periodic V-cycle multigrid corrections performed to improve the scalar flux solution. The test results presented below are from executions that perform the following steps:

1) Three pre-correction PBJ iterations at the finest grid,
2) One pre-correction PBJ iteration at intermediate coarse grids,
3) Exact solution of the coarsest grid's system of equations,
4) One post-correction PBJ iteration at intermediate coarse grids, and
5) Two post-correction PBJ iterations at the finest grid.

Note that this essentially means one *V*-cycle is performed for every five fine-grid PBJ iterations. A single pre-correction coarse-grid iteration on intermediate coarse grids is constituted by solving the residual equation assuming $\boldsymbol{\psi}_{in}^{ah} = 0$ because no information about this term is available until $\boldsymbol{\psi}_{out}^{ah}$ is computed. However, little to no benefit was found by performing full iterations over these grids for these test problems. Furthermore, a single post-correction iteration on intermediate coarse grids is constituted by correcting $\boldsymbol{\phi}^{ah}$ only. A coarser grid's $\boldsymbol{\phi}^{ah}$ is interpolated and the result is added to the current grid's solution. No further work is performed on that grid. The next finest grid continues the process by interpolating the solution and updating. These traits considered, the pre- and post-correction iterations form a scheme that successively restricts the solution vector from the finest grid to a coarse grid that has only one (sub-)domain and is solved directly for a correction to the fine-grid solution. This coarse-grid solution is interpolated at the intermediate grids until it is ultimately summed with the current iterate of the fine-grid scalar flux. Note from Table 3.7.II, that only the *P* = 1,024; 2,048; and 4,096 executions have an intermediate coarse stage.

Comparisons have been made with the PBJ method. The SMG solutions have been checked for correctness of the converged solution. The goal of these experiments is to determine under which circumstances the PBJ-SMG method successfully converges to the correct solution and does so at a reduced cost of wall-clock execution time. As will be seen shortly, the PBJ-SMG method will in some cases not converge to the correct solution, either stagnating in the reduction of error or completely diverging. These deficiencies in the algorithm must be addressed before wide-spread application of the method for this purpose, and they represent a topic for future research.

The numbers of iterations for the *c* = 0.9 cases are presented in Figs. 3.7.2–3.7.4. The +-marked, solid, black lines represent the reference PBJ solutions (PBJ-Ref); the dashed, blue lines represent the number of *V*-cycles being consumed in PBJ-SMG executions (SMG-Vcyc); the dotted lines represent the total numbers of iterations performed on all grid levels of PBJ-SMG executions per the above description of the type/number of iterations at each grid level (SMG-All); and the solid, blue lines represent the number of PBJ-SMG iterations that are performed at the finest grid (SMG-Fine). To be clear, the PBJ-SMG iterations do not represent equal amounts of work at different grid refinement levels or between pre- and post-correction steps. However, all these curves help to identify trends that describe the behavior of the iterative process, especially relative to changing *c* and *h*. Because iterations at the finest grid are regular PBJ iterations, the SMG-Fine curves are shown for comparison. If at any *P* the value of this curve is less than the reference PBJ iteration count, then SMG is successfully helping the PBJ method. Otherwise, the SMG method distracts the PBJ method's convergence properties.

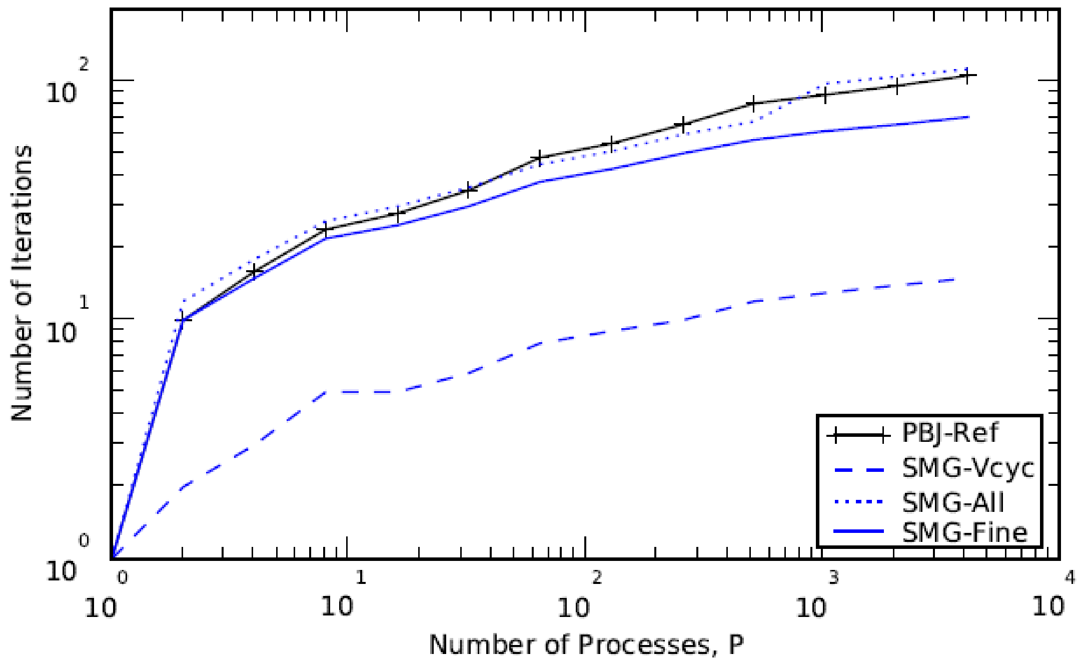


Fig. 3.7.2 JPF SMG weak scaling, Iterations *vs*. *P*, *c*=0.9, *h*=0.1 cm.

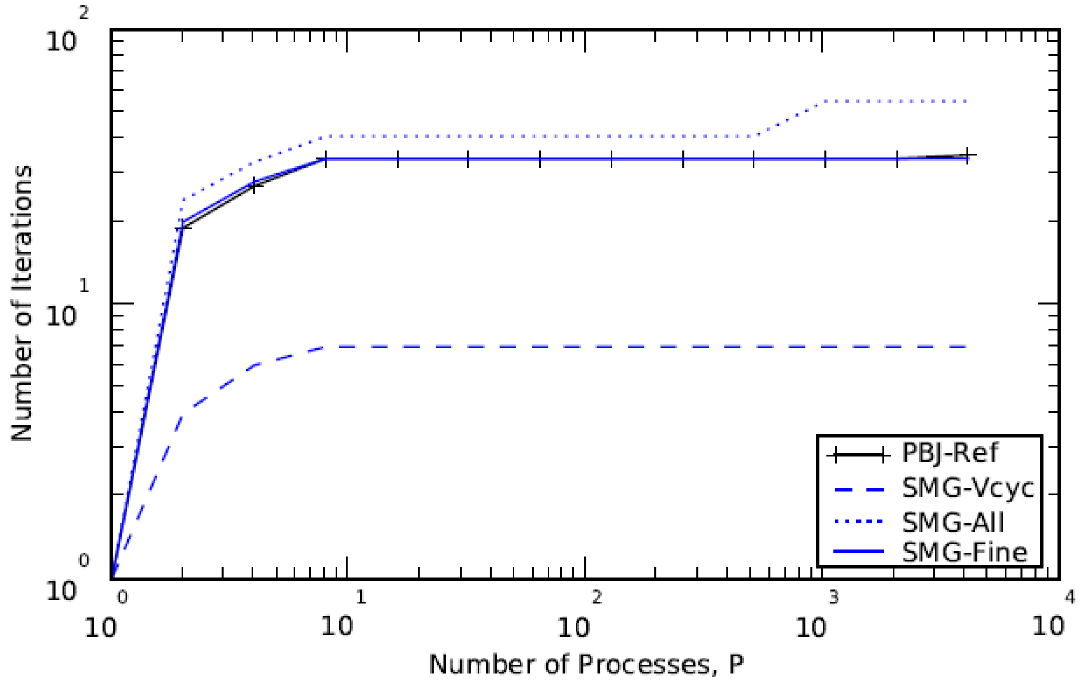


Fig. 3.7.3 JPF SMG weak scaling, Iterations *vs*. *P*, *c*=0.9, *h*=1.0 cm.

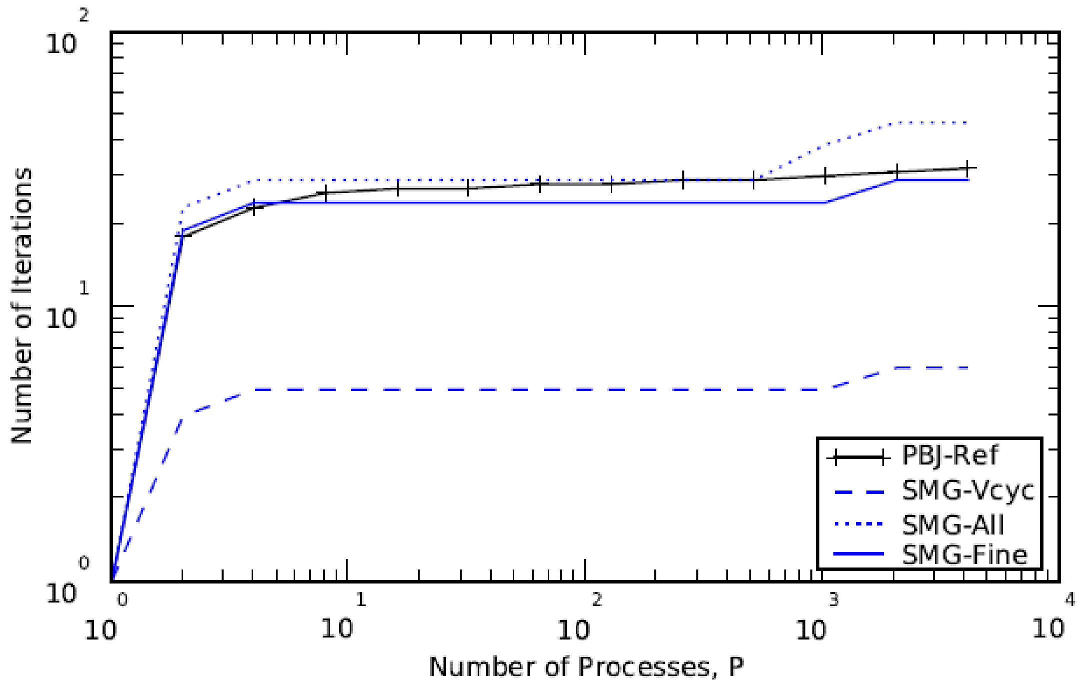


Fig. 3.7.4 JPF SMG weak scaling, Iterations *vs*. *P*, *c*=0.9, *h*=10.0 cm.

The iteration count curves for the *c* = 0.9, *h* = 0.1 cm case are shown in Fig. 3.7.2. The reference solution follows the trend anticipated from the previous results. The SMG method follows a similar trend of steadily growing iteration count no matter how those iterations are counted: *V*-cycles, all grids' iterations, or fine-grid iterations. Note that as *P* increases, the number of fine-grid iterations stays below the reference curve, suggesting that the SMG method is helping, albeit in a fairly small way. The iterations from all grids has a slight jump from *P* = 512 to 1,024 when the number of grids increases from two to three; the same jump is observed in the *h* = 1.0 and 10.0 cm experiments discussed below. At this point, the SMG transitions from a two-grid scheme to a three-grid scheme and each V-cycle performs pre- and post-correction iterations on the intermediate grid. These results are generally favorable, because the PBJ method is being aided by SMG.

Increasing the cell dimension to *h* = 1.0 cm eliminates the effectiveness of the SMG scheme. The iteration count curves shown in Fig. 3.7.3 for the *c* = 0.9, *h* = 1.0 cm case reveal that the SMG scheme is consuming an equal number of fine-grid iterations as the PBJ reference. The additional iterations performed on the coarse grids will only add to the SMG execution time and invariably make the scheme more expensive than the reference PBJ case. The fortunate result of this experiment is that the SMG scheme follows a consistent trend with the PBJ reference in that the curves are all flat and are essentially insensitive to *P*, the domain size.

When the optical thickness is increased again by making *h* = 10.0 cm, the SMG method again retains the feature of flat growth in number of iterations with increasing *P*. The number of iterations at the fine grid for SMG executions is a few iterations less than the reference executions. The increase in the number of grids and an increase in the number of *V*-cycles leads to a noticeable jump in the total number of iterations.

Learning from the previous figures that the SMG iteration trends are consistent with the PBJ reference is a positive sign in the development of this acceleration scheme. However, the arguably more important measure is the execution time. The execution time per coarse-grid iteration is anticipated to be less than the execution time per iteration on the fine grid. This is because the specific implementation does not feature communication of $\boldsymbol{\psi}_{out}^{ah}$ data during those iterations, saving time. However, when the number of SMG fine-grid iterations is approximately the same or greater than the reference number of PBJ iterations, one can be certain the total execution time will be greater due to the additional labor on coarse grids. Moreover, the execution time will increase due to the additional time spent constructing the coarse-grid ITMM operators.

The execution time curves for the c = 0.9, h = 0.1 cm case are given in Fig. 3.7.5. The solid blue line is the total execution time of the SMG method. It is the sum of the iterative solution time (dashed line) and the total ITMM operators construction time summed over all grid refinement levels (dotted line). The construction time has two increases as the number of grids increases and a PE is responsible for constructing a new set of ITMM operators. Interestingly, the second jump is larger, potentially because the memory burden per PE is larger and data fits into cache less efficiently. Nevertheless, the construction time is a fairly small contributor to execution time. With the total number of SMG iterations from all grid levels approximately equal to the number of reference PBJ iterations, it is evident that the SMG method reduces the total execution time only slightly. However, the gain is insignificant and indicates that SMG does not accelerate the PBJ method with the same success that DSA does for SI.

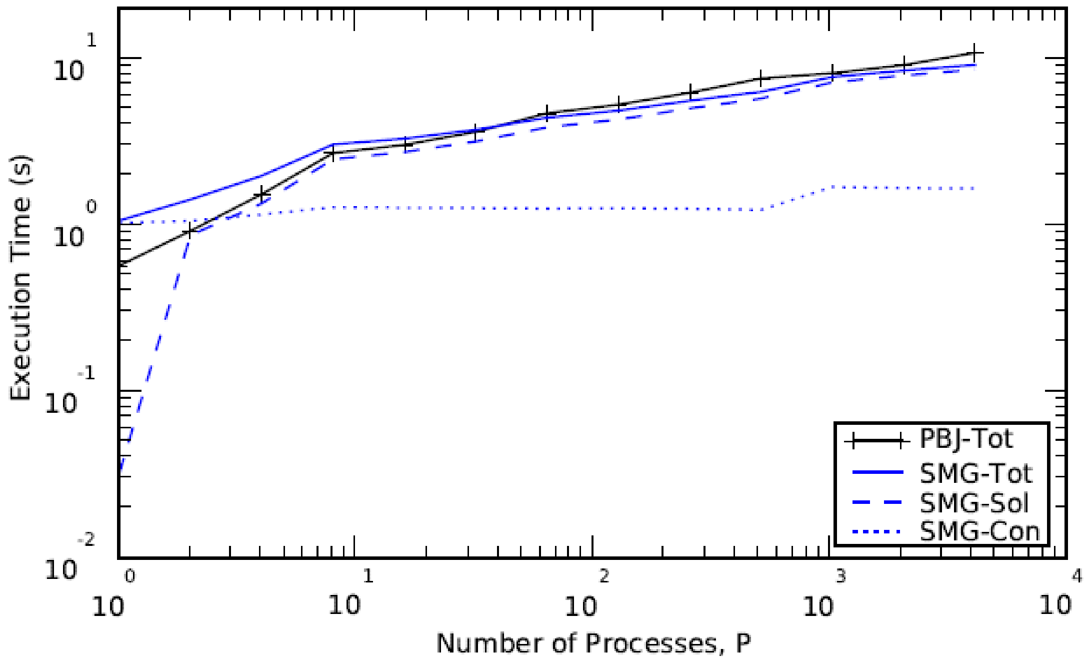


Fig. 3.7.5 JPF SMG weak scaling, Execution Time *vs*. *P*, *c*=0.9, *h*=0.1 cm.

In Fig. 3.7.6 the execution time plots for the *c* = 0.9, *h* = 1.0 cm case are given. Because the number of SMG fine-grid iterations is equal to the number of reference PBJ iterations in Fig. 3.7.3, the result that SMG consumes more execution time is not surprising. A benefit of the small number of operations performed per iteration on the coarse grids is that they add a relatively small amount of execution time to the iterative solution. That is, the relative difference between the SMG-All and PBJ-Ref iteration counts in Fig. 3.7.3 for the highest values of *P* is greater than the difference for the respective total execution times in Fig. 3.7.6.

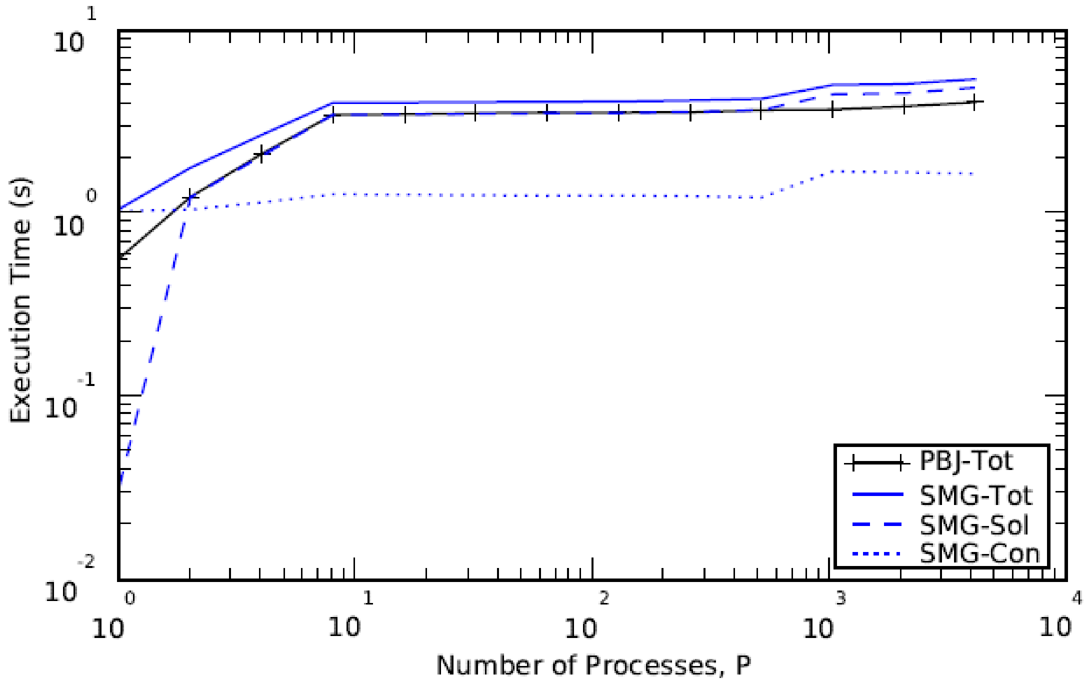


Fig. 3.7.6 JPF SMG weak scaling, Execution Time *vs*. *P*, *c*=0.9, *h*=1.0 cm.

The execution time plots for *c* = 0.9, *h* = 10.0 cm are given in Fig. 3.7.7. Whereas it was anticipated that the SMG method would consume more execution time for the *c* = 0.9, *h* = 1.0 cm case given the iteration trends, this case uses fewer fine-grid iterations but more iterations over all grids. The question is whether the expectedly faster coarse-grid iterations would be small enough to result in a faster overall execution. They do not; the SMG method consumes slightly more wall-clock execution time.

These results clearly indicate that while the number of SMG iterations over all grids can increase slightly from the reference PBJ number of iterations, it must be a small difference if any gain is to be attained in the execution time. Further, as Fig. 3.7.5 shows, this gain is very small. However, fortunately for these *c* = 0.9 cases, the SMG method converges to the same answer as PBJ and does so following essentially the same trends in iteration count curves and total execution time. The next challenge was to increase the scattering ratio and determine the impact on the SMG scheme relative to its already documented impact on the PBJ method.

The iteration count curves for the *c* = 0.99, *h* = 0.1 cm case are shown in Fig. 3.7.8. Two important points must be highlighted from these results. The first point is the SMG method is using a significantly smaller number of iterations, even over all grid refinement levels, than the PBJ reference calculations. Moreover, it is achieving this success by slightly bending the iteration curve, evident from the fact that the curves do not share a slope and the SMG method's curves are flattening with increasing *P*. However, the second point is that the SMG method failed to converge for the *P* = 4,096 job. Observing the relative error from the output of this execution, the SMG iterations are diverging from the correct solution, but the cause of this failure has not been identified.

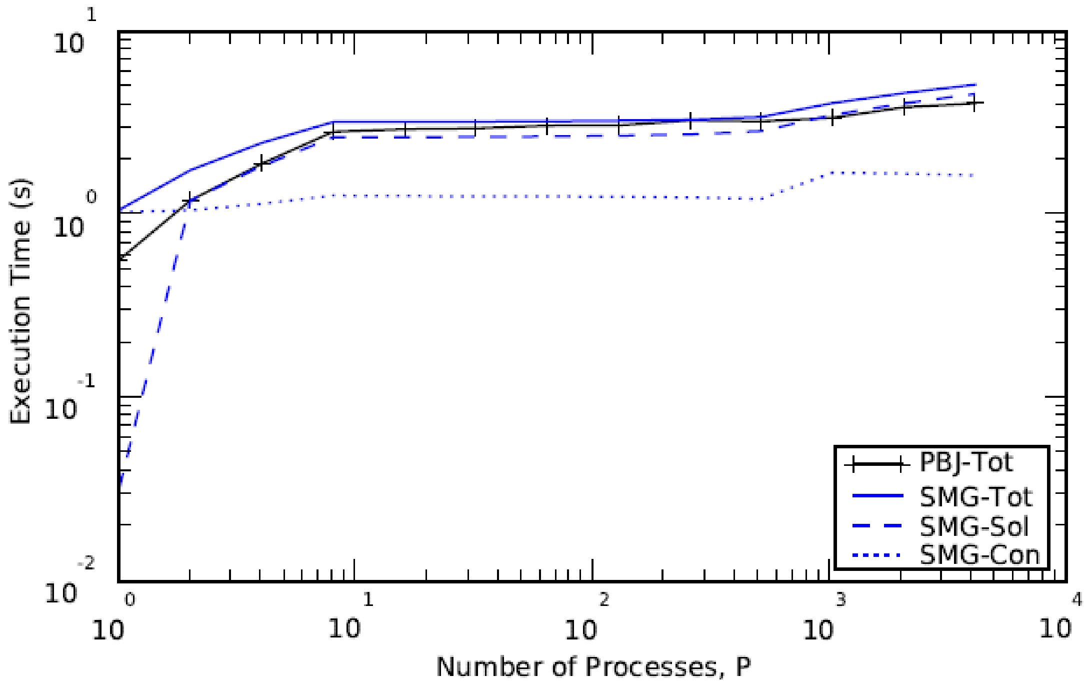


Fig. 3.7.7 JPF SMG weak scaling, Execution Time *vs*. *P*, *c*=0.9, *h*=10.0 cm.

The issue of diverging solutions worsens when the optical thickness is increased to *h* = 1.0 cm as shown in Fig. 3.7.9. Up to *P* = 512, the SMG method consumes fewer total iterations over all grids than the PBJ solution method, but the SMG curves are steeper than the PBJ curve. Unfortunately, the SMG method begins diverging once the number of stages increases to three. Similarly, the job that failed for the *h* = 0.1 cm case also had three stages. However, the fact that some jobs work for three stages and others do not is a likely indication that the SMG is fundamentally flawed for those problems, and the cause of the diverging iterations is not a coding error.

Interestingly, the *c* = 0.99, *h* = 10.0 cm case converges at all *P*, shown by the iteration count curves in Fig. 3.7.10. This is a benefit because it allows for improvement by modification of the SMG method (e.g., κ and the number of pre- and post-correction iterations), whereas the two optically thinner cases have not been confirmed to converge at all. Yet it is clear that the SMG method is interrupting the typical convergence path of PBJ as more fine-grid iterations are necessary in the SMG scheme than PBJ.

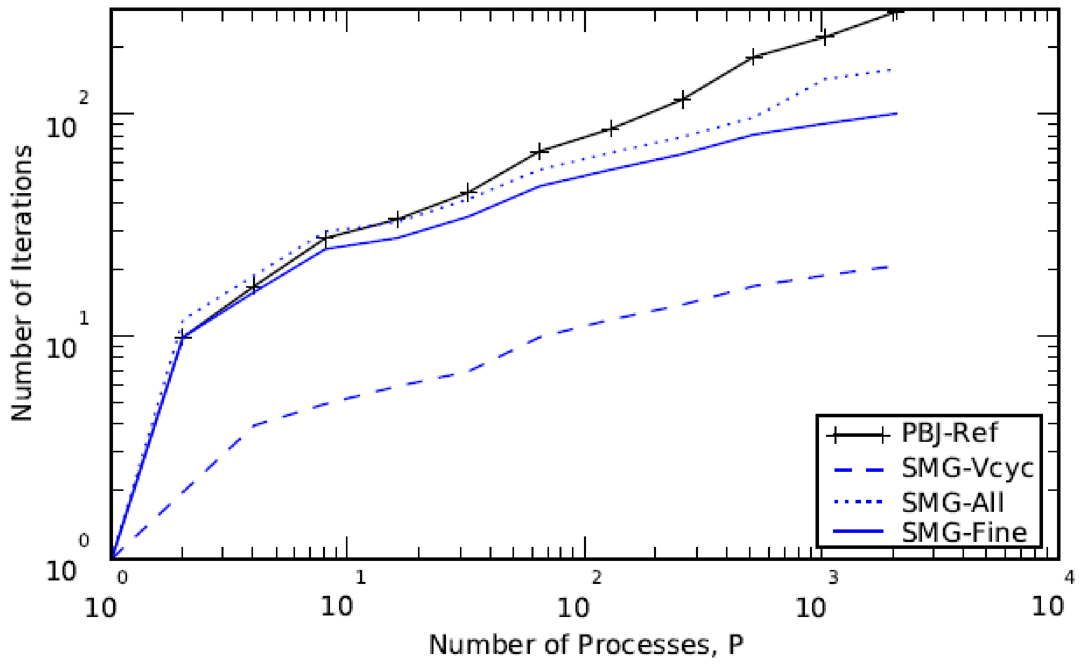


Fig. 3.7.8 JPF SMG weak scaling, Iterations *vs*. *P*, *c*=0.99, *h*=0.1 cm.

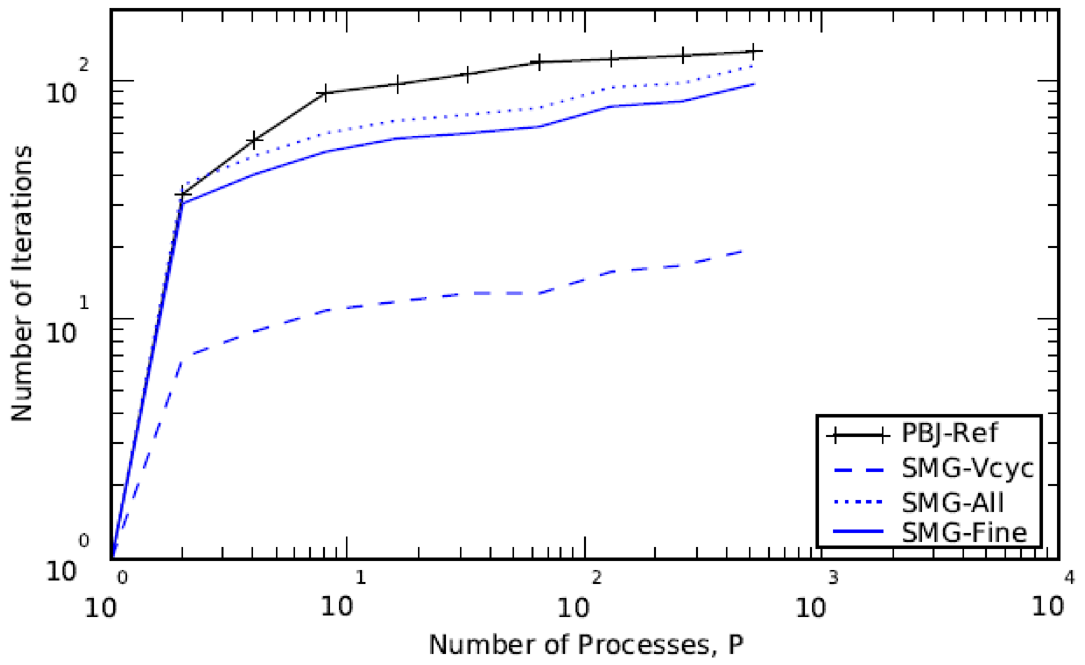


Fig. 3.7.9 JPF SMG weak scaling, Iterations *vs*. *P*, *c*=0.99, *h*=1.0 cm.

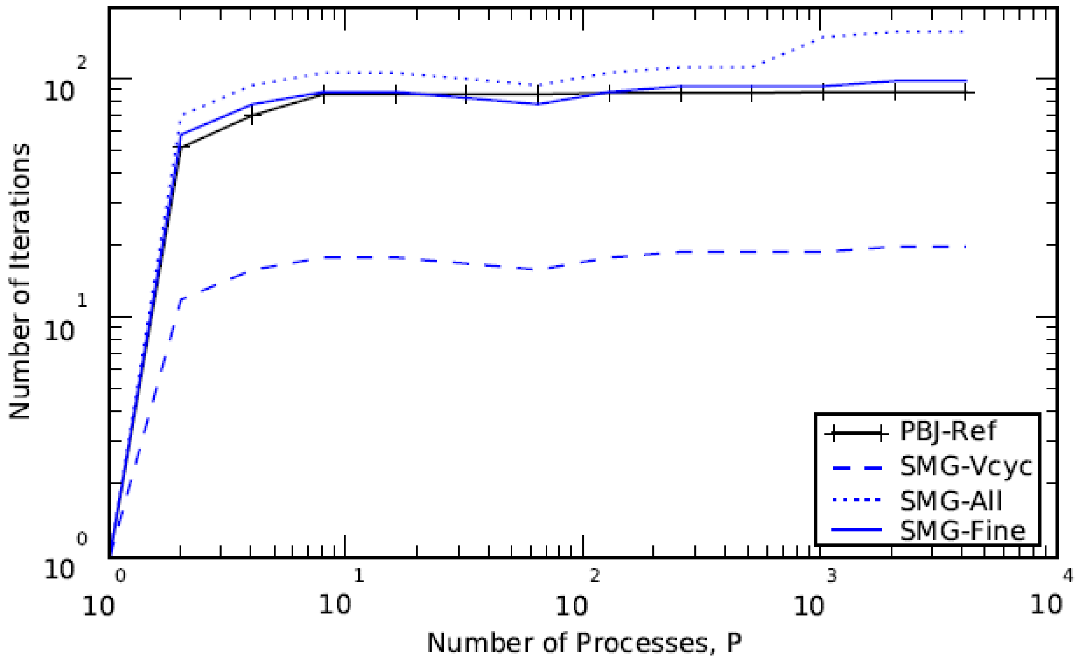


Fig. 3.7.10 JPF SMG weak scaling, Iterations *vs*. *P*, *c*=0.99, *h*=10.0 cm.

The execution time plots are given for the *c* = 0.99 cases in Figs. 3.7.11–3.7.13. Most impressively, the *h* = 0.1 cm case shown in Fig. 3.7.11 has an execution time at *P* = 2,048 that is over 50% faster than the PBJ solution. This is a better relative improvement over PBJ than was observed with red-black PGS. Moreover, it is a very favorable result because the SMG method is bending the total execution time curve to a flatter slope and achieving better scaling properties in the process. This is the most challenging system for the PBJ method because the sub-domains tend to stay tightly coupled even for problems with a large number of cells. The *h* = 1.0 cm, Fig. 3.7.12, exhibits a smaller gain in execution time up to *P* = 512 with the SMG method. Whether this case would continue to outperform PBJ if the problem correctly converged is of course unknown. Yet it is undeniable that for those problem sizes that do converge, the SMG method is benefiting the unaccelerated PBJ solution when the problem is optically thin. The *h* = 10.0 cm case, Fig. 3.7.13, consumes more wall-clock execution time with SMG than with straight PBJ but does converge at all values of *P*. Whether this case can be improved by modifying the SMG parameters requires further exploration outside the scope of this project.

3.7.3 Remarks

The PBJ method's parallel performance may potentially be improved with a spatial multigrid scheme similar to the one described and analyzed in this section. Some modifications may improve upon the presented results. For example, more extensive testing is necessary to determine if the parallel performance for optically thick problems could be improved by reducing the coarsening factor κ. The coarsening parameter utilized here may be too large, leading to cells that are also too large yielding poor approximations of the errors that are interpolated and used to correct the fine-grid solution. Moreover, further testing may reveal the cause of the divergent behavior observed for some of the *c* = 0.99, *h* = 0.1 cm and 1.0 cm cases.

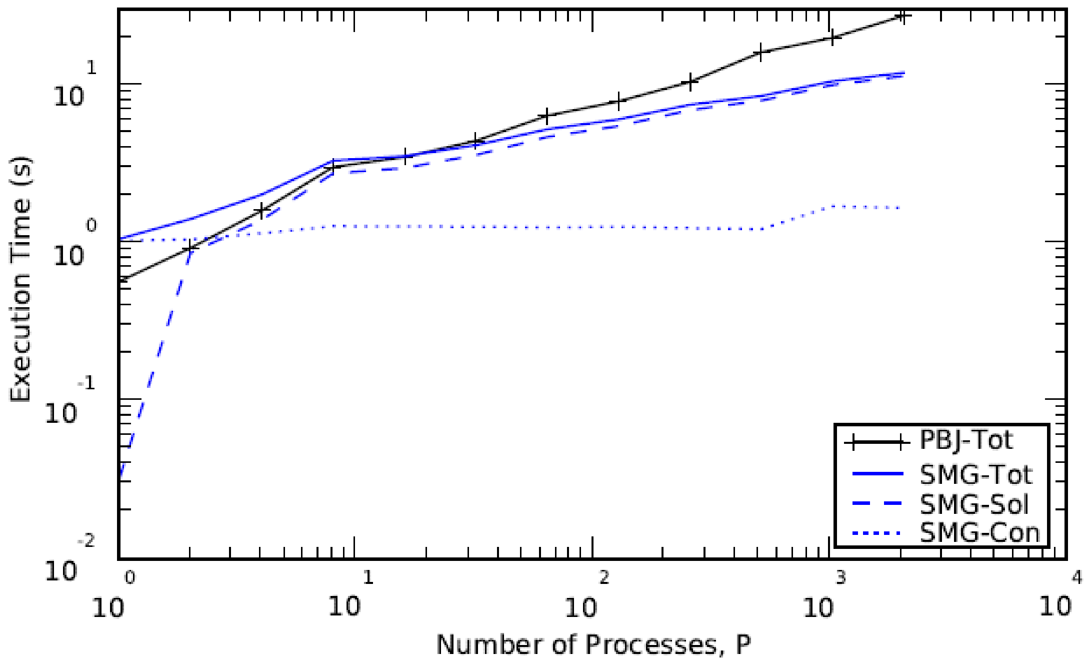


Fig. 3.7.11 JPF SMG weak scaling, Execution Time *vs*. *P*, *c*=0.99, *h*=0.1 cm.

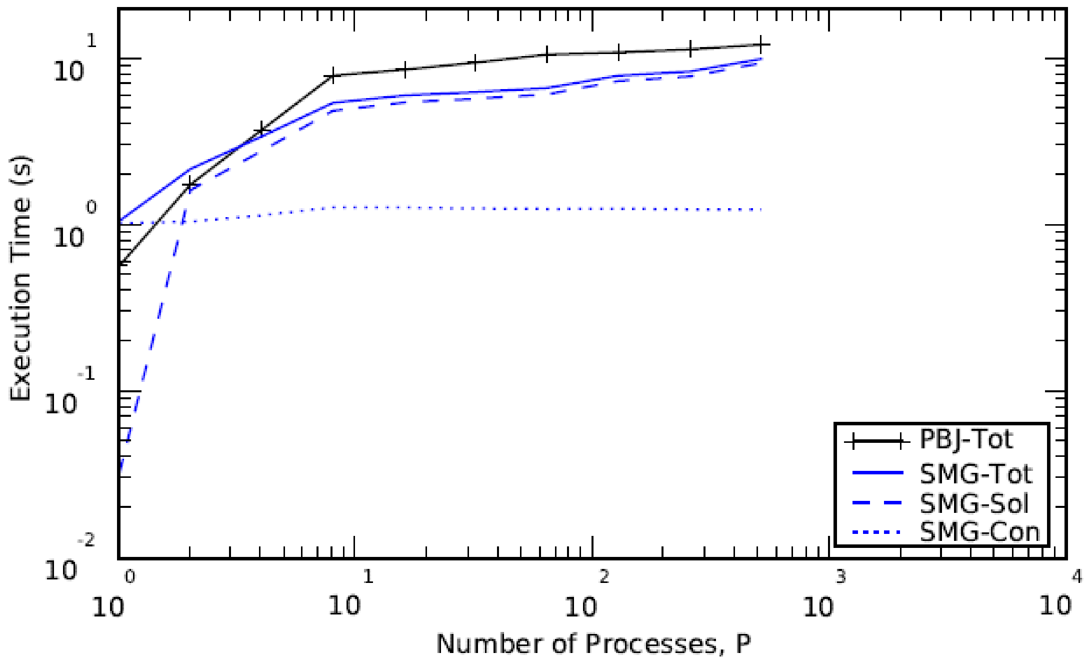


Figure 8.12 JPF SMG weak scaling, Execution Time *vs*. *P*, *c*=0.99, *h*=1.0 cm.

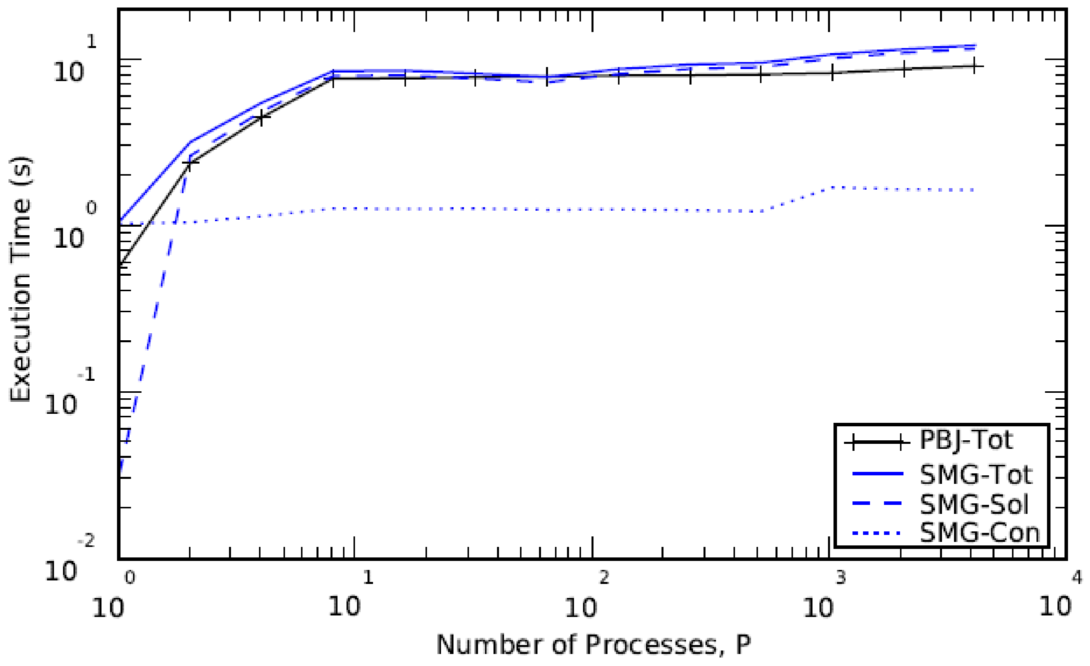


Fig. 3.7.13 JPF SMG weak scaling, Execution Time *vs*. *P*, *c*=0.99, *h*=10.0 cm.

If testing does not significantly improve the method, other considerations must be made. First, the restriction and interpolation operations may be poorly conceived and could benefit from a new approach. Alternatively, composing coarse-grid operators simply from new differential mesh sweeps may be ill-conceived. However, no research has been performed into how the coarse grid operators would be otherwise composed.

When the scattering ratio was high and the cells were optically thin, the SMG method did produce a significant improvement in the convergence rate and the overall execution time when the system properly converged. Moreover, the gains are mostly revealed when *P* is large, greater than 100. Such problems are ultimately the primary target of acceleration schemes developed for the ITMM, because these problems have the slowest convergence rate. Therefore, looking forward it must be determined why the method diverged for the *c* = 0.99, *h* = 0.1 and 1.0 cm cases and whether problems with larger domains would suffer similar challenges.

The SMG method must also be examined for strong scaling problems. It is currently unknown how the method will perform when the problem size is fixed. However, logistically the PBJ method is applicable to such studies, and it is anticipated that the method will be more successful at improving upon PBJ execution times for a large number of sub-domains as most of the results in this section also suggest.

The SMG method is currently applicable to heterogeneous domains, but the heterogeneities must be isolated in a manner that does not introduce the problem of mixing cells with different materials. Common homogenization procedures employ flux-volume weighting. In the case of the ITMM, using flux-volume weighting to alter nuclear data would require a new differential mesh sweep every *V*-cycle to properly update the operators. Volume-only weighting has not been tested as an alternative, but it could be considered. Nevertheless, improving the handling of heterogeneous domains is of lower priority than improving the method for more general problem parameters, namely lower values of *c* and large problem sizes in terms of number of cells and sub-domains.

Lastly, the SMG method could be applicable to the PGS and PGMRES methods. It should act as an acceleration tool for the PGS method. A test code has been developed for applying this SMG scheme to PGS but is still being tested. Preliminary results revealed an increase in the number of fine-grid iterations compared to the PGS method, possibly indicating a bug or a fundamental flaw in the SMG method. Moreover, the SMG method could act as a preconditioner for the PGMRES method in a manner similar to strategies described in the literature This approach has not been considered to date.

The SMG method described in this section shows promise for some cases and raises many new questions that could provide interesting research topics. Furthermore, other acceleration schemes based on multigrid methods—chiefly angular multigrid—may be developed that outperform the current implantation. Physical intuition suggests that SMG, if implemented in an optimized way, could improve the PBJ method by replacing indirect coupling via interfacial angular fluxes with direct coupling of coarsened regions. Yet a complete spectral analysis of these iterative methods would be necessary to verify this conjecture and thus constitutes a major objective for future work regarding the ITMM with parallel iterative solution techniques.

3.8 Task 8 – Contingency Tasks

Work completed under this contingency task was possible because of availability of funds for the student who was assigned to this work and who won a Computational Science Graduate Fellowship. His research comprised the development, implementation, verification, and testing of a new, highly efficient algorithm for constructing the ITMM operators. He also compared his new algorithm's performance to that of the differential sweep construction algorithm described in this report via straight numerical testing and a comprehensive performance model. He found his new algorithm to be substantially efficient over a significant range of problem sizes. Additionally, the new algorithm is far more amenable to extension to unstructured meshes where mesh sweeps are notorious for computational inefficiency.

The student earned his MS degree for this work and he published it in a paper presented at the Topical Meeting of the American Nuclear Society's Mathematics and Computation Division held in Sun Valley, Idaho, May 2013; see item 6 listed under the **List of Publications from the Project** section. The paper in PDF format is attached to this report and its Abstract is replicated below:

> *The Integral Transport Matrix Method (ITMM) has been shown to be an effective method for solving the neutron transport equation in large domains on massively parallel architectures. In the limit of very large number of processors, the speed of the algorithm, and its suitability for unstructured meshes, i.e. other than an ordered Cartesian grid, is limited by the construction of four matrix operators required for obtaining the solution in each sub-domain. The existing algorithm used for construction of these matrix operators, termed the differential mesh sweep, is computationally expensive and was developed for a structured grid. This work proposes the use of a new*

algorithm for construction of these operators based on the construction of a single, fundamental matrix representing the transport of a particle along every possible path throughout the sub-domain mesh. Each of the operators is constructed by multiplying an element of this fundamental matrix by two factors dependent only upon the operator being constructed and on properties of the emitting and incident cells. The ITMM matrix operator construction time for the new algorithm is demonstrated to be shorter than the existing algorithm in all tested cases with both isotropic and anisotropic scattering considered. While also being a more efficient algorithm on a structured Cartesian grid, the new algorithm is promising in its geometric robustness and potential for being applied to an unstructured mesh, with the ultimate goal of application to an unstructured tetrahedral mesh on a massively parallel architecture.

4. Bibliography

[1] B. R. Wienke and R. E. Hiromoto, "Parallel S_n Iteration Schemes," *Nuclear Science and Engineering*, **90**, 116 (1985).

[2] Y. Y. Azmy, "General Order Nodal Transport Methods and Application to Parallel Computing," *Transport Theory and Statistical Physics*, **22**(2 & 3), 359 (1993).

[3] A. Haghighat, "Angular Parallelization of a Curvilinear S_N Transport Theory Method," *Nuclear Science and Engineering*, **108**, 267 (1991).

[4] Y. Y. Azmy, "Multiprocessing for Neutron Diffusion and Deterministic Transport Methods," *Progress in Nuclear Energy*, **31**(3), 317 (1997).

[5] 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**, 287 (1996).

[6] 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, Oct. 5–9, 1997, Vol. I, p. 553, American Nuclear Society, La Grange Park, IL, USA (1997).

[7] M. Yavuz and E. W. Larsen, "Iterative Methods for Solving x-y Geometry S_N Problems on Parallel Architecture Computers," *Nuclear Science and Engineering*, **112**, 32 (1992).

[8] A. Haghighat and Y. Y. Azmy, "Parallelization of a Spherical S_N Algorithm Based on the Spatial Doman Decomposition," *Proceedings of the International Topical Meeting on Advances in Mathematics, Computations, and Reactor Physics*, Pittsburgh, PA, April 28–May 2, 1991, Vol. 1, Sec. 1.1, p. 3-1, American Nuclear Society, La Grange Park, IL, USA (1991).

[9] M. Rosa, J. Warsa, and T. Kelley, "Fourier Analysis of Cell-Wise Block-Jacobi Splitting in Two-Dimensional Geometry," *International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)*, Saratoga Springs, New York, USA, May 3–7, 2009, American Nuclear Society, La Grange Park, IL, USA, on CD-ROM (2009).

[10] R. S. Baker and K. R. Koch, "An S_n Algorithm for the Massively Parallel CM-200 Computer," *Nuclear Science and Engineering*, **128**, 312 (1998).

[11] 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**, 330 (2000).

[12] J. S. Warsa, K. Thompson, J. E. Morel, "Improving the Efficiency of Simple Parallel S_N Algorithms with Krylov Iterative Methods," *Transactions of the American Nuclear Society*, **89**, 449 (2003).

[13] R. E. Alcouffe, R. S. Baker, J. A. Dahl, S. A. Turner, and R.C. Ward, *PARTISN: A Time-Dependent, Parallel Neutral Particle Transport Code System*, LA-UR-08-07258, Los Alamos National Laboratory (2008).

[14] T. M. Evans, A. S. Stafford, R. N. Slaybaugh, and K. T. Clarno, "Denovo: A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE," *Nuclear Technology*, **171**, 171 (2010).

[15] James A. Fischer, "Comparison of Spatial and Angular Domain Decomposition Algorithms for Discrete Ordinates Transport Methods Using Parallel Performance Models," MS Thesis, The Pennsylvania State University (2003).

[16] 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*, Poços de Caldas Springs, MG, Brazil, August 18–22, 1997 (1997).

[17] "Wolfram *Mathematica*: Documentation Center," *Wolfram*, available online http://reference.wolfram.com/mathematica/guide/Mathematica.html, last accessed December 6, 2010 (2010).

[18] M. Rosa, Y. Y. Azmy, and J. E. Morel, "Properties of the S_N-Equivalent Integral Transport Operator in Slab Geometry and the Iterative Acceleration of Neutral Particle Transport Methods," *Nuclear Science and Engineering*, **162**, 234 (2009).

[19] R. J. Zerr, "Solution of the Within-Group Multidimensional Discrete Ordinates Transport Equations on Massively Parallel Architectures," PhD Thesis, The Pennsylvania State University, University Park, PA, USA (2011).

[20] 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).

[21] 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 24, 2009 (1999).

[22] "LION-XO PC Cluster", http://gears.aset.psu.edu/hpc/systems/lionxo/ , Last Accessed February 24, 2009 (2009).

[23] H. F. Jordan and G. Alaghband, *Fundamentals of Parallel Processing*, Pearson Education Inc., Upper Saddle River, NJ (2003).

[24] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd Ed., Pearson Education Limited, Harlow, England (2003).

[25] E. E. Lewis and J. F. Miller, Jr., *Computational Methods of Neutron Transport*, American Nuclear Society, La Grange Park, IL, USA (1993).

[26] J. J. Duderstadt and W. R. Martin, *Transport Theory*, John Wiley & Sons, Inc., New York, NY, USA (1979).

[27] R. J. Zerr and Y. Y. Azmy, "A Parallel Algorithm for Solving the Multidimensional Within-Group Discrete Ordinates Equations with Spatial Domain Decomposition," *Proceedings of PHYSOR 2010*, Pittsburgh, PA, USA, May 9–14, 2010, American Nuclear Society, LaGrange Park, IL, USA, on CD-ROM (2010).

[28] T. M. Evans, G. G. Davidson, and R. N. Slaybaugh, "Three-Dimensional Full Core Power Calculations for Pressurized Water Reactors," *Journal of Physics: Conference Series, SciDAC 2010*, Accepted (2010).

[29] Y. Y. Azmy, "Unconditionally Stable and Robust Adjacent-Cell Diffusive Preconditioning of Weighted-Difference Particle Transport Methods is Impossible," *Journal of Computational Physics*, **182**, 213 (2002).

[30] J. S. Warsa, T. A. Wareing, and J. E. Morel, "Krylov Iterative Methods and the Degraded Effectiveness of Diffusion Synthetic Acceleration for Multidimensional S_N Calculations in Problems with Material Discontinuities," *Nuclear Science and Engineering*, **147**, 218 (2004).

[31] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd Edition, SIAM, Philadelphia, PA, USA (1994).

[32] M. T. Heath, *Scientific Computing: An Introductory Survey*, 2nd Edition, McGraw-Hill, New York, NY, USA (2002).

5. List of Publications from the Project

1. R. Joseph Zerr, Yousry Y. Azmy, "A Parallel Algorithm for Solving the Integral Form of the Discrete Ordinates Equations", in the proceedings of the *International Conference on Advances in Mathematics, Computational Methods, and Reactor Physics*, Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, La Grange Park, IL (2009).

2. R. Joseph Zerr, Yousry Azmy, "A Parallel Algorithm for Solving the Multidimensional Within-Group Discrete Ordinates Equations with Spatial Domain Decomposition", in the proceedings of *PHYSOR* 2010 - *Advances in Reactor Physics to Power the Nuclear Renaissance*, Pittsburgh, Pennsylvania, May 9-14, 2010, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2010).

3. R. Joseph Zerr & Yousry Y. Azmy, "Improved Parallel Solution Techniques for the Integral Transport Matrix Method", in the proceedings of *International Conference on Mathematics*

and Computational Methods Applied to Nuclear Science and Engineering (*M&C* 2011), Rio de Janeiro, RJ, Brazil, May 8-12, 2011, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2011).

4. R. J. Zerr and Y. Y. Azmy, "Solution of the Within-Group Multidimensional Discrete Ordinates Transport Equations on Massively Parallel Architectures, invited, Mark Mills Award Winner", *Transactions of the American Nuclear Society* **105**, 429 (2011). [The 2011 American Nuclear Society Mark Mills Award Paper.]

5. Sebastian Schunert and Yousry Azmy, "A Verification Regime for the Spatial Discretization of the SN Transport Equations", in the proceedings of *PHYSOR 2012 Advances in Reactor Physics Linking Research, Industry, and Education*, Knoxville, Tennessee, USA, April 15-20, 2012, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2012).

6. Brian P. Powell and Yousry Y. Azmy, "An Advanced Algorithm for Construction of Integral Transport Matrix Method Operators Using Accumulation of Single Cell Coupling Factors", in the proceedings of *International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering* (*M&C* 2013), Sun Valley, Idaho, USA, May 5-9, 2013, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2013).

7. R. J. Zerr, "Solution of the Within-Group Multidimensional Discrete Ordinates Transport Equations on Massively Parallel Architectures," PhD Thesis, The Pennsylvania State University, University Park, Pennsylvania, USA (2011).

8. Brian P. Powell, "An Advanced Algorithm for Construction of Integral Transport Matrix Method Operators Using Accumulation of Single Cell Coupling Factors," MS Thesis, North Carolina State University, Raleigh, North Carolina, USA (2013).

U.S. DEPARTMENT OF ENERGY

Nuclear Energy

A Novel Algorithm for Solving the Multidimensional Neutron Transport Equation on Massively Parallel Architectures

OVERVIEW

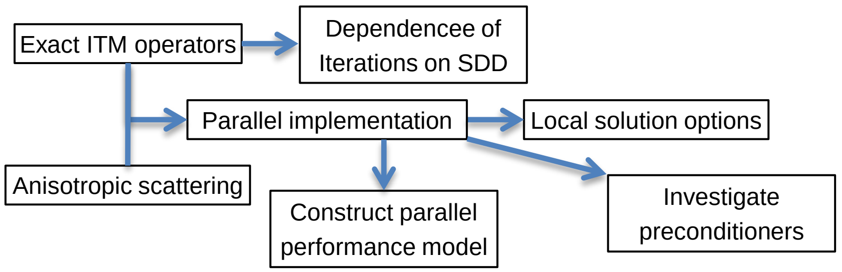
Purpose: Develop a new algorithm for solving the discrete ordinates equations in mutli-dimensional geometry without the intrinsically sequential mesh-sweep thereby making it suitable for massively parallel implementation thus enabling computationally efficient solution of large problems.

Objectives:

- ❖ Construct and verify the exact matrix operators
- ❖ Examine dependence of convergence rate on spatial decomposition
- ❖ Construct a preliminary parallel performance model
- ❖ Parallel Implementation of the new algorithms
- ❖ Evaluate options for solving local system
- ❖ Extend operators to anisotropic scattering
- ❖ Investigate preconditioners
- ❖ Contingency task: efficient ITMM construction extendible to unstructured meshes

IMPACT

Logical Path:


Exact ITM operators
Dependencee of Iterations on SDD
Parallel implementation
Local solution options
Anisotropic scattering
Construct parallel performance model
Investigate preconditioners

Outcomes:
This project delivers a novel algorithm for solving the discrete ordinates equations of neutron transport that is amenable to execution on massively parallel platforms employing millions of processors and with promising potential for unstructured meshes. We discovered new iterative behavior that is worthy of further examination.

DETAILS

Principal Investigator: Yousry Y. Azmy

Institution: North Carolina State University

Collaborators: N/A

Duration: 3 years (+1 year NCE)

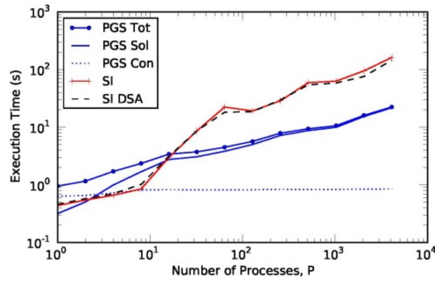
Total Funding Level: $350,000

TPOC: Keith Bradley

Federal Manager: Mark Schanfein

Workscope: AFM-1

PICSNE Workpackage #:
CFP 09-797



Weak scaling for Periodic Horizontal Layers on Jaguar

RESULTS

Status of Deliverables:

All tasks listed in the OVERVIEW section have been completed, including the Contingency Task. Deliverables have been met for all tasks, but further research is required to fully understand the iterative behavior discovered in task 2. Specific results:

- ❖ Equations for ITMM derived for subdomain BCs, anisotropic scattering, AHOT-N
- ❖ ITMM operator construction implemented & verified via numerical testing
- ❖ Performed preliminary spectral analysis to explain unexpected iterative behavior
- ❖ Performance model constructed & validated against measured code timing
- ❖ Parallel Block Jacobi (PBJ) & parallel Gauss-Seidel (PGS) versions of new algorithm implemented in PIDOTS code; executed on several platforms
- ❖ PIDOTS-PGS parallel performance compared to PARTISN on almost 33K cores of Jaguar supercomputer on up to 130M computational cells
- ❖ PIDOTS-PGS higher parallel speedup for optically thick, low scattering problems
- ❖ Multigrid acceleration developed & tested with mixed results
- ❖ New ITMM construction algorithm: faster & amenable to unstructured grids

Publications & Presentations:
See full list of eight publications in the Final Report on this project

International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)
Sun Valley, Idaho, USA, May 5-9, 2013, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2013)

AN ADVANCED ALGORITHM FOR CONSTRUCTION OF INTEGRAL TRANSPORT MATRIX METHOD OPERATORS USING ACCUMULATION OF SINGLE CELL COUPLING FACTORS

Brian P. Powell and Yousry Y. Azmy
North Carolina State University, Department of Nuclear Engineering
Burlington Engineering Labs, 2500 Stinston Drive, Raleigh, NC 27695
bppowell@ncsu.edu; yyazmy@ncsu.edu

ABSTRACT

The Integral Transport Matrix Method (ITMM) has been shown to be an effective method for solving the neutron transport equation in large domains on massively parallel architectures. In the limit of very large number of processors, the speed of the algorithm, and its suitability for unstructured meshes, i.e. other than an ordered Cartesian grid, is limited by the construction of four matrix operators required for obtaining the solution in each sub-domain. The existing algorithm used for construction of these matrix operators, termed the differential mesh sweep, is computationally expensive and was developed for a structured grid. This work proposes the use of a new algorithm for construction of these operators based on the construction of a single, fundamental matrix representing the transport of a particle along every possible path throughout the sub-domain mesh. Each of the operators is constructed by multiplying an element of this fundamental matrix by two factors dependent only upon the operator being constructed and on properties of the emitting and incident cells. The ITMM matrix operator construction time for the new algorithm is demonstrated to be shorter than the existing algorithm in all tested cases with both isotropic and anisotropic scattering considered. While also being a more efficient algorithm on a structured Cartesian grid, the new algorithm is promising in its geometric robustness and potential for being applied to an unstructured mesh, with the ultimate goal of application to an unstructured tetrahedral mesh on a massively parallel architecture.

Key Words: Neutron Transport, Matrix Operator, ITMM

1. INTRODUCTION

The foundations of the Integral Transport Matrix Method (ITMM) were first developed in 1992 by Hanebutte and Lewis [1], who first proposed using a “response matrix algorithm” to iteratively solve the transport equation. However, this method was limited to the response of a single cell sub-domain. As such, the method lacked the ability to be applied to sub-domains comprised of multiple cells due to the enormous number of iterations that would be required to achieve convergence on a solution.

As such, the method was not further developed until 1997 by Azmy [2], who first proposed a method for using full-domain operators, rather than single cell operators, to allow for a solution to the transport equation without the need for repetitive mesh sweeps that continue to dominate deterministic method solution algorithms. Azmy [3] continued to evolve the method in 1999 with the development of a specific algorithm for construction of the matrix operators required by this method.

Rosa, Azmy, and Morel [4] continued this work in 2009, examining spectral properties of the operators and further developing the algorithm for construction of the matrix operators on configurations with vacuum boundary conditions. Zerr [5] completed this work in 2011 by finishing the construction algorithm (termed the Differential Mesh Sweep) for all four matrix operators of the ITMM that are necessary to account for non-trivial incoming angular flux. Zerr also developed a code for parallel implementation of this algorithm in constructing the operators and iteratively solving the transport equation across sub-domains.

The work presented here is motivated by the desire to implement the ITMM on other geometries, e.g. unstructured grids. Zerr's complete algorithm was developed solely for a Cartesian mesh. The rise of unstructured tetrahedral mesh transport methods and codes as a means to a more accurate representation of geometric configurations typical of radiation transport problems requires adaptations of the ITMM for the application of this method to such a mesh. The Differential Mesh Sweep, although effective for constructing the matrix operators on a Cartesian grid, is computationally expensive and requires orderly mesh structure to perform well. As such, the need for a method of constructing the ITMM matrix operators that is both more computationally efficient and geometrically robust is evident.

The algorithm presented in this paper, although still for a Cartesian mesh at this stage, is promising in that it has demonstrated significantly faster matrix operator construction times and has the potential to be applied with relative ease to an unstructured tetrahedral mesh. This algorithm derives its efficiency from the fact that all transport of particles through a mesh, whether originating from a source or an incoming angular flux, follows the same paths from the emergent cell to the incident cell. This principle allows for the creation of a fundamental matrix for cell-to-cell transport, from which each of the matrix operators required for the ITMM can be constructed. It will therefore be termed the Fundamental Matrix Method (FMM). The derivation of this method first requires a quick review of the ITMM.

2 The Integral Transport Matrix Method

The basis of the ITMM lies in the neutron balance equation for a single cell in three dimensions [6]:

$$\begin{aligned}\varepsilon^x_{n,i,j,k}\psi_{n,i_{out},j,k} + \varepsilon^y_{n,i,j,k}\psi_{n,i,j_{out},k} + \varepsilon^z_{n,i,j,k}\psi_{n,i,j,k_{out}} + \psi_{n,i,j,k} \\ = c_{i,j,k}\phi_{i,j,k} + \sigma^{-1}_{t,i,j,k}q_{i,j,k} + \varepsilon^x_{n,i,j,k}\psi_{n,i_{in},j,k} + \varepsilon^y_{n,i,j,k}\psi_{n,i,j_{in},k} + \varepsilon^z_{n,i,j,k}\psi_{n,i,j,k_{in}}\end{aligned} \quad (1)$$

where,

$$\varepsilon^x_{n,i,j,k} = \frac{|\mu_n|}{\sigma_{t,i,j,k}\Delta x_i}, \text{AFyz} \quad (2)$$

AFyz stands for "analogously for y and z", and, by the discrete ordinates approximation with quadrature weights w_n and total number of angles D,

$$\phi_{i,j,k} = \sum_{n=1}^{D} w_n\psi_{n,i,j,k} \quad (3)$$

In the above equations, μ_n is the angular cosine with respect to the x-axis of the direction of particle travel along the n^{th} discrete ordinate. $\sigma_{t,i,j,k}$ is the macroscopic total interaction cross section of the material in cell i,j,k. Δx_i is the width of the cell in the x direction. $c_{i,j,k}$ is the scattering ratio, $\frac{\sigma_{s,i,j,k}}{\sigma_{t,i,j,k}}$, of the material in cell i,j,k , where $\sigma_{s,i,j,k}$ is the macroscopic scattering cross section in the same cell. $q_{i,j,k}$ is the distributed source in cell i,j,k. $\psi_{n,i_{out},j,k}$ is the angular flux along the n^{th} discrete ordinate leaving cell i,j,k out of the $x = constant$ face, with analogous definitions for angular flux leaving at the $y = constant$ and $z = constant$ faces. $\psi_{n,i_{in},j,k}$ is the angular flux traveling along the n^{th} discrete ordinate entering cell i,j,k in the $x = constant$ face, with analogous definitions for angular flux entering at the $y = constant$ and $z = constant$ faces. Finally, $\psi_{n,i,j,k}$ is the cell averaged angular flux in cell i,j,k of neutrons traveling along the n^{th} discrete ordinate. The diamond difference relation in all three directions is used to establish a closed matrix system of equations for the unknown fluxes on the left hand side of equation 1 [6],

$$\psi_{n,i,j,k} = \frac{1}{2}(\psi_{n,i_{in},j,k} + \psi_{n,i_{out},j,k}), \text{AFyz}, \tag{4}$$

resulting in [5]

$$\begin{bmatrix} \psi_{n,i,j,k} \\ \psi_{n,i,j,k_{out}} \\ \psi_{n,i,j_{out},k} \\ \psi_{n,i_{out},j,k} \end{bmatrix} = \Gamma_n \begin{bmatrix} c_{i,j,k}\phi^{p}_{i,j,k} + \sigma^{-1}_{t,i,j,k} q_{i,j,k} \\ \psi_{n,i,j,k_{in}} \\ \psi_{n,i,j_{in},k} \\ \psi_{n,i_{in},j,k} \end{bmatrix}, \tag{5}$$

where

$$\Gamma_n = \begin{bmatrix} j_\phi & k_{\phi,z} & k_{\phi,y} & k_{\phi,x} \\ j_{\psi,z} & k_{\psi,z\to z} & k_{\psi,y\to z} & k_{\psi,x\to z} \\ j_{\psi,y} & k_{\psi,z\to y} & k_{\psi,y\to y} & k_{\psi,x\to y} \\ j_{\psi,x} & k_{\psi,z\to x} & k_{\psi,y\to x} & k_{\psi,x\to x} \end{bmatrix}. \tag{6}$$

The Γ matrix is then a set of coupling factors (named here according to their function, which will be clarified later) between the distributed (fixed and scattering) source, incoming angular fluxes, and outgoing angular fluxes.

All of the previous equations provide the relationships between the incoming and outgoing angular flux and the scalar flux for only a single cell. When considering a multi-cell sub-domain, the relationship between the cells is simply an extrapolation of

$$\psi_{n,i+1_{in},j,k} = \psi_{n,i_{out},j,k}, \text{AFyz}. \tag{7}$$

Considering a multi-cell domain with vacuum boundary conditions in a source iteration scheme, the system reduces to [5]

$$\boldsymbol{\phi^v} = \boldsymbol{A}(\boldsymbol{C}\boldsymbol{\phi^p} + \boldsymbol{\Sigma_t^{-1}}\boldsymbol{q}). \tag{8}$$

The vectors $\boldsymbol{\phi^v}$ and $\boldsymbol{\phi^p}$ are the new and previous iterates of the scalar flux, respectively. The vector $\boldsymbol{q}$ is the fixed source. $\boldsymbol{C}$ is the scattering ratio diagonal matrix and $\boldsymbol{\Sigma_t^{-1}}$ is the inverse total cross section diagonal matrix. $\boldsymbol{A}$ is then a coefficient matrix constructed from elements of Γ which relates the previous scalar flux iterate to the new scalar flux iterate. Partially differentiating equation 8 with respect to $\boldsymbol{\phi^p}$ yields the iteration Jacobian Matrix,

$$\frac{\partial \boldsymbol{\phi^v}}{\partial \boldsymbol{\phi^p}} = \boldsymbol{AC}. \tag{9}$$

$\boldsymbol{AC}$ was denoted by Zerr [5] as $\boldsymbol{J_\phi}$. Further manipulation, considering the effect of incoming angular fluxes at the boundaries of the sub-domain, and setting $\boldsymbol{\phi^v} = \boldsymbol{\phi^p} = \boldsymbol{\phi^\infty}$, where the superscript ∞ indicates the converged solution, results in [5],

$$\boldsymbol{\phi^\infty} = (\boldsymbol{I} - \boldsymbol{J_\phi})^{-1}\boldsymbol{J_\phi}\boldsymbol{\Sigma_s^{-1}}\boldsymbol{q} + (\boldsymbol{I} - \boldsymbol{J_\phi})^{-1}\boldsymbol{K_\phi}\boldsymbol{\psi^\infty_{in}}, \tag{10}$$

where $\boldsymbol{\psi^\infty_{in}}$ is a vector containing all incoming angular fluxes to the sub-domain and $\boldsymbol{K_\phi}$ is a matrix operator constructed from the elements of Γ which defines the effect of the incoming angular flux on the scalar flux in each cell of the sub-domain. The dimension of $\boldsymbol{\psi^\infty_{in}}$ is then the number of faces comprising part of the exterior of the sub-domain multiplied by the number of incoming ordinates to each surface. $\boldsymbol{K_\phi}$ has the same number of columns as the dimension of $\boldsymbol{\psi^\infty_{in}}$ and the number of rows is equal to the number of cells in the sub-domain. The construction of $\boldsymbol{K_\phi}$ will be described in the next section.

Both the effect of a fixed source ($\boldsymbol{J_\phi}$) and of an incoming angular flux ($\boldsymbol{K_\phi}$) on the scalar flux in each cell in the sub-domain have now been accounted for. The next consideration, therefore, is the calculation of the outgoing angular flux from a sub-domain consisting of two components: Upon convergence, the outgoing angular flux satisfies [5],

$$\boldsymbol{\psi}_{\boldsymbol{out}}^{\infty} = \boldsymbol{J}_{\psi}\boldsymbol{\phi}^{\infty} + \boldsymbol{K}_{\psi}\boldsymbol{\psi}_{\boldsymbol{in}}^{\infty} \tag{11}$$

where ${\boldsymbol{\psi}_{\boldsymbol{in}}}^{\infty}$ and $\boldsymbol{\phi}^{\infty}$ have been previously defined, $\boldsymbol{J}_{\psi}$ is a matrix operator constructed from the elements of Γ with dimensions that are the transpose of the previously defined matrix operator, $\boldsymbol{K}_{\phi}$, and $\boldsymbol{K}_{\psi}$ is a square matrix operator (assuming the typical reflective symmetry of discrete ordinates) constructed from the elements of Γ with a dimension equal to the dimension of the vector $\boldsymbol{\psi}_{\boldsymbol{in}}^{\infty}$.

As the construction of each matrix operator can become intractable, it is useful to summarize their definitions, shown in Table I, below.

Table I: Matrix Operator Definitions

Matrix Operator	Definition
$\boldsymbol{J}_{\phi}$	The effect of the cell averaged distributed source in each cell on the cell averaged uncollided scalar flux in all cells
$\boldsymbol{J}_{\psi}$	The effect of the cell averaged distributed source in each cell on the outgoing angular flux on all external faces
$\boldsymbol{K}_{\phi}$	The effect of the incoming angular flux on each external face on the cell averaged uncollided scalar flux in all cells
$\boldsymbol{K}_{\psi}$	The effect of the incoming angular flux on each external face on the outgoing angular flux on all external faces

2.1 Consideration of Anisotropic Scattering

Consideration of anisotropic scattering in equations 10 and 11 does not alter the equations. It does, however, significantly alter the contents of the vectors ϕ^{∞} and ψ_{in} and therefore the construction of the four matrix operators which correspond to those vectors. A lengthy derivation by Zerr [5], avoided here for brevity, provides that the scattering source is given by:

$$q_s = \sum_{l=0}^{L}\sum_{m=0}^{l} \sigma_{sl}(2-\delta_{m0})[Y_{lm}^{e}(\hat{\Omega})\varphi_l^m + Y_{lm}^{o}(\hat{\Omega})\vartheta_l^m]. \tag{12}$$

In the case of isotropic scattering (i.e. $L = 0$), it can be seen that the scattering source reduces to

$$q_s = \sigma_{s,0}\phi_0^o \tag{13}$$

as expected. The formation of the scattering source in the anisotropic manner, however, requires significant alterations to the matrix equation 5.

$$\begin{bmatrix} \psi_{n,i,j,k} \\ \psi_{n,i,j,k_{out}} \\ \psi_{n,i,j_{out},k} \\ \psi_{n,i_{out},j,k} \end{bmatrix} = \Gamma_{anis} \begin{bmatrix} \sigma_{s0}\varphi_0^0 + q_{00}^e \\ \sigma_{s1}\varphi_1^0 + q_{10}^e \\ \sigma_{s1}\varphi_1^1 + q_{11}^e \\ \sigma_{s1}\vartheta_1^1 + q_{11}^o \\ \vdots \\ \sigma_{sL}\vartheta_L^L + q_{LL}^o \\ \psi_{n,i,j,k_{in}} \\ \psi_{n,i,j_{in},k} \\ \psi_{n,i_{in},j,k} \end{bmatrix}. \tag{14}$$

The Γ_{anis} matrix has dimensions 4 x $((L+1)^2+3)$ and is a set of coupling factors between each angular moment of the scattering plus fixed source, incoming fluxes to the cell, and outgoing face angular fluxes. The effect of including anisotropic scattering on the dimensions of each matrix operator and, ultimately, the effect on construction algorithm performance will be examined in section 4.

An algorithm for the construction of the four matrix operators, $\boldsymbol{J_\phi}$, $\boldsymbol{J_\psi}$, $\boldsymbol{K_\phi}$, and , $\boldsymbol{K_\psi}$, was developed by Zerr [5] and termed the differential mesh sweep. The algorithm developed in this work is intended to improve upon the differential mesh sweep by accomplishing two goals: Be more geometrically robust and less computationally expensive. In consideration of these goals, this algorithm was developed on the foundation of the matrix equation 5.

3 The Fundamental Matrix Method

Beginning with equation 5, partial derivatives are taken with respect to each incoming angular flux component and the cell averaged scalar flux.

First, partially differentiating equation 5 with respect to the cell averaged scalar flux:

$$\begin{bmatrix} \frac{\partial\psi_{n,i,j,k}}{\partial\phi^p_{i,j,k}} \\ \frac{\partial\psi_{n,i,j,k_{out}}}{\partial\phi^p_{i,j,k}} \\ \frac{\partial\psi_{n,i,j_{out},k}}{\partial\phi^p_{i,j,k}} \\ \frac{\partial\psi_{n,i_{out},j,k}}{\partial\phi^p_{i,j,k}} \end{bmatrix} = \begin{bmatrix} j_{\phi_{i,j,k}} c_{i,j,k} \\ j_{\psi,z_{i,j,k}} c_{i,j,k} \\ j_{\psi,y_{i,j,k}} c_{i,j,k} \\ j_{\psi,x_{i,j,k}} c_{i,j,k} \end{bmatrix}. \tag{15}$$

Partially differentiating equation 5 with respect to the incoming angular flux in the x direction and analogously in the y and z directions:

$$\begin{bmatrix} \frac{\partial\psi_{n,i,j,k}}{\partial\psi_{n,i_{in},j,k}} \\ \frac{\partial\psi_{n,i,j,k_{out}}}{\partial\psi_{n,i_{in},j,k}} \\ \frac{\partial\psi_{n,i,j_{out},k}}{\partial\psi_{n,i_{in},j,k}} \\ \frac{\partial\psi_{n,i_{out},j,k}}{\partial\psi_{n,i_{in},j,k}} \end{bmatrix} = \begin{bmatrix} k_{\phi,x_{i,j,k}} \\ k_{\psi,x\to z_{i,j,k}} \\ k_{\psi,x\to y_{i,j,k}} \\ k_{\psi,x\to x_{i,j,k}} \end{bmatrix}, \text{AFyz.} \tag{16}$$

To create a general framework for the response of the angular flux in one cell of a domain to another cell, only the bottom three equations of matrix equation 16 and the analogous equations in the y and z directions are used. As such, only the transport of the angular flux is being considered. These equations, however, also only consider a single cell system. To consider an entire domain, these single cell operators need to be accumulated along every possible path of particles from the starting to destination cells. This accumulation of values representing the response of the angular flux in one cell of a domain to another cell is termed the fundamental transport matrix and denoted by $\boldsymbol{F}$. The elements of $\boldsymbol{F}$ are represented by

$$F_{n(i,j,k)(i',j',k'),di,df} = \sum_{p=1}^{P} \prod_{m=1}^{M} k_{\psi,d1_{m,p} \to d2_{m,p},n,m,p}, \tag{17}$$

where p is a possible path from cell i', j', k' to cell i, j, k and di and df are the incident and emergent constant faces (x, y, or z) of the element of $\boldsymbol{F}$. The total number of possible paths is P, a value which depends on the geometry of the domain, and the total number of cells along each respective path is M. The subscripts of k_ψ, $d1_{m,p}$ and $d2_{m,p}$ are either x, y, or z depending on which constant face the particle is incident on and emergent at, respectively, for cell m of path p. For the first cell of path p, $d1_{1,p} = di$, and for the last cell of path p, $d2_{M,p} = df$. $k_{\psi,d1_{m,p} \to d2_{m,p},n,m,p}$ is then the value of k_ψ from face $d1$ to face $d2$ in cell m along path p for the n^{th} discrete ordinate.

For an angular flux emergent from the face of one cell and incident on the face of another cell in a three dimensional system, there are nine distinct elements of $\boldsymbol{F}$ resulting from three possible emergent faces and three possible incident faces. To accumulate values into each ITMM operator, the only remaining calculation (after the calculation of the appropriate element of $\boldsymbol{F}$) is to multiply $F_{n(i,j,k)(i',j',k'),di,df}$ by the respective values it is required to represent for the emergent and incident cells. Recalling that the emergent cell is i', j', k' and the incident cell is i, j, k, the respective operators require multiplication of $F_{n(i,j,k)(i',j',k'),di,df}$ by the single cell operators in the manner shown in Table II.

Table II: Emergent and Incident Multipliers of $F_{n(i,j,k)(i',j',k'),di,df}$ for Construction of ITMM Operators

Operator	Emergent Cell Multiplier	Incident Cell Multiplier
$\boldsymbol{J_\phi}$	$j_{\psi,di,i',j',k'} c_{i',j',k'}$	$k_{\phi,df,i,j,k}$
$\boldsymbol{J_\psi}$	$j_{\psi,di,i',j',k'} c_{i',j',k'}$	$k_{\psi,d1 \to df,i,j,k}$
$\boldsymbol{K_\phi}$	$k_{\psi,d1 \to di,i',j',k'}$	$k_{\phi,df,i,j,k}$
$\boldsymbol{K_\psi}$	$k_{\psi,d1 \to di,i',j',k'}$	$k_{\psi,d1 \to df,i,j,k}$

The matrix operator $\boldsymbol{J_\phi}$, of dimensions (I x J x K) x (I x J x K), is constructed as

$$\boldsymbol{J_\phi} = \begin{bmatrix} j_{\phi,1,1,1} c_{1,1,1} & \cdots & j_{\psi,I,J,K} c_{I,J,K} F_{n(1,1,1)(I,J,K)} k_{\phi,1,1,1} \\ \vdots & \ddots & \vdots \\ j_{\psi,1,1,1} c_{1,1,1} F_{n(I,J,K)(1,1,1)} k_{\phi,I,J,K} & \cdots & j_{\phi,I,J,K} c_{I,J,K} \end{bmatrix}, \tag{18}$$

where

$$\begin{aligned} j_{\psi,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k')}k_{\phi,(i,j,k)} &= j_{\psi,x,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),x,x}k_{\phi,x,(i,j,k)}+ \\ & j_{\psi,x,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),x,y}k_{\phi,y,(i,j,k)} + j_{\psi,x,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),x,z}k_{\phi,z,(i,j,k)}+ \\ & j_{\psi,y,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),y,x}k_{\phi,x,(i,j,k)} + j_{\psi,y,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),y,y}k_{\phi,y,(i,j,k)}+ \\ & j_{\psi,y,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),y,z}k_{\phi,z,(i,j,k)} + j_{\psi,z,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),z,x}k_{\phi,x,(i,j,k)}+ \\ & j_{\psi,z,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),z,y}k_{\phi,y,(i,j,k)} + j_{\psi,z,(i',j',k')}c_{i',j',k'}F_{n(i,j,k)(i',j',k'),z,z}k_{\phi,z,(i,j,k)}. \end{aligned} \quad (19)$$

The matrix operators $\boldsymbol{J_\psi}$, $\boldsymbol{K_\phi}$, and $\boldsymbol{K_\psi}$ are constructed analogously. It is important to note that diagonal elements of $\boldsymbol{J_\phi}$ do not use an element of $\boldsymbol{F}$ in their construction. This is because the diagonal elements represent the effect of the cell averaged distributed source in a cell on the cell averaged scalar flux in the same cell. Since there is no transport to account for between cells, no element of $\boldsymbol{F}$ is used and the single cell operator j_ϕ is used instead of j_ψ on the diagonal elements.

In essence, we have consolidated the expensive, recursive calculations required to compute the four ITMM operators into the computation of $F_{n(i,j,k)(i',j',k')}$, followed by a straightforward and computationally cheap pre-fix and post-fix operations indicated in equation 18 and the analogous relations.

In comparison to the to the FMM, the Differential Mesh Sweep (DMS) requires twelve intermediate matrices in order to populate the four ITMM matrix operators. These are the three matrices with values representing the outgoing angular flux from every cell in the x, y, and z directions and the nine matrices representing the effect of incoming angular flux in each direction on the outgoing angular flux in each direction. These values are then used to create the four ITMM matrix operators. The FMM uses only a single intermediate matrix, $\boldsymbol{F}$, from which all the elements of the four ITMM matrix operators are created in a relatively simple manner.

The main reason that this difference is important is the time required for memory access. As each element of $\boldsymbol{F}$ is created, it is promptly operated on to create the corresponding elements of the four ITMM matrix operators, allowing for the element of $\boldsymbol{F}$ to remain in the memory cache for the necessary operations and then be discarded without another access. The DMS, however, creates the necessary twelve intermediate matrices and places them into memory, requiring that they be accessed at a later time to create the four ITMM matrix operators.

Another advantage of the FMM compared to the DMS is the nature of the mesh sweep. The DMS uses a single mesh sweep (per angle) to calculate the the required values to populate the aforementioned twelve matrices. In comparison, the FMM conducts a sweep from each cell of the mesh to all other cells. While seemingly a disadvantage for the FMM due to the quantity of sweeps required, the calculations are more simple in nature because only elements of $\boldsymbol{F}$ are being created in the sweep. The FMM, in this manner, is also more geometrically robust in that only knowledge of the spatial relation to the adjacent cells is required to create the elements of $\boldsymbol{F}$.

4 Results and Performance Model

4.1 Performance Model

It is useful to be able to predict run times for scaling purposes. As such, a performance model has been developed to serve this purpose. The performance of the construction algorithm in terms of computational time is dependent upon three variables: The number of cells in the sub-domain, N, the order of anisotropy, L, and the number of angles, D. The performance model was determined based on analysis of the operator construction subroutines, particularly the number of calculations required to build each operator.

First, consider the construction of the fundamental matrix $\boldsymbol{F}$. As previously demonstrated, all the ITMM operators are constructed through multiplications to elements of $\boldsymbol{F}$. $\boldsymbol{F}$ must therefore contain the basis of every possible element of each of the operators. The performance model of the construction of $\boldsymbol{F}$ involves three terms: The construction of elements of $\boldsymbol{F}$ along a row or a column intersecting the starting cell ($3a_1DN^{4/3}$), along planes intersecting the starting cell ($3a_2D(N^{5/3} - N^{4/3})$), and through the entire sub-domain from the starting cell ($a_3D(N^2 - 3N^{5/3} - 3N^{4/3})$), where a_i ($i = 1, 2, 3, ...$) are measures of the time consumed in completing a single instance of the corresponding instructions listed above. These terms are added together to create the performance model for constructing the fundamental matrix.

$$t_F = 3a_1DN^{4/3} + 3a_2D(N^{5/3} - N^{4/3}) + a_3D(N^2 - 3N^{5/3} - 3N^{4/3}) \tag{20}$$

The number of calculations for each of the matrix operators is a function of the number of cells involved in the operator construction, the number of angles, and the number of angular moments. The number of angular moments, represented here as H, is calculated by $H = (L + 1)^2$. While the number of calculations for each operator requires a multiplication by D to account for the number of angles, the use of H differs between the operators. $\boldsymbol{K_\psi}$ is the only operator completely independent of the scalar flux in any cell and, using Table I as a reference for the number of cells involved in the calculation, the performance model for $\boldsymbol{K_\psi}$ is

$$t_{\boldsymbol{K_\psi}} = a_4DN^{4/3}. \tag{21}$$

$\boldsymbol{J_\psi}$ and $\boldsymbol{K_\phi}$ both involve the cell averaged scalar flux on one end of the calculation. As such, the number of calculations required for these operators is multiplied by H, resulting in

$$t_{\boldsymbol{J_\psi}+\boldsymbol{K_\phi}} = a_5DN^{5/3}H. \tag{22}$$

$\boldsymbol{J_\phi}$ requires consideration of the scalar flux in both the initial cell and the final cell. Therefore, the number of calculations is multiplied by H^2. In the algorithm, several calculations are conducted in a single moment loop to improve computational efficiency. Hence the need for another term multiplied by only H but the same number of cells, N^2. These terms comprise the performance model for the calculation of $\boldsymbol{J_\phi}$,

$$t_{\boldsymbol{J_\phi}} = a_6DN^2H + a_7DN^2H^2. \tag{23}$$

The total construction time for the operators is then

$$t_{total} = t_F + t_{\boldsymbol{K_\psi}} + t_{\boldsymbol{J_\psi}+\boldsymbol{K_\phi}} + t_{\boldsymbol{J_\phi}}. \tag{24}$$

Substituting the previously defined terms results in

$$\begin{aligned} t_{total} = 3a_1DN^{4/3} + 3a_2D(N^{5/3} - N^{4/3}) + a_3D(N^2 - 3N^{5/3} - 3N^{4/3}) \\ + a_4DN^{4/3} + a_5DN^{5/3}H + a_6DN^2H + a_7DN^2H^2. \end{aligned} \tag{25}$$

In order to determine the values of the constants in equation 25, timed runs of the code with varying number of cells, number of angles, and order of anisotropy were conducted.

4.2 Timing Results

The following tables (III for L=0, IV for L=1, V for L=3) show the measured execution times of these trials. Each time shown is the average measured execution time across ten runs of the code on a 3.7 GHz processor. “FMM” designates the Fundamental Matrix Method of constructing the matrix operators, outlined in this work, and “DMS” designates the Differential Mesh Sweep method developed by Zerr [5]. As can be seen in each of these tables, the Fundamental Matrix Method significantly outperforms the Differential Mesh Sweep in all tested cases.

Table III: Matrix Operator Construction Time (s) for L=0 (isotropic)

Method	N	S_4	S_8	S_{12}	S_{16}
FMM	64	.0088	.0148	.0268	.0384
DMS	64	.0156	.0356	.0624	.1016
FMM	216	.0416	.0876	.1688	.2768
DMS	216	.0596	.1844	.3820	.6572
FMM	512	.1952	.4764	.9408	1.780
DMS	512	.2468	.8149	1.711	2.925
FMM	1000	.6588	1.507	3.961	6.722
DMS	1000	.8213	2.735	5.651	9.849

Table IV: Matrix Operator Construction Time (s) for L=1

Method	N	S_4	S_8	S_{12}	S_{16}
FMM	64	.0140	.0260	.0528	.0868
DMS	64	.0212	.0576	.1116	.1920
FMM	216	.0736	.1892	.4040	.7388
DMS	216	.1296	.4076	.8493	1.467
FMM	512	.4048	1.192	2.871	5.118
DMS	512	.6880	2.273	4.836	8.263
FMM	1000	1.245	4.085	9.172	16.22
DMS	1000	2.242	7.455	15.60	27.04

Table V: Matrix Operator Construction Time (s) for L=3

Method	N	S_4	S_8	S_{12}	S_{16}
FMM	64	.0396	.1148	.2528	.4668
DMS	64	.1036	.3704	.6608	1.110
FMM	216	.3136	1.045	2.193	4.332
DMS	216	.5936	1.965	4.078	6.970
FMM	512	1.944	6.961	13.62	28.63
DMS	512	9.815	30.65	66.90	113.7
FMM	1000	7.394	25.01	45.05	79.77
DMS	1000	12.16	40.46	85.16	147.8

In order to determine the values of the model constants of equation 25, a fitting function was applied to the timing data for the Fundamental Matrix Method in Tables III - V. This function uses a least squares algorithm to determine the minimal residual among all the data points by converging on an optimal value of the constants. This procedure results in the values shown in Table VI for the constants, applied to equation 25. These constants are accurate for the measured data points on a 3.7 GHz processor. Any change in the system running the code will change the constants, but the model will remain accurate.

Table VI: Least Squares Fit Constant Values for Equation 25

Constant	Value
a_1	$3.259 \text{ x } 10^{-10}$
a_2	$6.361 \text{ x } 10^{-9}$
a_3	$2.693 \text{ x } 10^{-8}$
a_4	$7.411 \text{ x } 10^{-10}$
a_5	$2.751 \text{ x } 10^{-9}$
a_6	$5.024 \text{ x } 10^{-9}$
a_7	$6.813 \text{ x } 10^{-10}$

This is now the complete performance model of the FMM algorithm. Figures 1 through 3 show the performance model plotted against the measured data points.

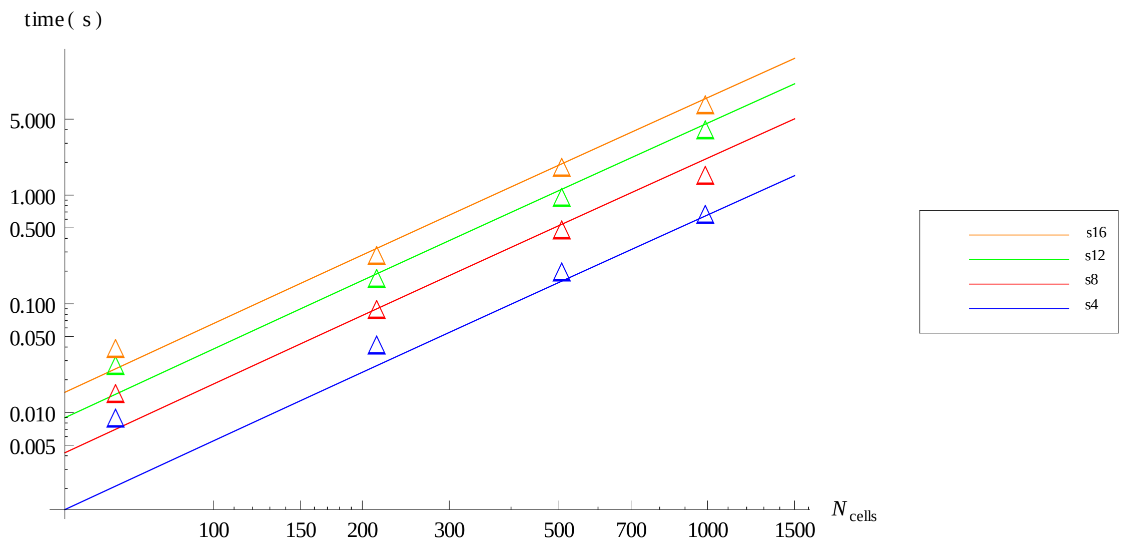


Figure 1: Performance Model (lines) and Measured Matrix Operator Construction Times (triangles) for L=0 (isotropic)

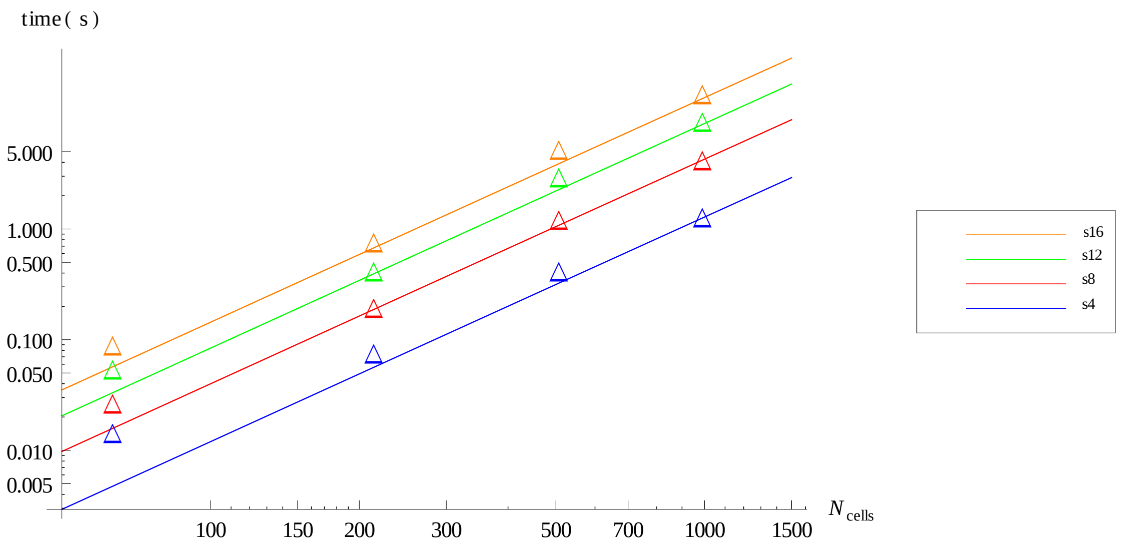


Figure 2: Performance Model (lines) and Measured Matrix Operator Construction Times (triangles) for L=1

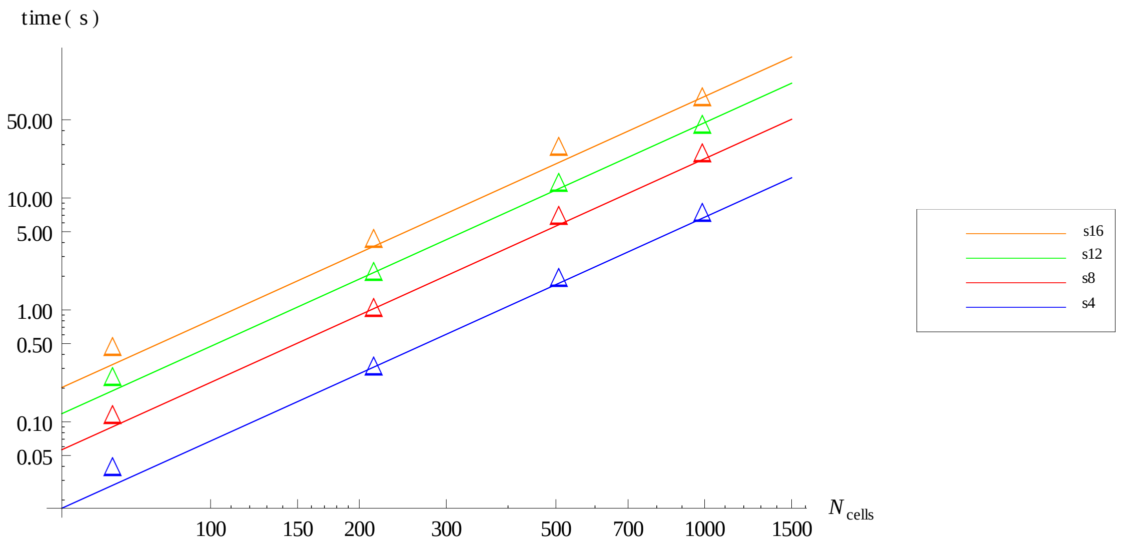


Figure 3: Performance Model (lines) and Measured Matrix Operator Construction Times (triangles) for L=3

Errors in the fit of the performance model to the data points can be attributed to several causes. First among these is the general fit to the entire data set for the determination of the constants. Any data point that could be considered somewhat aberrant can damage the overall fit and compound errors in conforming to individual data sets. Second would be the effect on execution time from the frequency of memory cache hits and misses. Each miss can increase the overall time while each hit can reduce it. Given the sheer number of memory accesses required for the FMM algorithm, any such problem could cause significant variation in the data points. The last source of error would be background processes on the system. While each data point is an average of ten code executions, the averaging does not account for background processes which may slow the performance of the algorithm. Considering these sources of error, the performance model trends, if not exactly predicts, the execution time of the FMM algorithm. The major error visible in the plots above is that the model consistently under-predicts the execution time for small N. This result is both understandable and expected given that equation 25 is the full accounting of time that establishes the N^2 asymptotic trend. The model will therefore more accurately predict execution times in the asymptotic regime of large N.

The timing results shown here are limited by the memory requirements of the ITMM. The size of the matrix operators necessarily grows with increased size, number of angles, and anisotropic order. This constraint is significant as memory requirements of the matrix operators (specifically $\boldsymbol{J_\phi}$) quickly exceed system memory limits when any of these variables are augmented beyond the examples shown in this work (e.g., for a 12x12x12 sub-domain, S_{16} quadrature, and $L = 5$, memory requirements exceed 30 GB), requiring the use of smaller sub-domains and therefore an increased number of iterations. Also, previous parallel applications of the ITMM have found that 4x4x4 sub-domains perform optimally [5].

5. CONCLUSIONS

In this work, a new algorithm that provides for faster construction of the ITMM matrix operators has been described. This algorithm, the FMM, is based on the construction of a single, fundamental matrix, $\boldsymbol{F}$, representing the transport of a particle along every possible path throughout the sub-domain mesh. Each of the operators is constructed by multiplying an element of this fundamental matrix by two values dependent only upon the operator being constructed. The elements of $\boldsymbol{F}$ are face-based, meaning that they represent the transport of a particle along the aforementioned path from the emergent face of one cell to the incident face of another cell. By being a face-based quantity, the elements of $\boldsymbol{F}$ can then be used to relate one face to another face, a face to a cell, or a cell to a cell, dependent on the single cell coupling factors by which they are multiplied.

It can be seen from the results presented here that the FMM significantly outperforms the DMS in terms of matrix operator construction time for the ITMM. It should be noted that the time required for the solution algorithm, which was not detailed in this work, is significantly greater than the operator construction time. The FMM gains significance, however, when considering a calculation requiring multiple energy groups, time steps, or depletion steps, where repeated evaluations of the ITMM operators must be performed. In this situation, the time savings gained when using the FMM versus the DMS is multiplied by the number of time steps and/or the number of groups. As such, the FMM algorithm becomes more critical to the computational cost of the ITMM as the complexity of the target problem grows [7].

Although the FMM as shown here was developed for a structured Cartesian grid, an advantage of the FMM is its applicability to an unstructured mesh. The geometric simplicity of the FMM algorithm lies in that it does not require an orderly mesh sweep, but rather can be applied to any grouping of cells as long as their spatial relation to each other is known. Although the challenge of an unstructured tetrahedral mesh is to manage the bookkeeping of how the cells (and in the planned parallel environment, the sub-domains) fit together, this algorithm has demonstrated its potential to overcome this challenge of spatial domain decomposition, thus paving the way for an effective massively parallel method of solving the neutron transport equation on an unstructured tetrahedral mesh.

ACKNOWLEDGEMENTS

The Department of Energy Computational Science Graduate Fellowship (DOE CSGF) and the Department of Energy Nuclear Energy University Program (DOE NEUP) both supported this work, in part.

REFERENCES

[1] U. R. Hanebutte and E. E. Lewis, A Massively Parallel Discrete Ordinates Response Matrix Method for Neutron Transport, *Nuclear Science and Engineering*, **111**, 46 (1992).

[2] 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 1822, 1997 (1997).

[3] Y. Y. Azmy, Iterative Convergence Acceleration of Neutral Particle Transport Methods via Adjacent-Cell Preconditioners, *Journal of Computational Physics*, **152**, 359 (1999).

[4] M. Rosa, Y. Y. Azmy, and J. E. Morel, Properties of the S_N Equivalent Integral Transport Operator in Slab Geometry and the Iterative Acceleration of Neutral Particle Transport Methods, *Nuclear Science and Engineering*, **162** (3), 234 (2009).

[5] R. J. Zerr, Solution of the Within Group Multidimensional Discrete Ordinates Transport Equations on Massively Parallel Architectures, PhD Thesis, The Pennsylvania State University (2011).

[6] E. E. Lewis and W. F. Miller, Jr., *Computational Methods of Neutron Transport*, American Nuclear Society, La Grange Park, IL, USA (1993).

[7] R. J. Zerr, Personal Communication, January 10, 2013 (2013).

3.2 Fourier Analysis of the Integral Transport Matrix Method for the Diamond Difference Scheme

We performed Fourier analysis of the Integral Transport Matrix Method (ITMM) for the Diamond Difference (DD) scheme in various geometries. The ITMM is a spatial domain decomposition method for solving the discretized particle transport equation. We consider one-group transport problems with isotropic scattering and source. In the presented studies, it is assumed that there is one cell per subdomain. For each geometry, the equations for iterative errors of the ITMM are derived in a general form. A detailed analysis is carried out for S$_2$ quadrature set. The analysis of the ITMM in 1D slab geometry showed that the spectral radius $\rho_{1D} \leq 1$ and it depends on value of total cross section σ_t, the scattering ration c, and mesh size. The results of Fourier analysis in 2D and 3D Cartesian geometries demonstrated that the spectral radius $\rho_{2D} = \rho_{3D} = 1$ without regard to values of σ_t, c and mesh cell sizes. The iterative properties of the ITMM in 2D and 3D are similar. We study in details features of eigenvalues and associated eigenvectors in 2D geometry.

3.2.1 1D Slab Geometry

3.2.1.1 Formulation of the Computational Method

In this section we consider transport problems in 1D slab geometry. The DD method is given by

$$\mu_m(\psi_{m,i+1/2} - \psi_{m,i-1/2}) + \sigma_t \Delta x \psi_{m,i} = \frac{1}{2}\big(\sigma_s \Delta x \phi_i + q_i\big)\,, \tag{3.2.1}$$

$$\psi_{m,i} = \frac{1}{2}(\psi_{m,i+1/2} + \psi_{m,i-1/2})\,, \tag{3.2.2}$$

$$\phi_i = \sum_{m=1}^{M} \psi_{m,i} w_m\,. \tag{3.2.3}$$

We use standard notations. Equations (3.2.1)-(3.2.3) can be cast in the following general form:

$$\nu_m(\psi^+_{m,i} - \psi^-_{m,i}) + \psi_{m,i} = \frac{1}{2}(c\phi_i + \frac{q_i}{\sigma_t})\,, \tag{3.2.4}$$

$$\psi_{m,i} = \frac{1}{2}(\psi^+_{m,i} + \psi^-_{m,i})\,, \tag{3.2.5}$$

$$\phi_i = \sum_{m=1}^{M} \psi_{m,i} w_m\,, \tag{3.2.6}$$

$$\nu_m = \frac{|\mu_m|}{\sigma_t \Delta x}\,, \quad c = \frac{\sigma_s}{\sigma_t}\,, \tag{3.2.7}$$

$$\psi^-_{m,i} = \psi_{m,i\mp 1/2} \quad \text{for} \quad \mu_m \gtrless 0\,, \tag{3.2.8}$$

$$\psi^+_{m,i} = \psi_{m,i\pm 1/2} \quad \text{for} \quad \mu_m \gtrless 0\,. \tag{3.2.9}$$

The iteration scheme of ITMM can be formulated as follows:

$$\nu_m\big(\psi^{+(s)}_{m,i} - \psi^{-(s-1)}_{m,i}\big) + \psi^{(s)}_{m,i} = \frac{1}{2}\big(c\phi^{(s)}_i + \frac{q_i}{\sigma_t}\big)\,, \tag{3.2.10}$$

$$\psi^{(s)}_{m,i} = \frac{1}{2}\big(\psi^{+(s)}_{m,i} + \psi^{-(s-1)}_{m,i}\big)\,, \tag{3.2.11}$$

$$\phi_i^{(s)} = \sum_{m=1}^{M} \psi_m^{(s)} w_m \,. \tag{3.2.12}$$

The equations for the error in the s-th iterate

$$\delta\psi^{(s)} = \psi - \psi^{(s)} \,, \quad \delta\phi^{(s)} = \phi - \phi^{(s)} \tag{3.2.13}$$

are given by

$$\nu_m \big(\delta\psi_{m,i}^{+(s)} - \delta\psi_{m,i}^{-(s-1)}\big) + \delta\psi_{m,i}^{(s)} = \frac{1}{2} c\delta\phi_i^{(s)} \,, \tag{3.2.14a}$$

$$\delta\psi_{m,i}^{(s)} = \frac{1}{2}\big(\delta\psi_{m,i}^{+(s)} + \delta\psi_{m,i}^{-(s-1)}\big) \,, \tag{3.2.14b}$$

$$\delta\phi_i^{(s)} = \sum_{m=1}^{M} \delta\psi_m^{(s)} w_m \,. \tag{3.2.14c}$$

3.2.1.2 Fourier Analysis of Equations for Cell-Edge Angular Fluxes in S$_2$ Case

We now consider S$_2$ case:

$$M = 2 \,, \quad w_m = 1 \,, \quad |\mu_m| = \frac{1}{\sqrt{3}} \,. \tag{3.2.15}$$

The equations for errors in the cell-edge angular fluxes are the following:

$$(2\nu + 1 - 0.5c)\delta\psi_{1,i-1/2}^{(s)} - 0.5c\delta\psi_{2,i+1/2}^{(s)} = (2\nu - 1 + 0.5c)\delta\psi_{1,i+1/2}^{(s-1)} + 0.5c\delta\psi_{2,i-1/2}^{(s-1)} \,, \tag{3.2.16a}$$

$$(2\nu + 1 - 0.5c)\delta\psi_{2,i+1/2}^{(s)} - 0.5c\delta\psi_{1,i-1/2}^{(s)} = (2\nu - 1 + 0.5c)\delta\psi_{2,i-1/2}^{(s-1)} + 0.5c\delta\psi_{1,i+1/2}^{(s-1)} \,, \tag{3.2.16b}$$

where

$$\nu = \frac{1}{\sqrt{3}\sigma_t \Delta x} \,. \tag{3.2.17}$$

The equations (3.2.16) can be written as

$$p^+ \delta\psi_{1,i-1/2}^{(s)} - 0.5c\delta\psi_{2,i+1/2}^{(s)} = p^- \delta\psi_{1,i+1/2}^{(s-1)} + 0.5c\delta\psi_{2,i-1/2}^{(s-1)} \,, \tag{3.2.18a}$$

$$p^+ \delta\psi_{2,i+1/2}^{(s)} - 0.5c\delta\psi_{1,i-1/2}^{(s)} = p^- \delta\psi_{2,i-1/2}^{(s-1)} + 0.5c\delta\psi_{1,i+1/2}^{(s-1)} \,, \tag{3.2.18b}$$

where

$$p^{\pm} = 2\nu \pm 1 \mp 0.5c \,. \tag{3.2.19}$$

The equations (3.2.18) can be presented in a matrix form as

$$\mathbf{A}_{1D} \delta\vec{\psi}^{+(s)} = \mathbf{B}_{1D} \delta\vec{\psi}^{-(s-1)} \,, \tag{3.2.20}$$

$$\delta\vec{\psi}^{+} = (\delta\psi_{1,i-1/2}, \delta\psi_{2,i+1/2})^T \,, \quad \delta\vec{\psi}^{-} = (\delta\psi_{1,i+1/2}, \delta\psi_{2,i-1/2})^T \,, \tag{3.2.21}$$

where

$$\mathbf{A}_{1D} = \begin{pmatrix} p^+ & -0.5c \\ -0.5c & p^+ \end{pmatrix} \,, \quad \mathbf{B}_{1D} = \begin{pmatrix} p^- & 0.5c \\ 0.5c & p^- \end{pmatrix} \,. \tag{3.2.22}$$

Hence, the iteration process is defined by

$$\delta\vec{\psi}^{+(s)} = \mathbf{T}_{1D} \delta\vec{\psi}^{-(s-1)} \,, \tag{3.2.23}$$

where

$$\mathbf{T}_{1D} = \mathbf{A}_{1D}^{-1}\mathbf{B}_{1D}\,. \tag{3.2.24}$$

To perform the Fourier analysis we consider a single Fourier error mode with arbitrary λ and introduce the following ansatz:

$$\delta\psi_{m,i+1/2}^{(s)} = \alpha_m \omega^s(\lambda) e^{\mathbf{i}\lambda\sigma_t x_{i+1/2}}\,, \quad \mathbf{i} = \sqrt{-1}\,. \tag{3.2.25}$$

We apply (3.2.25) to Eqs. (3.2.18) to get

$$(p^+\omega - p^- e^{\mathbf{i}\tilde{\lambda}})\alpha_1 - 0.5c(1 + \omega e^{\mathbf{i}\tilde{\lambda}})\alpha_2 = 0\,, \tag{3.2.26}$$

$$-0.5c(\omega + e^{\mathbf{i}\tilde{\lambda}})\alpha_1 + (p^+\omega e^{\mathbf{i}\tilde{\lambda}} - p^-)\alpha_2 = 0\,, \tag{3.2.27}$$

where

$$\tilde{\lambda} = \lambda\sigma_t\Delta x\,. \tag{3.2.28}$$

The resulting equation for the eigenvalue $\omega(\lambda)$

$$\Bigg((p^+)^2 - \frac{1}{4}c^2\Bigg)\omega^2 - 2\cos\tilde{\lambda}\Bigg(p^+p^- + \frac{1}{4}c^2\Bigg)\omega + (p^-)^2 - \frac{1}{4}c^2 = 0\,. \tag{3.2.29}$$

There are two eigenvalues for each value of the wave number λ.

We note that Eq. (3.2.29) give rise to

$$\omega^2 - \cos\tilde{\lambda}\,\omega + 1 = 0 \quad \text{as} \quad \sigma_t\Delta x \to 0\,, \tag{3.2.30}$$

and

$$\omega^2 + \cos\tilde{\lambda}\,\omega + 1 = 0 \quad \text{as} \quad \sigma_t\Delta x \to \infty\,. \tag{3.2.31}$$

Thus the eigenvalues do not depend of the scattering ratio c and

$$\omega = e^{\pm\mathbf{i}\tilde{\lambda}} \quad \text{as} \quad \sigma_t\Delta x \to 0\,, \tag{3.2.32}$$

$$\omega = -e^{\pm\mathbf{i}\tilde{\lambda}} \quad \text{as} \quad \sigma_t\Delta x \to \infty\,. \tag{3.2.33}$$

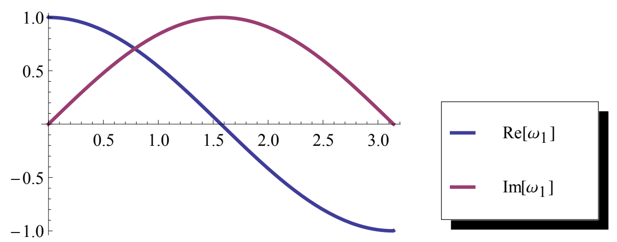
As a result, we have

$$|\omega| \to 1 \quad \forall\tilde{\lambda} \quad \text{as} \quad \sigma_t\Delta x \to \infty \quad \text{or} \quad \sigma_t\Delta x \to 0\,. \tag{3.2.34}$$

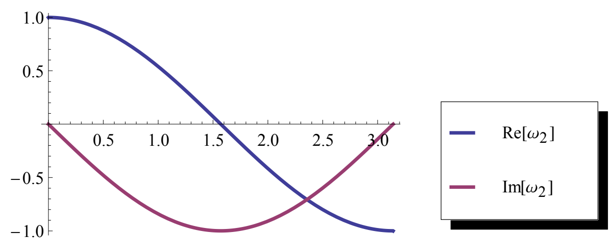
Thus,

$$\rho_{1D} \to 1 \quad \forall c \quad \text{as} \quad \sigma_t\Delta x \to \infty \quad \text{or} \quad \sigma_t\Delta x \to 0. \tag{3.2.35}$$

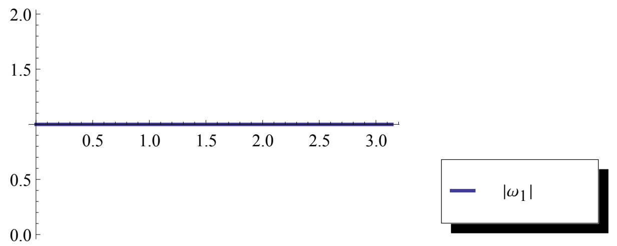
Figures 3.2.1-3.2.5 show the eigenvalues as functions of $\tilde{\lambda}$ (Eq. (3.2.28)) for $c = 0.9$ and selected values of $\sigma_t\Delta x$. These results demonstrate typical behaviour of ω. We note that ω is complex in general. The values of spectral radii $\rho = \sup_\lambda |\omega(\lambda)|$ for some range of c and $\sigma_t\Delta x$ are listed in Table 3.2.1. These results show that for a given c spectral radius has minimum around $\sigma_t\Delta x = 1$. It tends to 1 as $\sigma_t\Delta x$ tends to 0 and ∞.



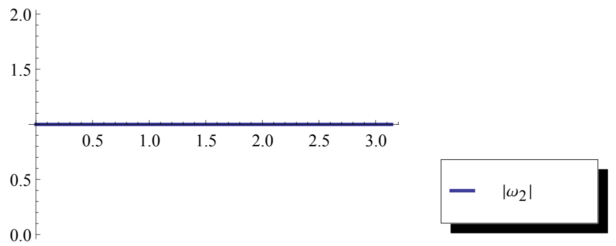
(a) ω_1



(b) ω_2

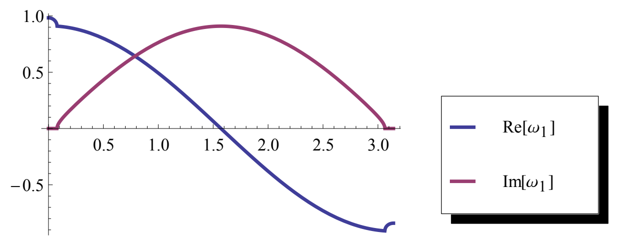


(c) $|\omega_1|$

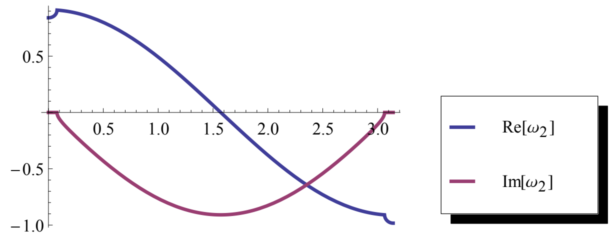


(d) $|\omega_2|$

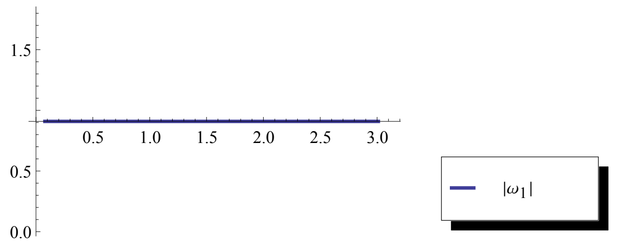
Figure 3.2.1: $\omega(\tilde{\lambda})$ for 1D, $c = 0.9$, $\sigma_t \Delta x$=10^{-3}



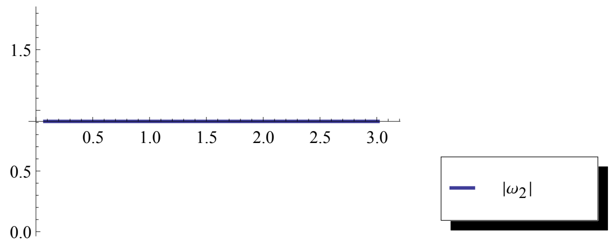
(a) ω_1



(b) ω_2

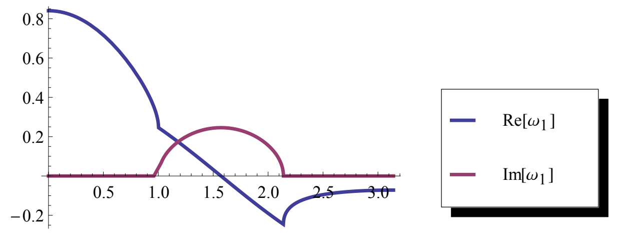


(c) $|\omega_1|$

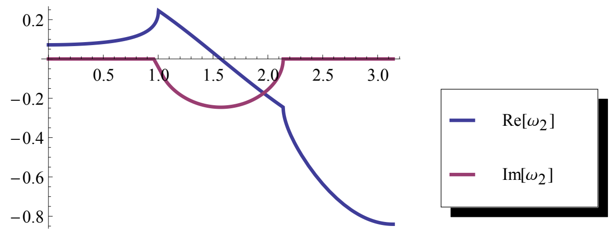


(d) $|\omega_2|$

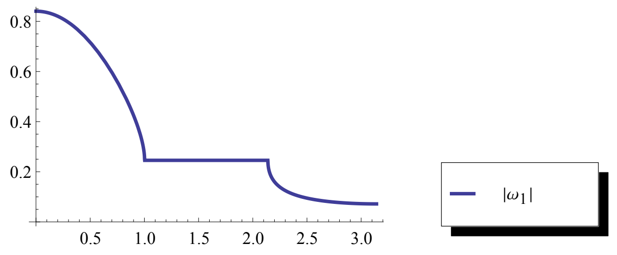
Figure 3.2.2: $\omega(\tilde{\lambda})$ for 1D, $c = 0.9$, $\sigma_t \Delta x$=0.1



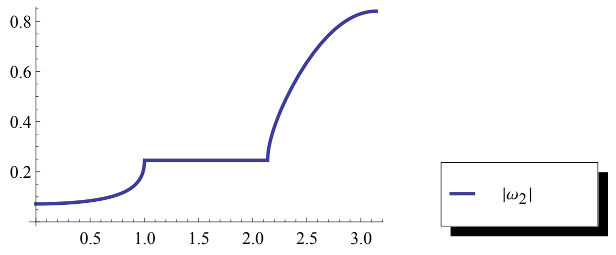
(a) ω_1



(b) ω_2

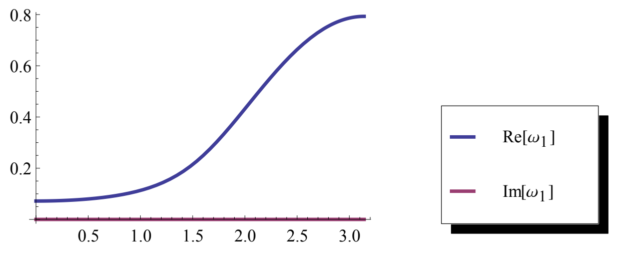


(c) $|\omega_1|$

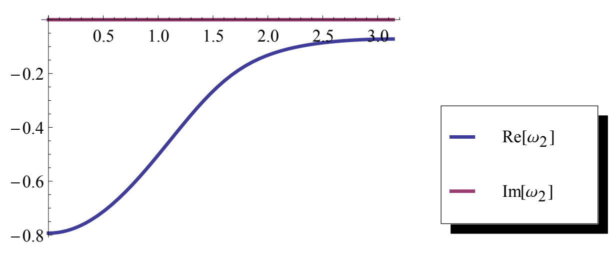


(d) $|\omega_2|$

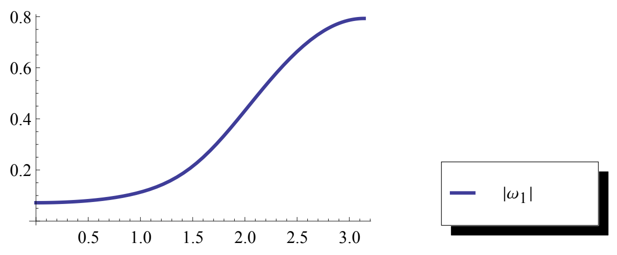
Figure 3.2.3: $\omega(\tilde{\lambda})$ for 1D, $c = 0.9$, $\sigma_t \Delta x$=1



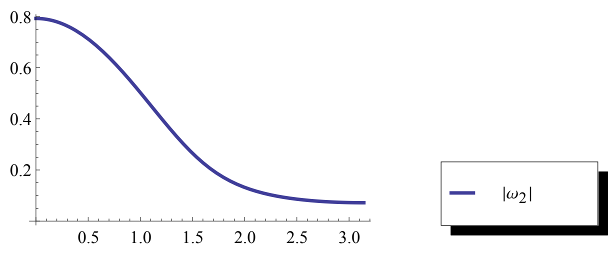
(a) ω_1



(b) ω_2

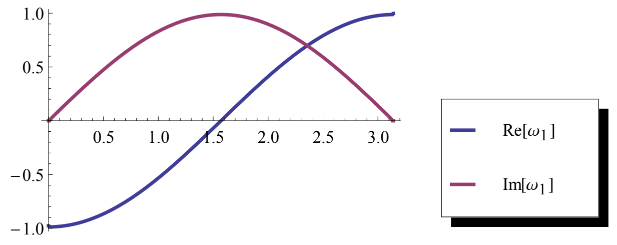


(c) $|\omega_1|$

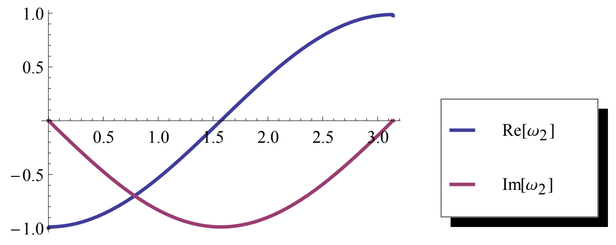


(d) $|\omega_2|$

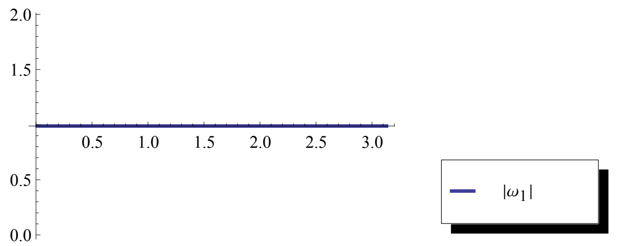
Figure 3.2.4: $\omega(\tilde{\lambda})$ for 1D, $c = 0.9$, $\sigma_t \Delta x$=10



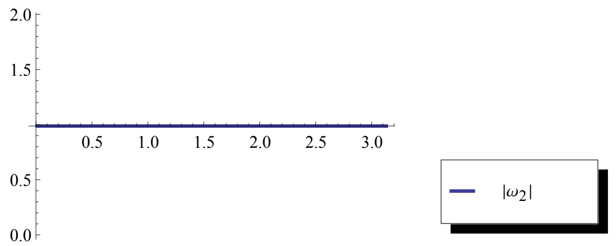
(a) ω_1



(b) ω_2



(c) $|\omega_1|$



(d) $|\omega_2|$

Figure 3.2.5: $\omega(\tilde{\lambda})$ for 1D, $c = 0.9$, $\sigma_t \Delta x$=10^3

Table 3.2.1: Theoretical estimation of spectral radii ρ_{1D}

	$\sigma_t \Delta x$													
c	0.001	0.01	0.1	0.25	0.5	0.75	1	2	2.5	3	5	10	100	1000
0.1	0.998	0.984	0.855	0.674	0.439	0.262	0.124	0.268	0.368	0.444	0.625	0.793	0.977	0.998
0.5	0.999	0.991	0.917	0.805	0.644	0.51	0.396	0.268	0.368	0.444	0.625	0.793	0.977	0.998
0.7	0.999	0.995	0.949	0.878	0.77	0.674	0.588	0.316	0.368	0.444	0.625	0.793	0.977	0.998
0.9	0.999	0.998	0.983	0.958	0.917	0.878	0.841	0.705	0.644	0.588	0.625	0.793	0.977	0.998
0.99	0.999	0.999	0.998	0.996	0.991	0.987	0.983	0.966	0.958	0.949	0.917	0.841	0.977	0.998
0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.998	0.997	0.996	0.995	0.991	0.983	0.977	0.998

3.2.1.3 Fourier Analysis of Equations for Complete Set of Unknowns in S$_2$ Case

We now perform analysis of the iteration process that involves explicitly complete set of unknowns, namely, the cell-edge and cell-average angular fluxes. It is defined by Eqs. (3.2.14). The equations for iterative errors in S$_2$ case are the following:

$$(2\nu + 1 - 0.5c)\delta\psi_{1,i}^{(s)} - 0.5c\delta\psi_{2,i}^{(s)} - 2\nu\delta\psi_{1,i+1/2}^{(s-1)} = 0\,, \tag{3.2.36a}$$

$$(2\nu + 1 - 0.5c)\delta\psi_{2,i}^{(s)} - 0.5c\delta\psi_{1,i}^{(s)} - 2\nu\delta\psi_{2,i-1/2}^{(s-1)} = 0\,, \tag{3.2.36b}$$

$$2\delta\psi_{1,i}^{(s)} - \delta\psi_{1,i-1/2}^{(s)} - \delta\psi_{1,i+1/2}^{(s-1)} = 0\,, \tag{3.2.36c}$$

$$2\delta\psi_{2,i}^{(s)} - \delta\psi_{2,i+1/2}^{(s)} - \delta\psi_{2,i-1/2}^{(s-1)} = 0\,. \tag{3.2.36d}$$

We introduce the following ansatz:

$$\delta\psi_{m,i+1/2}^{(s)} = \alpha_m \omega^s(\lambda) e^{\mathbf{i}\lambda\sigma_t x_{i+1/2}}\,, \tag{3.2.37}$$

$$\delta\psi_{m,i}^{(s)} = \gamma_m \omega^s(\lambda) e^{\mathbf{i}\lambda\sigma_t x_i} \,. \tag{3.2.38}$$

Using (3.2.37) and (3.2.38) in Eqs. (3.2.36) we get

$$\omega\big(p^+\gamma_1 - 0.5c\gamma_2\big) - 2\nu e^{\mathbf{i}z}\alpha_1 = 0\,, \tag{3.2.39a}$$

$$\omega\big(-0.5c\gamma_1 + p^+\gamma_2\big) - 2\nu e^{-\mathbf{i}z}\alpha_2 = 0\,, \tag{3.2.39b}$$

$$2\omega\gamma_1 - \big(\omega e^{-\mathbf{i}z} + e^{\mathbf{i}z}\big)\alpha_1 = 0\,, \tag{3.2.39c}$$

$$2\omega\gamma_2 - \big(e^{-\mathbf{i}z} + \omega e^{\mathbf{i}z}\big)\alpha_2 = 0\,, \tag{3.2.39d}$$

where

$$z = \frac{1}{2}\tilde{\lambda} = \frac{1}{2}\lambda\sigma_t\Delta x\,. \tag{3.2.40}$$

The matrix form of Eqs. (3.2.39) is given by

$$\begin{pmatrix} p^+\omega & -0.5c\omega & -2\nu e^{\mathbf{i}z} & 0 \\ -0.5c\omega & p^+\omega & 0 & -2\nu e^{-\mathbf{i}z} \\ 2\omega & 0 & -(\omega e^{-\mathbf{i}z} + e^{\mathbf{i}z}) & 0 \\ 0 & 2\omega & 0 & -(e^{-\mathbf{i}z} + \omega e^{\mathbf{i}z}) \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = 0\,, \tag{3.2.41}$$

It reduces to

$$\mathbf{C}_{1D}\vec{f} = 0\,, \tag{3.2.42}$$

where

$$\mathbf{C}_{1D} = \begin{pmatrix} 2\omega & 0 & -(\omega e^{-\mathbf{i}z} + e^{\mathbf{i}z}) & 0 \\ 0 & 2\omega & 0 & -(e^{-\mathbf{i}z} + \omega e^{\mathbf{i}z}) \\ 0 & 0 & 0.5(p^+\omega e^{-\mathbf{i}z} - p^- e^{\mathbf{i}z}) & -0.25c(e^{-\mathbf{i}z} + \omega e^{\mathbf{i}z}) \\ 0 & 0 & -0.25c(\omega e^{-\mathbf{i}z} + e^{\mathbf{i}z}) & 0.5(p^+\omega e^{\mathbf{i}z} - p^- e^{-\mathbf{i}z}) \end{pmatrix} \tag{3.2.43}$$

and

$$\vec{f} = \big(\gamma_1, \gamma_2, \alpha_1, \alpha_2\big)^T\,. \tag{3.2.44}$$

The determinant of $\mathbf{C}_{1D}$ has the following form:

$$\det \mathbf{C}_{1D} = 4\omega^2\left[\left(p^+\omega - p^- e^{\mathbf{i}2z}\right)\left(p^+\omega e^{\mathbf{i}2z} - p^-\right) - \frac{1}{4}c^2(\omega + e^{\mathbf{i}2z})(1 + \omega e^{\mathbf{i}2z})\right]. \tag{3.2.45}$$

Thus ω=0 is the eigenvalue of multiplicity 2 without regard to parameters of the transport problem and spatial interval width. The equation for the rest two eigenvalues is given by Eq. (3.2.29).

3.2.1.4 Summary

The main results of the analysis of ITMM in 1D slab geometry are the following:

1. The spectral radius of ITTM

   $$\rho_{1D} = \sup_{\lambda} |\omega(\lambda)| \leq 1\,,$$

   It depends on values of c on Δx.

2. The convergence of ITMM slows down with increase in optical thickness of mesh intervals as well as with its decrease. We showed that

   $$|\omega| \to 1 \quad \forall\lambda \quad \text{as} \quad \sigma_t\Delta x \to \infty \quad \text{or} \quad \sigma_t\Delta x \to 0 \tag{3.2.46}$$

   without regard to the value of c. Thus,

   $$\rho_{1D} \to 1 \quad \forall c \quad \text{as} \quad \sigma_t\Delta x \to \infty \quad \text{or} \quad \sigma_t\Delta x \to 0. \tag{3.2.47}$$

3.2.2 2D Cartesian Geometry

3.2.2.1 Formulation of the Computational Method

In this section we consider one-group transport problems in 2D Cartesian geometry with isotropic scattering and source. The DD method is defined by the following set of equations:

$$\nu_{x,m}(\psi^{x,+}_{m,i,j} - \psi^{x,-}_{m,i,j}) + \nu_{y,m}(\psi^{y,+}_{m,i,j} - \psi^{y,-}_{m,i,j}) + \psi_{m,i,j} = \frac{1}{4\pi}\left(c\phi_{i,j} + \frac{q}{\sigma_t}\right), \tag{3.2.48a}$$

$$\psi_{m,i,j} = \frac{1}{2}\left(\psi^{x,+}_{m,i,j} + \psi^{x,-}_{m,i,j}\right), \tag{3.2.48b}$$

$$\psi_{m,i,j} = \frac{1}{2}\left(\psi^{y,+}_{m,i,j} + \psi^{y,-}_{m,i,j}\right), \tag{3.2.48c}$$

$$\phi_{i,j} = \sum_{m=1}^{M} \psi_{m,i,j} w_m , \tag{3.2.48d}$$

$$\nu_{x,m} = \frac{|\Omega_{x,m}|}{\sigma_t \Delta x}, \quad \nu_{y,m} = \frac{|\Omega_{y,m}|}{\sigma_t \Delta y}, \quad c = \frac{\sigma_s}{\sigma_t}, \tag{3.2.49}$$

$$\psi^{x,\pm}_{m,i,j} = \psi_{m,i\pm1/2,j}, \quad \psi^{y,\pm}_{m,i,j} = \psi_{m,i,j\pm1/2}, \quad \text{for} \quad \Omega_{x,m} > 0, \ \Omega_{y,m} > 0, \tag{3.2.50a}$$

$$\psi^{x,\pm}_{m,i,j} = \psi_{m,i\mp1/2,j}, \quad \psi^{y,\pm}_{m,i,j} = \psi_{m,i,j\pm1/2}, \quad \text{for} \quad \Omega_{x,m} < 0, \ \Omega_{y,m} > 0, \tag{3.2.50b}$$

$$\psi^{x,\pm}_{m,i,j} = \psi_{m,i\mp1/2,j}, \quad \psi^{y,\pm}_{m,i,j} = \psi_{m,i,j\mp1/2}, \quad \text{for} \quad \Omega_{x,m} < 0, \ \Omega_{y,m} < 0, \tag{3.2.50c}$$

$$\psi^{x,\pm}_{m,i,j} = \psi_{m,i\pm1/2,j}, \quad \psi^{y,\pm}_{m,i,j} = \psi_{m,i,j\mp1/2}, \quad \text{for} \quad \Omega_{x,m} > 0, \ \Omega_{y,m} < 0. \tag{3.2.50d}$$

The standard notations are used. The iteration scheme is defined as follows

$$\nu_{x,m}(\psi^{x,+(s)}_{m,i,j} - \psi^{x,-(s-1)}_{m,i,j}) + \nu_{y,m}(\psi^{y,+(s)}_{m,i,j} - \psi^{y,-(s-1)}_{m,i,j}) + \psi^{(s)}_{m,i,j} = \frac{1}{4\pi}\left(c\phi^{(s)}_{i,j} + \frac{q_{i,j}}{\sigma_t}\right), \tag{3.2.51a}$$

$$\psi^{(s)}_{m,i,j} = \frac{1}{2}\left(\psi^{x,+(s)}_{m,i,j} + \psi^{x,-(s-1)}_{m,i,j}\right), \tag{3.2.51b}$$

$$\psi^{(s)}_{m,i,j} = \frac{1}{2}\left(\psi^{y,+(s)}_{m,i,j} + \psi^{y,-(s-1)}_{m,i,j}\right), \tag{3.2.51c}$$

$$\phi^{(s)}_{i,j} = \sum_{m=1}^{M} \psi^{(s)}_{m,i,j} w_m , \tag{3.2.51d}$$

and hence the equations for the iterative errors are given by

$$\nu_{x,m}\left(\delta\psi^{x,+(s)}_{m,i,j} - \delta\psi^{x,-(s-1)}_{m,i,j}\right) + \nu_{y,m}\left(\delta\psi^{y,+(s)}_{m,i,j} - \delta\psi^{y,-(s-1)}_{m,i,j}\right) + \psi^{(s)}_{m,i,j} = \frac{c}{4\pi}\delta\phi^{(s)}_{i,j}, \tag{3.2.52a}$$

$$\delta\psi^{(s)}_{m,i,j} = \frac{1}{2}\left(\delta\psi^{x,+(s)}_{m,i,j} + \delta\psi^{x,-(s-1)}_{m,i,j}\right), \tag{3.2.52b}$$

$$\delta\psi^{(s)}_{m,i,j} = \frac{1}{2}\left(\delta\psi^{y,+(s)}_{m,i,j} + \delta\psi^{y,-(s-1)}_{m,i,j}\right), \tag{3.2.52c}$$

$$\delta\phi_{i,j}^{(s)} = \sum_{m=1}^{M} \delta\psi_{m,i,j}^{(s)} w_m \,. \tag{3.2.52d}$$

We cast Eqs. (3.2.52) as follows

$$\nu_{x,m}(\delta\psi_{m,i,j}^{x,+(s)} - \delta\psi_{m,i,j}^{x,-(s-1)}) + \nu_{y,m}(\delta\psi_{m,i,j}^{y,+(s)} - \delta\psi_{m,i,j}^{y,-(s-1)}) + \psi_{m,i,j}^{(s)} = \frac{c}{4\pi}\delta\phi_{i,j^{(s)}} \,, \tag{3.2.53a}$$

$$\delta\psi_{m,i,j}^{(s)} = \frac{1}{4}\left(\delta\psi_{m,i,j}^{x,+(s)} + \delta\psi_{m,i,j}^{x,-(s-1)} + \delta\psi_{m,i,j}^{y,+(s)} + \delta\psi_{m,i,j}^{y,-(s-1)}\right), \tag{3.2.53b}$$

$$\delta\psi_{m,i,j}^{x,+(s)} + \delta\psi_{m,i,j}^{x,-(s-1)} = \delta\psi_{m,i,j}^{y,+(s)} + \delta\psi_{m,i,j}^{y,-(s-1)} \,, \tag{3.2.53c}$$

$$\delta\phi_{i,j}^{(s)} = \sum_{m=1}^{M} \delta\psi_{m,i,j}^{(s)} w_m \,. \tag{3.2.53d}$$

Using these equations we derive the equations for the face-average angular fluxes:

$$\begin{aligned} p_{x,m}^{+}\delta\psi_{m,i,j}^{x,+(s)} + p_{x,m}^{-}\delta\psi_{m,i,j}^{x,-(s-1)} + p_{y,m}^{+}\delta\psi_{m,i,j}^{y,+(s)} + p_{y,m}^{-}\delta\psi_{m,i,j}^{y,-(s-1)} &= \\ \frac{c}{4\pi}\sum_{m'\neq m}\left(\delta\psi_{m,i,j}^{x,+(s)} + \delta\psi_{m,i,j}^{x,-(s-1)} + \delta\psi_{m,i,j}^{y,+(s)} + \delta\psi_{m,i,j}^{y,-(s-1)}\right)w_{m'} \,, \end{aligned} \tag{3.2.54a}$$

$$\delta\psi_{m,i,j}^{x,+(s)} + \delta\psi_{m,i,j}^{x,-(s-1)} = \delta\psi_{m,i,j}^{y,+(s)} + \delta\psi_{m,i,j}^{y,-(s-1)} \,, \tag{3.2.54b}$$

where

$$p_{\alpha,m}^{\pm} = 1 \pm 4\nu_{\alpha,m} - \frac{cw_m}{4\pi} \,, \quad \alpha = x, y \,. \tag{3.2.55}$$

3.2.2.2 Alternative Form of Equations

There is an alternative way for deriving equation of ITMM. The auxiliary conditions (3.2.48b) and (3.2.48c) can be written as

$$\psi_{m,i,j}^{\alpha,+} = 2\psi_{m,i,j} - \psi_{m,i,j}^{\alpha,-} \,, \quad \alpha = x, y \tag{3.2.56}$$

and hence Eq. (3.2.48a) leads to

$$2\nu_{x,m}(\psi_{m,i,j} - \psi_{m,i,j}^{x,-}) + 2\nu_{y,m}(\psi_{m,i,j} - \psi_{m,i,j}^{y,-}) + \psi_{m,i,j} = \frac{c}{4\pi}\phi_{i,j} \,. \tag{3.2.57}$$

Here we set the source term to zero to simplify derivation of equations for iterative errors. From Eq. (3.2.57) we get

$$\psi_{m,i,j} = \frac{c\phi_{i,j}}{4\pi(1 + 2(\nu_{x,m} + \nu_{y,m}))} + \frac{2(\nu_{x,m}\psi_{m,i,j}^{x,-} + \nu_{y,m}\psi_{m,i,j}^{y,-})}{1 + 2(\nu_{x,m} + \nu_{y,m})} \,. \tag{3.2.58}$$

Thus, the equation for the cell-average scalar flux has the following form:

$$\phi_{i,j} = 2\left[1 - \frac{c}{4\pi}\sum_{m}\frac{w_m}{1 + 2(\nu_{x,m} + \nu_{y,m})}\right]^{-1}\sum_{m}\frac{w_m(\nu_{x,m}\psi_{m,i,j}^{x,-} + \nu_{y,m}\psi_{m,i,j}^{y,-})}{1 + 2(\nu_{x,m} + \nu_{y,m})} \tag{3.2.59}$$

and hence

$$\phi_{i,j} = \frac{4\pi}{c}\sum_{m}(\gamma_m^x\psi_{m,i,j}^{x,-} + \gamma_m^y\psi_{m,i,j}^{y,-}) \,, \tag{3.2.60}$$

where

$$\gamma_m^\alpha = \frac{\eta \nu_{\alpha,m} w_m}{1 + 2(\nu_{x,m} + \nu_{y,m})}\,, \tag{3.2.61}$$

$$\eta = 2\left[\frac{4\pi}{c} - \sum_m \frac{w_m}{1 + 2(\nu_{x,m} + \nu_{y,m})}\right]^{-1}. \tag{3.2.62}$$

We note that the auxiliary conditions (3.2.48b) and (3.2.48c) also give rise to

$$\psi_{m,i,j} = \frac{1}{4}\left(\psi_{m,i,j}^{x,+} + \psi_{m,i,j}^{x,-} + \psi_{m,i,j}^{y,+} + \psi_{m,i,j}^{y,-}\right). \tag{3.2.63}$$

Using Eq. (3.2.63) in Eq. (3.2.48a) we obtain

$$\left(\frac{1}{4} + \nu_{x,m}\right)\psi_{m,i,j}^{x,+} + \left(\frac{1}{4} - \nu_{x,m}\right)\psi_{m,i,j}^{x,-} + \left(\frac{1}{4} + \nu_{y,m}\right)\psi_{m,i,j}^{y,+} + \left(\frac{1}{4} - \nu_{y,m}\right)\psi_{m,i,j}^{y,-} = \frac{c}{4\pi}\phi_{i,j}\,. \tag{3.2.64}$$

The equations (3.2.60) and (3.2.64) lead to the following equation in terms of incoming and outgoing face-average angular fluxes:

$$\begin{aligned}\left(\frac{1}{4} + \nu_{x,m}\right)\psi_{m,i,j}^{x,+} + \left(\frac{1}{4} - \nu_{x,m} - \gamma_m^x\right)\psi_{m,i,j}^{x,-}& \\ + \left(\frac{1}{4} + \nu_{y,m}\right)\psi_{m,i,j}^{y,+} + \left(\frac{1}{4} - \nu_{y,m} - \gamma_m^y\right)\psi_{m,i,j}^{y,-} &= \\ \sum_{m' \neq m}\left(\gamma_{m'}^x \psi_{m',i,j}^{x,-} + \gamma_{m'}^y \psi_{m',i,j}^{y,-}\right)&.\end{aligned} \tag{3.2.65}$$

From Eq. (3.2.65) we get the following equations for iterative errors

$$\begin{aligned}\Big(1 + 4\nu_{x,m}\Big)\delta\psi_{m,i,j}^{x,+(s)} + \Big(1 + 4\nu_{y,m}\Big)\delta\psi_{m,i,j}^{y,+(s)} &= \\ \Big(4(\nu_{x,m} + \gamma_m^x) - 1\Big)\delta\psi_{m,i,j}^{x,-(s-1)} + \Big(4(\nu_{y,m} + \gamma_m^y) - 1\Big)\delta\psi_{m,i,j}^{y,-(s-1)}& \\ + 4\sum_{m' \neq m}\Big(\gamma_{m'}^x \psi_{m',i,j}^{x,-(s-1)} + \gamma_{m'}^y \psi_{m',i,j}^{y,-(s-1)}\Big)&.\end{aligned} \tag{3.2.66}$$

The auxiliary condition is defined as

$$\delta\psi_{m,i,j}^{x,+(s)} + \delta\psi_{m,i,j}^{x,-(s-1)} = \delta\psi_{m,i,j}^{y,+(s)} + \delta\psi_{m,i,j}^{y,-(s-1)}\,. \tag{3.2.67}$$

3.2.2.3 Fourier Analysis of S$_2$ Case

In case of S$_2$ quadrature set we have

$$M = 4\,, \quad w_m = \pi\,, \quad |\Omega_{m,x}| = |\Omega_{m,y}| = \frac{1}{\sqrt{3}}\,. \tag{3.2.68}$$

The equations (3.2.54) lead to

$$\begin{aligned}p_x^+\delta\psi^{x,+(s)} + p_x^-\delta\psi_{m,i,j}^{x,-(s-1)} + p_y^+\delta\psi_{m,i,j}^{y,+(s)} + p_y^-\delta\psi_{m,i,j}^{y,-(s-1)} &= \\ \tilde{c}\sum_{m' \neq m}\left(\delta\psi_{m',i,j}^{x,+(s)} + \delta\psi_{m',i,j}^{x,-(s-1)} + \delta\psi_{m',i,j}^{y,+(s)} + \delta\psi_{m',i,j}^{y,-(s-1)}\right)&,\end{aligned} \tag{3.2.69a}$$

$$\delta\psi^{x,+(s)}_{m,i,j} + \delta\psi^{x,-(s-1)}_{m,i,j} = \delta\psi^{y,+(s)}_{m,i,j} + \delta\psi^{y,-(s-1)}_{m,i,j}\,, \tag{3.2.69b}$$

where

$$\tilde{c} = \frac{1}{4}c \tag{3.2.70}$$

$$p^{\pm}_{\alpha} = 1 \pm 4\nu_{\alpha} - \tilde{c}\,, \quad \alpha = x, y\,, \tag{3.2.71}$$

$$\nu_{\alpha} = \frac{1}{\sqrt{3}\sigma_t \Delta\alpha}\,. \tag{3.2.72}$$

The matrix form of equations (3.2.69) is the following:

$$\mathbf{A}_{2D}\delta\vec{\psi}^{+(s)} = \mathbf{B}_{2D}\delta\vec{\psi}^{-(s-1)}\,, \tag{3.2.73a}$$

$$\delta\vec{\psi}^{+} = (\delta\psi_{1,i+1/2,j}, \delta\psi_{2,i-1/2,j}, \delta\psi_{3,i-1/2,j}, \delta\psi_{4,i+1/2,j}, \\ \delta\psi_{1,i,j+1/2}, \delta\psi_{2,i,j+1/2}, \delta\psi_{3,i,j-1/2}, \delta\psi_{4,i,j-1/2})^{T}\,, \tag{3.2.73b}$$

$$\delta\vec{\psi}^{-} = (\delta\psi_{1,i-1/2,j}, \delta\psi_{2,i+1/2,j}, \delta\psi_{3,i+1/2,j}, \delta\psi_{4,i-1/2,j}, \\ \delta\psi_{1,i,j-1/2}, \delta\psi_{2,i,j-1/2}, \delta\psi_{3,i,j+1/2}, \delta\psi_{4,i,j+1/2})^{T}\,, \tag{3.2.73c}$$

$$\mathbf{A}_{2D} = \begin{pmatrix} p^{+}_{x} & -\tilde{c} & -\tilde{c} & -\tilde{c} & p^{+}_{y} & -\tilde{c} & -\tilde{c} & -\tilde{c} \\ -\tilde{c} & p^{+}_{x} & -\tilde{c} & -\tilde{c} & -\tilde{c} & p^{+}_{y} & -\tilde{c} & -\tilde{c} \\ -\tilde{c} & -\tilde{c} & p^{+}_{x} & -\tilde{c} & -\tilde{c} & -\tilde{c} & p^{+}_{y} & -\tilde{c} \\ -\tilde{c} & -\tilde{c} & -\tilde{c} & p^{+}_{x} & -\tilde{c} & -\tilde{c} & -\tilde{c} & p^{+}_{y} \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix}\,, \tag{3.2.73d}$$

$$\mathbf{B}_{2D} = \begin{pmatrix} -p^{-}_{x} & \tilde{c} & \tilde{c} & \tilde{c} & -p^{-}_{y} & \tilde{c} & \tilde{c} & \tilde{c} \\ \tilde{c} & -p^{-}_{x} & \tilde{c} & \tilde{c} & \tilde{c} & -p^{-}_{y} & \tilde{c} & \tilde{c} \\ \tilde{c} & \tilde{c} & -p^{-}_{x} & \tilde{c} & \tilde{c} & \tilde{c} & -p^{-}_{y} & \tilde{c} \\ \tilde{c} & \tilde{c} & \tilde{c} & -p^{-}_{x} & \tilde{c} & \tilde{c} & \tilde{c} & -p^{-}_{y} \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{pmatrix}\,. \tag{3.2.73e}$$

Thus

$$\delta\vec{\psi}^{+(s)} = \mathbf{T}_{2D}\delta\vec{\psi}^{-(s-1)}\,, \tag{3.2.74}$$

where

$$\mathbf{T}_{2D} = \mathbf{A}^{-1}_{2D}\mathbf{B}_{2D}\,. \tag{3.2.75}$$

To carry out the Fourier analysis we introduce the following ansatz:

$$\delta\psi^{(s)}_{m,i+1/2,j} = \omega^{s}(\lambda)a_{x,m}e^{\mathbf{i}\sigma_t(\lambda_x x_{i+1/2} + \lambda_y y_j)}\,, \tag{3.2.76a}$$

$$\delta\psi^{(s)}_{m,i,j+1/2} = \omega^{s}(\lambda)a_{y,m}e^{\mathbf{i}\sigma_t(\lambda_x x_i + \lambda_y y_{j+1/2})} \tag{3.2.76b}$$

considering a single Fourier error mode with arbitrary λ_x and λ_y. We apply (3.2.76) in Eqs. (3.2.69) taking into account (3.2.50). As a result, we obtain the equation for ω given by

$$\Big(\omega\tilde{\mathbf{A}}_{2D} - \tilde{\mathbf{B}}_{2D}\Big)\vec{a} = 0\,, \tag{3.2.77}$$

where $\vec{a}$ is the vector of coefficients of the Fourier expansion

$$\vec{a} = (a_{x,1}, a_{x,2}, a_{x,3}, a_{x,4}, a_{y,1}, a_{y,2}, a_{y,3}, a_{y,4})^T\,, \tag{3.2.78}$$

the matrices are defined as

$$\tilde{\mathbf{A}}_{2D} = \begin{pmatrix} p_x^+e^{\mathbf{i}\tau_x} & -\tilde{c}e^{-\mathbf{i}\tau_x} & -\tilde{c}e^{-\mathbf{i}\tau_x} & -\tilde{c}e^{\mathbf{i}\tau_x} & p_y^+e^{\mathbf{i}\tau_y} & -\tilde{c}e^{\mathbf{i}\tau_y} & -\tilde{c}e^{-\mathbf{i}\tau_y} & -\tilde{c}e^{-\mathbf{i}\tau_y} \\ -\tilde{c}e^{\mathbf{i}\tau_x} & p_x^+e^{-\mathbf{i}\tau_x} & -\tilde{c}e^{-\mathbf{i}\tau_x} & -\tilde{c}e^{\mathbf{i}\tau_x} & -\tilde{c}e^{\mathbf{i}\tau_y} & p_y^+e^{\mathbf{i}\tau_y} & -\tilde{c}e^{-\mathbf{i}\tau_y} & -\tilde{c}e^{-\mathbf{i}\tau_y} \\ -\tilde{c}e^{\mathbf{i}\tau_x} & -\tilde{c}e^{-\mathbf{i}\tau_x} & p_x^+e^{-\mathbf{i}\tau_x} & -\tilde{c}e^{\mathbf{i}\tau_x} & -\tilde{c}e^{\mathbf{i}\tau_y} & -\tilde{c}e^{\mathbf{i}\tau_y} & p_y^+e^{-\mathbf{i}\tau_y} & -\tilde{c}e^{-\mathbf{i}\tau_y} \\ -\tilde{c}e^{\mathbf{i}\tau_x} & -\tilde{c}e^{-\mathbf{i}\tau_x} & -\tilde{c}e^{-\mathbf{i}\tau_x} & p_x^+e^{\mathbf{i}\tau_x} & -\tilde{c}e^{\mathbf{i}\tau_y} & -\tilde{c}e^{\mathbf{i}\tau_y} & -\tilde{c}e^{-\mathbf{i}\tau_y} & p_y^+e^{-\mathbf{i}\tau_y} \\ e^{\mathbf{i}\tau_x} & 0 & 0 & 0 & -e^{\mathbf{i}\tau_y} & 0 & 0 & 0 \\ 0 & e^{-\mathbf{i}\tau_x} & 0 & 0 & 0 & -e^{\mathbf{i}\tau_y} & 0 & 0 \\ 0 & 0 & e^{-\mathbf{i}\tau_x} & 0 & 0 & 0 & -e^{-\mathbf{i}\tau_y} & 0 \\ 0 & 0 & 0 & e^{\mathbf{i}\tau_x} & 0 & 0 & 0 & -e^{-\mathbf{i}\tau_y} \end{pmatrix}, \tag{3.2.79}$$

$$\tilde{\mathbf{B}}_{2D} = \begin{pmatrix} -p_x^-e^{-\mathbf{i}\tau_x} & \tilde{c}e^{\mathbf{i}\tau_x} & \tilde{c}e^{\mathbf{i}\tau_x} & \tilde{c}e^{-\mathbf{i}\tau_x} & -p_y^-e^{-\mathbf{i}\tau_y} & \tilde{c}e^{-\mathbf{i}\tau_y} & \tilde{c}e^{\mathbf{i}\tau_y} & \tilde{c}e^{\mathbf{i}\tau_y} \\ \tilde{c}e^{-\mathbf{i}\tau_x} & -p_x^-e^{\mathbf{i}\tau_x} & \tilde{c}e^{\mathbf{i}\tau_x} & \tilde{c}e^{-\mathbf{i}\tau_x} & \tilde{c}e^{-\mathbf{i}\tau_y} & -p_y^-e^{-\mathbf{i}\tau_y} & \tilde{c}e^{\mathbf{i}\tau_y} & \tilde{c}e^{\mathbf{i}\tau_y} \\ \tilde{c}e^{-\mathbf{i}\tau_x} & \tilde{c}e^{\mathbf{i}\tau_x} & -p_x^-e^{\mathbf{i}\tau_x} & \tilde{c}e^{-\mathbf{i}\tau_x} & \tilde{c}e^{-\mathbf{i}\tau_y} & \tilde{c}e^{-\mathbf{i}\tau_y} & -p_y^-e^{\mathbf{i}\tau_y} & \tilde{c}e^{\mathbf{i}\tau_y} \\ \tilde{c}e^{-\mathbf{i}\tau_x} & \tilde{c}e^{\mathbf{i}\tau_x} & \tilde{c}e^{\mathbf{i}\tau_x} & -p_x^-e^{-\mathbf{i}\tau_x} & \tilde{c}e^{-\mathbf{i}\tau_y} & \tilde{c}e^{-\mathbf{i}\tau_y} & \tilde{c}e^{\mathbf{i}\tau_y} & -p_y^-e^{\mathbf{i}\tau_y} \\ -e^{-\mathbf{i}\tau_x} & 0 & 0 & 0 & e^{-\mathbf{i}\tau_y} & 0 & 0 & 0 \\ 0 & -e^{\mathbf{i}\tau_x} & 0 & 0 & 0 & e^{-\mathbf{i}\tau_y} & 0 & 0 \\ 0 & 0 & -e^{\mathbf{i}\tau_x} & 0 & 0 & 0 & e^{\mathbf{i}\tau_y} & 0 \\ 0 & 0 & 0 & -e^{-\mathbf{i}\tau_x} & 0 & 0 & 0 & e^{\mathbf{i}\tau_y} \end{pmatrix}, \tag{3.2.80}$$

$$\tau_\alpha = \frac{1}{2}\tilde{\lambda}_\alpha \quad \alpha = x, y\,, \tag{3.2.81}$$

$$\tilde{\lambda}_\alpha = \lambda_\alpha \sigma_t \Delta\alpha \quad \alpha = x, y\,. \tag{3.2.82}$$

We note that

$$\Big(\tilde{\mathbf{A}}_{2D}^{-1}\tilde{\mathbf{B}}_{2D} - \omega\mathbf{I}\Big)\vec{a} = 0\,, \tag{3.2.83}$$

and hence ω is the eigenvalue of matrix $\tilde{\mathbf{T}}_{2D} = \tilde{\mathbf{A}}_{2D}^{-1}\tilde{\mathbf{B}}_{2D}$ and $\vec{a}$ is the associated eigenvector.

If we consider the case of a spatially constant mode, i.e. $\lambda_x = \lambda_y$=0, then the matrix of Eq. (3.2.77) has the following form:

$$\omega\tilde{\mathbf{A}}_{2D} - \tilde{\mathbf{B}}_{2D} = \omega\mathbf{A}_{2D} - \mathbf{B}_{2D} =$$
$$\begin{pmatrix} p_x^+\omega + p_x^- & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & p_y^+\omega + p_y^- & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) \\ -\tilde{c}(\omega+1) & p_x^+\omega + p_x^- & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & p_y^+\omega + p_y^- & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) \\ -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & p_x^+\omega + p_x^- & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & p_y^+\omega + p_y^- & -\tilde{c}(\omega+1) \\ -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & p_x^+\omega + p_x^- & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & -\tilde{c}(\omega+1) & p_y^+\omega + p_y^- \\ \omega+1 & 0 & 0 & 0 & -(\omega+1) & 0 & 0 & 0 \\ 0 & \omega+1 & 0 & 0 & 0 & -(\omega+1) & 0 & 0 \\ 0 & 0 & \omega+1 & 0 & 0 & 0 & -(\omega+1) & 0 \\ 0 & 0 & 0 & \omega+1 & 0 & 0 & 0 & -(\omega+1) \end{pmatrix}. \tag{3.2.84}$$

Analysis of the matrix (3.2.84) shows that in this case $\omega = -1$ is the eigenvalue of multiplicity 4 without regard to the value of scattering ratio c, total cross section σ_t and cell dimensions Δx and Δy.

If $\tilde{\lambda}_x = \tilde{\lambda}_y = \pi$, then the matrix of Eq. (3.2.77) has the following form:

$$\omega\tilde{\mathbf{A}}_{2D} - \tilde{\mathbf{B}}_{2D} =$$
$$\begin{pmatrix}
p_x^+\omega - p_x^- & \tilde{c}(\omega-1) & \tilde{c}(\omega-1) & \tilde{c}(1-\omega) & p_y^+\omega - p_y^- & \tilde{c}(1-\omega) & \tilde{c}(\omega-1) & \tilde{c}(\omega-1) \\
\tilde{c}(1-\omega) & p_x^+\omega - p_x^- & \tilde{c}(\omega-1) & \tilde{c}(1-\omega) & \tilde{c}(1-\omega) & p_y^+\omega - p_y^- & \tilde{c}(\omega-1) & \tilde{c}(\omega-1) \\
\tilde{c}(1-\omega) & \tilde{c}(\omega-1) & p_x^+\omega - p_x^- & \tilde{c}(1-\omega) & \tilde{c}(1-\omega) & \tilde{c}(1-\omega) & p_y^+\omega - p_y^- & \tilde{c}(\omega-1) \\
\tilde{c}(1-\omega) & \tilde{c}(\omega-1) & \tilde{c}(\omega-1) & p_x^+\omega - p_x^- & \tilde{c}(1-\omega) & \tilde{c}(1-\omega) & \tilde{c}(\omega-1) & p_y^+\omega - p_y^- \\
\omega-1 & 0 & 0 & 0 & 1-\omega & 0 & 0 & 0 \\
0 & 1-\omega & 0 & 0 & 0 & 1-\omega & 0 & 0 \\
0 & 0 & 1-\omega & 0 & 0 & 0 & \omega-1 & 0 \\
0 & 0 & 0 & \omega-1 & 0 & 0 & 0 & \omega-1
\end{pmatrix} . \tag{3.2.85}$$

In this case $\omega = 1$ is the eigenvalue of multiplicity 4.

If $\tilde{\lambda}_x = \tilde{\lambda}_y = \frac{\pi}{2}$, then the matrix of Eq. (3.2.77) has the following form:

$$\omega\tilde{\mathbf{A}}_{2D} - \tilde{\mathbf{B}}_{2D} =$$
$$\begin{pmatrix}
e^{\mathbf{i}\frac{\pi}{4}}(p_x^+\omega - \mathbf{i}p_x^-) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & e^{\mathbf{i}\frac{\pi}{4}}(p_y^+\omega - \mathbf{i}p_y^-) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) \\
-\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & e^{-\mathbf{i}\frac{\pi}{4}}(p_x^+\omega + \mathbf{i}p_x^-) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & e^{\mathbf{i}\frac{\pi}{4}}(p_y^+\omega - \mathbf{i}p_y^-) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) \\
-\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & e^{-\mathbf{i}\frac{\pi}{4}}(p_x^+\omega + \mathbf{i}p_x^-) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & e^{-\mathbf{i}\frac{\pi}{4}}(p_y^+\omega + \mathbf{i}p_y^-) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) \\
-\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & e^{\mathbf{i}\frac{\pi}{4}}(p_x^+\omega - \mathbf{i}p_x^-) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & -\tilde{c}e^{\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & -\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & e^{-\mathbf{i}\frac{\pi}{4}}(p_y^+\omega + \mathbf{i}p_y^-) \\
e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & 0 & 0 & 0 & -e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & 0 & 0 & 0 \\
0 & e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & 0 & 0 & 0 & -e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & 0 & 0 \\
0 & 0 & e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & 0 & 0 & 0 & -e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i}) & 0 \\
0 & 0 & 0 & e^{\mathbf{i}\frac{\pi}{4}}(\omega-\mathbf{i}) & 0 & 0 & 0 & -e^{-\mathbf{i}\frac{\pi}{4}}(\omega+\mathbf{i})
\end{pmatrix} . \tag{3.2.86}$$

In this case $\omega = \pm\mathbf{i}$.

Thus, the spectral radius $\rho_{2D} \geq 1$. Further analysis of ω and ρ_{2D} for various cases of c and optical thickness of cells showed that the spectral radius doesn't exceed 1 and hence

$$\rho_{2D} = 1\,. \tag{3.2.87}$$

Tables 3.2.2-3.2.4 show eigenvalue ω for selected wave numbers for problems with c=0.1 and three different cases of optical thicknesses of square cells ($\Delta x = \Delta y$), namely, 0.1, 1, and 100. The eigenvalues for high scattering ratio c=0.999 are listed in Tables 3.2.5-3.2.7. Figures 3.2.6-3.2.11 present plots of $|\omega|$ versus

$$\tilde{\lambda}_x = \lambda_x\sigma_t\Delta x\,, \quad \tilde{\lambda}_y = \lambda_y\sigma_t\Delta y \tag{3.2.88}$$

in case $c = 0.1, 0.9, 0.999$ and $\sigma_t\Delta x = \sigma_t\Delta y = 0.1, 1$.

3.2.2.4 Analysis of Eigenvectors

If $\omega = -1$, then the matrix (3.2.84) gives rise to

$$\omega\tilde{\mathbf{A}}_{2D} - \tilde{\mathbf{B}}_{2D} =$$
$$\begin{pmatrix} p_x^- - p_x^+ & 0 & 0 & 0 & p_y^- - p_y^+ & 0 & 0 & 0 \\ 0 & p_x^- - p_x^+ & 0 & 0 & 0 & p_y^- - p_y^+ & 0 & 0 \\ 0 & 0 & p_x^- - p_x^+ & 0 & 0 & 0 & p_y^- - p_y^+ & 0 \\ 0 & 0 & 0 & p_x^- - p_x^+ & 0 & 0 & 0 & p_y^- - p_y^+ \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.2.89}$$

and hence the equations for the eigenvector are the following:

$$(p_x^- - p_x^+)a_{x,l} + (p_y^- - p_y^+)a_{y,l} = 0\,,\ l = 1, ..., 4 \tag{3.2.90}$$

and hence

$$\nu_x a_{x,l} + \nu_y a_{y,l} = 0\,,\ l = 1, ..., 4\,. \tag{3.2.91}$$

Thus, if $\Delta x = \Delta y$, then

$$a_{x,l} = -a_{y,l}\,,\ l = 1, ..., 4\,. \tag{3.2.92}$$

Analysis of the matrix (3.2.85) for $\omega = 1$ leads to similar results.
If $\omega = \mathbf{i}$, then the matrix (3.2.86) gives rise to

$$\omega\tilde{\mathbf{A}}_{2D} - \tilde{\mathbf{B}}_{2D} =$$
$$\begin{pmatrix} e^{\mathbf{i}\frac{\pi}{4}}(p_x^+ - p_x^-) & -2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & 0 & e^{\mathbf{i}\frac{\pi}{4}}(p_y^+ - p_y^-) & 0 & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} \\ 0 & e^{-\mathbf{i}\frac{\pi}{4}}(p_x^+ + p_x^-) & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & 0 & 0 & e^{\mathbf{i}\frac{\pi}{4}}(p_y^+ - p_y^-) & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} \\ 0 & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & e^{-\mathbf{i}\frac{\pi}{4}}(p_x^+ + p_x^-) & 0 & 0 & 0 & e^{-\mathbf{i}\frac{\pi}{4}}(p_y^+ + p_y^-) & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} \\ 0 & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & e^{\mathbf{i}\frac{\pi}{4}}(p_x^+ - p_x^-) & 0 & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & e^{-\mathbf{i}\frac{\pi}{4}}(p_y^+ + p_y^-) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & 0 & 0 & 0 & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\tilde{c}e^{-\mathbf{i}\frac{\pi}{4}} \end{pmatrix}. \tag{3.2.93}$$

The equations for the eigenvector are the following:

$$(p_x^+ - p_x^-)a_{x,1} + (p_y^+ - p_y^-)a_{y,1} = 0\,, \tag{3.2.94}$$

$$a_{x,l} = a_{y,l} = 0\,, \quad l = 2, 3, 4\,, \tag{3.2.95}$$

and hence

$$\nu_x a_{x,1} + \nu_y a_{y,1} = 0\,, \tag{3.2.96}$$

$$a_{x,l} = a_{y,l} = 0\,, \quad l = 2, 3, 4\,, \tag{3.2.97}$$

If $\omega = -\mathbf{i}$, then the matrix (3.2.86) gives rise to

$$\omega\tilde{\mathbf{A}}_{2D} - \tilde{\mathbf{B}}_{2D} =$$
$$\begin{pmatrix} -e^{\mathbf{i}\frac{\pi}{4}}(p_x^+ + p_x^-) & 0 & 0 & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & -e^{\mathbf{i}\frac{\pi}{4}}(p_y^+ + p_y^-) & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & 0 \\ 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & -e^{-\mathbf{i}\frac{\pi}{4}}(p_x^+ - p_x^-) & 0 & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & -e^{\mathbf{i}\frac{\pi}{4}}(p_y^+ + p_y^-) & 0 & 0 \\ 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & -e^{-\mathbf{i}\frac{\pi}{4}}(p_x^+ - p_x^-) & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & -e^{-\mathbf{i}\frac{\pi}{4}}(p_y^+ - p_y^-) & 0 \\ 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & 0 & -e^{\mathbf{i}\frac{\pi}{4}}(p_x^+ + p_x^-) & -2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & 0 & -e^{-\mathbf{i}\frac{\pi}{4}}(p_y^+ - p_y^-) \\ -2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & 0 & 0 & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2\tilde{c}e^{\mathbf{i}\frac{\pi}{4}} & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.2.98}$$

As a result we obtain the equations for the eigenvector

$$(p_x^+ - p_x^-)a_{x,3} + (p_y^+ - p_y^-)a_{y,3} = 0\,, \tag{3.2.99}$$

$$a_{x,l} = a_{y,l} = 0\,, \quad l = 1,2,4\,, \tag{3.2.100}$$

and hence

$$\nu_x a_{x,3} + \nu_y a_{y,3} = 0\,, \tag{3.2.101}$$

$$a_{x,l} = a_{y,l} = 0\,, \quad l = 1,2,4\,. \tag{3.2.102}$$

Tables 3.2.8-3.2.13 present calculated eigenvectors for some wave numbers in case c=0.1, 0.999.

3.2.2.5 Analysis of Iterative Error Modes of the Cell-Average Scalar Flux

We now analyze iterative errors in the cell-average scalar flux. We use Eq. (3.2.60) to get error in s-th iterate in the following way:

$$\delta\phi_{i,j}^{(s)} = \frac{4\pi}{c}\sum_m (\gamma_m^x \delta\psi_{m,i,j}^{x,-(s-1)} + \gamma_m^y \delta\psi_{m,i,j}^{y,-(s-1)})\,, \tag{3.2.103}$$

In S$_2$ case we have

$$\gamma_m^\alpha = \gamma^\alpha = \frac{\pi\eta\nu_\alpha}{1+2(\nu_x+\nu_y)}\,, \tag{3.2.104}$$

$$\eta = \frac{1}{2\pi}\left[\frac{1}{c} - \frac{1}{1+2(\nu_x+\nu_y)}\right]^{-1}. \tag{3.2.105}$$

Thus,

$$\delta\phi_{i,j}^{(s)} = \frac{4\pi}{c}\Big[\gamma^x\Big(\delta\psi_{1,i-1/2,j}^{(s-1)} + \delta\psi_{2,i+1/2,j}^{(s-1)} + \delta\psi_{3,i+1/2,j}^{(s-1)} + \delta\psi_{4,i-1/2,j}^{(s-1)}\Big) + \gamma^y\Big(\delta\psi_{1,i,j-1/2}^{(s-1)} + \delta\psi_{2,i,j-1/2}^{(s-1)} + \delta\psi_{3,i,j+1/2}^{(s-1)} + \delta\psi_{4,i,j+1/2}^{(s-1)}\Big)\Big]. \tag{3.2.106}$$

This leads to

$$\delta\phi_{i,j}^{(s)} = \frac{4\pi}{c}e^{\mathbf{i}\sigma_t(\lambda_x x_i + \lambda_y y_j)}\omega^{s-1}\Big[\gamma^x\Big((a_{x,1}+a_{x,4})e^{-\mathbf{i}\tau_x} + (a_{x,2}+a_{x,3})e^{\mathbf{i}\tau_x}\Big) + \gamma^y\Big((a_{y,1}+a_{y,2})e^{-\mathbf{i}\tau_y} + (a_{y,3}+a_{y,4})e^{\mathbf{i}\tau_y}\Big)\Big]. \tag{3.2.107}$$

In case $\lambda_x = \lambda_y = 0$ and the eigenvalue $\omega = -1$ we have

$$\delta\phi_{i,j}^{(s)} = \frac{4\pi}{c}e^{\mathbf{i}\sigma_t(\lambda_x x_i + \lambda_y y_j)}(-1)^{s-1}\gamma^x\left[\sum_{m=1}^4 a_{x,m} + \frac{\nu_y}{\nu_x}\sum_{m=1}^4 a_{y,m}\right]. \tag{3.2.108}$$

Taking into account that components of the eigenvector meet Eq. (3.2.91), we get

$$\delta\phi_{i,j}^{(s)} = 0 \tag{3.2.109}$$

for an arbitrary rectangular grid. Similarly, we get that this is also true for $\omega = 1$, $\tilde{\lambda}_x = \tilde{\lambda}_y = \pi$.

If $\tilde{\lambda}_x = \tilde{\lambda}_y = \frac{\pi}{2}$ and $\omega = \pm\mathbf{i}$ we get

$$\delta\phi_{i,j}^{(s)} = \frac{4\pi}{c} e^{\mathbf{i}\sigma_t(\lambda_x x_i + \lambda_y y_j)} (\pm\mathbf{i})^{s-1} \frac{1}{\sqrt{2}} \gamma^x \Bigg[\Big((a_{x,1} + a_{x,4})(1-\mathbf{i}) + (a_{x,2} + a_{x,3})(1+\mathbf{i}) \Big) + \frac{\nu_y}{\nu_x} \Big((a_{y,1} + a_{y,2})(1-\mathbf{i}) + (a_{y,3} + a_{y,4})(1+\mathbf{i}) \Big) \Bigg] . \quad (3.2.110)$$

Using Eqs. (3.2.96) and (3.2.97) for $\omega = \mathbf{i}$ and Eqs. (3.2.101) and (3.2.102) for $\omega = -\mathbf{i}$ in Eq. (3.2.110), we obtain that in each case

$$\delta\phi_{i,j}^{(s)} = 0 \quad (3.2.111)$$

for arbitrary rectangular cells.

3.2.2.6 Summary

The main results of the analysis of ITMM in 2D Cartesian geometry are the following:

1. The spectral radius of ITTM
$$\rho_{2D} = \sup_{\lambda_x, \lambda_y} |\omega(\lambda_x, \lambda_y)| = 1 \, ,$$
It does not depend on values of c, Δx, Δy.

2. The slowest convergent modes of the errors in the face-averaged angular fluxes associated with $|\omega| = 1$ are anisotropic. The dependence of the modes on the angular variable is given by the following relations:
$$\nu_x a_{x,l} + \nu_y a_{y,l} = 0 \, , \; l = 1, ..., 4$$
for $\lambda_x = \lambda_y = 0$, $\omega = -1$ and $\lambda_x = \lambda_y = \pi$, $\omega = 1$,
$$\nu_x a_{x,1} + \nu_y a_{y,1} = 0 \, , \quad a_{x,l} = a_{y,l} = 0 \, , \quad l = 2, 3, 4$$
for $\tilde{\lambda}_x = \tilde{\lambda}_y = \frac{\pi}{2}$, $\omega = \mathbf{i}$,
$$\nu_x a_{x,3} + \nu_y a_{y,3} = 0 \, , \quad a_{x,l} = a_{y,l} = 0 \, , \quad l = 1, 2, 4$$
for $\tilde{\lambda}_x = \tilde{\lambda}_y = \frac{\pi}{2}$, $\omega = -\mathbf{i}$.

3. If $|\omega| = 1$, then $\delta\phi_{i,j}^{(s)} = 0$ for any c, Δx and Δy. Thus, the error in the cell-averaged scalar flux decreases.

Table 3.2.2: Eigenvalues ω_k for c=0.1, $\sigma_t \Delta x = \sigma_t \Delta y = 0.1$ ($\tilde{\lambda}_\alpha = \lambda_\alpha \sigma_t \Delta\alpha$)

$\tilde{\lambda}_x$	$\tilde{\lambda}_y$		ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8
		$Re(\omega)$	-1	-1	-1	-1	0.924981	0.916992	0.916992	0.916992
0	0	$Im(\omega)$	0	0	0	0	0	0	0	0
		$Re(\omega)$	-0.698636	-0.698636	-0.698193	-0.698193	0.659129	0.659129	0.656689	0.656689
0	$\pi/2$	$Im(\omega)$	0.698637	-0.698637	-0.698193	0.698193	-0.659122	0.659122	-0.656689	0.656689
		$Re(\omega)$	-0.809017	-0.809017	-0.992173	-0.991763	0.928244	0.924608	0.743481	0.743481
$\pi/5$	$\pi/5$	$Im(\omega)$	0.587785	-0.587785	0	0	0	0	-0.540152	0.540152
		$Re(\omega)$	0	0	-0.959676	0.959676	0.957597	-0.957597	0	0
$\pi/2$	$\pi/2$	$Im(\omega)$	-1	1	0	0	0	0	0.918983	-0.918983
		$Re(\omega)$	1	1	1	1	-0.924981	-0.916992	-0.916992	-0.916992
π	π	$Im(\omega)$	0	0	0	0	0	0	0	0

Table 3.2.3: Eigenvalues ω_k for c=0.1, $\sigma_t \Delta x = \sigma_t \Delta y = 1$ ($\tilde{\lambda}_\alpha = \lambda_\alpha \sigma_t \Delta\alpha$)

$\tilde{\lambda}_x$	$\tilde{\lambda}_y$		ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8
		$Re(\omega)$	-1	-1	-1	-1	0.439148	0.395661	0.395661	0.395661
0	0	$Im(\omega)$	0	0	0	0	0	0	0	0
		$Re(\omega)$	-0.625022	-0.625022	-0.620826	-0.620826	0.333724	0.333724	0.318657	0.318657
0	$\pi/2$	$Im(\omega)$	-0.625153	0.625153	-0.620826	0.620826	-0.333123	0.333123	0.318657	-0.318657
		$Re(\omega)$	-0.809017	-0.809017	-0.923312	-0.91931	0.451847	0.430389	0.328959	0.328959
$\pi/5$	$\pi/5$	$Im(\omega)$	-0.587785	0.587785	0	0	0	0	0.237846	-0.237846
		$Re(\omega)$	0	0	-0.64582	0.64582	0.629016	-0.629016	0	0
$\pi/2$	$\pi/2$	$Im(\omega)$	1	-1	0	0	0	0	-0.405992	0.405992
		$Re(\omega)$	1	1	1	1	-0.439148	-0.395661	-0.395661	-0.39566
π	π	$Im(\omega)$	0	0	0	0	0	0	0	0

Table 3.2.4: Eigenvalues ω_k for c=0.1, $\sigma_t \Delta x = \sigma_t \Delta y = 100$ ($\tilde{\lambda}_\alpha = \lambda_\alpha \sigma_t \Delta\alpha$)

$\tilde{\lambda}_x$	$\tilde{\lambda}_y$		ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8
		$Re(\omega)$	-1	-1	-1	-1	-0.954855	-0.954855	-0.954855	-0.949964
0	0	$Im(\omega)$	0	0	0	0	0	0	0	0
		$Re(\omega)$	-0.977167	-0.977167	-2.607E-04	-2.607E-04	-0.977133	-2.909E-04	-2.909E-04	-0.974694
0	$\pi/2$	$Im(\omega)$	-2.607E-04	2.607E-04	0.977167	-0.977167	0	0.975913	-0.975913	0
		$Re(\omega)$	-0.809017	-0.809017	-0.790755	-0.790755	-0.78982	-0.78982	-0.771451	-0.771451
$\pi/5$	$\pi/5$	$Im(\omega)$	-0.587785	0.587785	-0.574074	0.574074	-0.573347	0.573347	-0.560485	0.560485
		$Re(\omega)$	0	0	0	0	0	0	0	0
$\pi/2$	$\pi/2$	$Im(\omega)$	1	-1	0.977167	-0.977167	-0.975982	0.975982	-0.953562	0.953562
		$Re(\omega)$	1	1	1	1	0.954855	0.954855	0.954855	0.949964
π	π	$Im(\omega)$	0	0	0	0	0	0	0	0

Table 3.2.5: Eigenvalues ω_k for c=0.999, $\sigma_t\Delta x = \sigma_t\Delta y = 0.1$ ($\tilde{\lambda}_\alpha = \lambda_\alpha\sigma_t\Delta\alpha$)

$\tilde{\lambda}_x$	$\tilde{\lambda}_y$		ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8
		$Re(\omega)$	-1	-1	-1	-1	0.999913	0.916992	0.916992	0.916992
0	0	$Im(\omega)$	0	0	0	0	0	0	0	0
		$Re(\omega)$	-0.702565	-0.702565	-0.698193	-0.698193	0.681791	0.681791	0.656689	0.656689
0	$\pi/2$	$Im(\omega)$	-0.702701	0.702701	0.698193	-0.698193	-0.68102	0.68102	-0.656689	0.656689
		$Re(\omega)$	-0.809017	-0.809017	-0.99587	-0.991763	0.961714	0.758905	0.758905	0.924608
$\pi/5$	$\pi/5$	$Im(\omega)$	0.587785	-0.587785	0	0	0	-0.549512	0.549512	0
		$Re(\omega)$	0	0	0.978567	-0.978567	-0.957597	0.957597	0	0
$\pi/2$	$\pi/2$	$Im(\omega)$	-1	1	0	0	0	0	0.937035	-0.937035
		$Re(\omega)$	1	1	1	1	-0.999913	-0.916992	-0.916992	-0.916992
π	π	$Im(\omega)$	0	0	0	0	0	0	0	0

Table 3.2.6: Eigenvalues ω_k for c=0.999, $\sigma_t\Delta x = \sigma_t\Delta y = 1$ ($\tilde{\lambda}_\alpha = \lambda_\alpha\sigma_t\Delta\alpha$)

$\tilde{\lambda}_x$	$\tilde{\lambda}_y$		ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8
		$Re(\omega)$	-1	-1	-1	-1	0.999134	0.395661	0.395661	0.395661
0	0	$Im(\omega)$	0	0	0	0	0	0	0	0
		$Re(\omega)$	-0.657136	-0.657136	-0.620826	-0.620826	0.505835	0.505835	0.318657	0.318657
0	$\pi/2$	$Im(\omega)$	-0.670439	0.670439	0.620826	-0.620826	-0.438959	0.438959	0.318657	-0.318657
		$Re(\omega)$	-0.809017	-0.809017	-0.959487	-0.91931	0.823164	0.387908	0.387908	0.430389
$\pi/5$	$\pi/5$	$Im(\omega)$	-0.587785	0.587785	0	0	0	-0.218091	0.218091	0
		$Re(\omega)$	0	0	-0.811703	0.811703	-0.629016	0.629016	0	0
$\pi/2$	$\pi/2$	$Im(\omega)$	1	-1	0	0	0	0	-0.487234	0.487234
		$Re(\omega)$	1	1	1	1	-0.999134	-0.395661	-0.395661	-0.395661
π	π	$Im(\omega)$	0	0	0	0	0	0	0	0

Table 3.2.7: Eigenvalues ω_k for c=0.999, $\sigma_t\Delta x = \sigma_t\Delta y = 100$ ($\tilde{\lambda}_\alpha = \lambda_\alpha\sigma_t\Delta\alpha$)

$\tilde{\lambda}_x$	$\tilde{\lambda}_y$		ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7	ω_8
		$Re(\omega)$	-1	-1	-1	-1	-0.954855	-0.954855	-0.954855	0.916992
0	0	$Im(\omega)$	0	0	0	0	0	0	0	0
		$Re(\omega)$	-0.489195	-0.489195	-0.977167	-2.607E-04	-2.607E-04	-0.977167	-0.977161	0.936619
0	$\pi/2$	$Im(\omega)$	0.846985	-0.846985	-2.607E-04	-0.977167	0.977167	2.607E-04	0	0
		$Re(\omega)$	-0.809017	-0.809017	-0.790755	-0.790755	-0.971274	0.567405	-0.78172	0.922572
$\pi/5$	$\pi/5$	$Im(\omega)$	0.587785	-0.587785	0.574074	-0.574074	0	-0.78172	-0.567405	0
		$Re(\omega)$	0	0	0	0	0	0	0.946599	-0.946599
$\pi/2$	$\pi/2$	$Im(\omega)$	-1	1	-0.977167	0.977167	-0.965949	0.965949	0	0
		$Re(\omega)$	1	1	1	1	0.954855	0.954855	0.954855	-0.916992
π	π	$Im(\omega)$	0	0	0	0	0	0	0	0

Table 3.2.8: Eigenvectors $\vec{a}$ for c=0.1, $\sigma_t \Delta x = \sigma_t \Delta y = 1$, $\tilde{\lambda}_x = \tilde{\lambda}_y$=0

k	ω_k	$a_{x,1}$	$a_{x,2}$	$a_{x,3}$	$a_{x,4}$	$a_{y,1}$	$a_{y,2}$	$a_{y,3}$	$a_{y,4}$
1	-1	1.784E-01	1.441E-01	-6.689E-01	-2.411E-03	-1.784E-01	-1.441E-01	6.689E-01	2.411E-03
2	-1	3.553E-01	5.712E-01	2.179E-01	0.000E+00	-3.553E-01	-5.712E-01	-2.179E-01	0.000E+00
3	-1	-5.847E-01	3.911E-01	-7.169E-02	-7.886E-04	5.847E-01	-3.911E-01	7.169E-02	7.886E-04
4	-1	4.369E-05	-9.278E-04	2.361E-03	-7.071E-01	-4.369E-05	9.278E-04	-2.361E-03	7.071E-01
5	4.391E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01
6	3.957E-01	-1.468E-01	-4.251E-02	-3.842E-01	5.736E-01	-1.468E-01	-4.251E-02	-3.842E-01	5.736E-01
7	3.957E-01	-1.057E-01	-5.213E-01	4.148E-01	2.122E-01	-1.057E-01	-5.213E-01	4.148E-01	2.122E-01
8	3.957E-01	5.850E-01	-3.185E-01	-2.352E-01	-3.138E-02	5.850E-01	-3.185E-01	-2.352E-01	-3.138E-02

Table 3.2.9: Eigenvectors $\vec{a}$ for c=0.1, $\sigma_t \Delta x = \sigma_t \Delta y = 1$, $\tilde{\lambda}_x = \tilde{\lambda}_y = \frac{\pi}{2}$

k	ω_k	$a_{x,1}$	$a_{x,2}$	$a_{x,3}$	$a_{x,4}$
1	i	0.707107	0	0	0
2	-i	0	0	-0.707107	0
3	-0.64582	2.8219E-4 +i1.18733E-2	-0.45561-i0.205617	-5.1413E-3+i1.07062E-2	0.499859
4	0.64582	2.8219E-4 -i1.18733E-2	0.45561 -i0.205617	5.1413E-3+i1.07062E-2	0.499859
5	0.629016	0	0.450694 -i0.216506	0	-0.5
6	-0.629016	0	-0.450694 -i0.216507	0	-0.5
7	-i0.405992	0.706768	i1.83158E-2	i9.10803 E-3	-7.73811E-3
8	i0.405992	-i9.10803 E-3	-7.73811E-3	0.706768	-i1.83158E-2

k	ω_k	$a_{y,1}$	$a_{y,2}$	$a_{y,3}$	$a_{y,4}$
1	i	-0.707107	0	0	0
2	-i	0	0	0.707107	0
3	-0.64582	2.8219E-4 +i1.18733E-2	0.499859	-5.1413E-3+i1.07062E-2	-0.45561-i0.205617
4	0.64582	2.8219E-4 -i1.18733E-2	0.499859	5.1413E-3+i1.07062E-3	0.45561-i0.205618
5	0.629016	0	0.5	0	-0.450694 -i0.216506
6	-0.629016	0	0.5	0	0.450694 +i0.216507
7	-i0.405992	0.706768	-7.73811E-3	i9.10803 E-3	i1.83158E-2
8	i0.405992	-i9.10803 E-3	-i1.83158E-2	0.706768	-7.73811E-3

Table 3.2.10: Eigenvectors $\vec{a}$ for c=0.1, $\sigma_t \Delta x = \sigma_t \Delta y = 1$, $\tilde{\lambda}_x = \tilde{\lambda}_y = \pi$

k	ω_k	$a_{x,1}$	$a_{x,2}$	$a_{x,3}$	$a_{x,4}$	$a_{y,1}$	$a_{y,2}$	$a_{y,3}$	$a_{y,4}$
1	1	-5.182E-01	3.993E-01	-1.306E-01	-2.345E-01	5.182E-01	3.993E-01	1.306E-01	-2.345E-01
2	1	-7.071E-01	4.375E-17	1.464E-18	-2.982E-19	7.071E-01	6.378E-17	2.304E-18	1.924E-18
3	1	3.359E-01	-2.559E-01	-5.635E-01	-6.463E-02	-3.359E-01	-2.559E-01	5.635E-01	-6.463E-02
4	1	-5.596E-01	4.302E-01	1.831E-02	3.729E-02	5.596E-01	4.302E-01	-1.831E-02	3.729E-02
5	-0.439148	3.536E-01	-3.536E-01	-3.536E-01	3.536E-01	3.536E-01	3.536E-01	-3.536E-01	-3.536E-01
6	-0.395661	1.173E-01	-4.783E-01	4.979E-01	-9.765E-02	1.173E-01	4.783E-01	4.979E-01	9.765E-02
7	-0.395661	4.794E-02	-4.967E-01	4.583E-02	-4.989E-01	4.794E-02	4.967E-01	4.583E-02	4.989E-01
8	-0.395661	5.512E-01	-4.593E-02	3.874E-01	-2.097E-01	5.512E-01	4.593E-02	3.874E-01	2.097E-01

Table 3.2.11: Eigenvectors $\vec{a}$ for c=0.999, $\sigma_t\Delta x = \sigma_t\Delta y = 1$, $\tilde{\lambda}_x = \tilde{\lambda}_y$=0

k	ω_k	$a_{x,1}$	$a_{x,2}$	$a_{x,3}$	$a_{x,4}$	$a_{y,1}$	$a_{y,2}$	$a_{y,3}$	$a_{y,4}$
1	-1	6.939E-17	1.479E-02	4.103E-01	-5.757E-01	0.000E+00	-1.479E-02	-4.103E-01	5.757E-01
2	-1	1.388E-17	-2.075E-02	-5.754E-01	-4.105E-01	-1.180E-16	2.075E-02	5.754E-01	4.105E-01
3	-1	-4.028E-02	7.055E-01	-2.544E-02	-1.041E-17	4.028E-02	-7.055E-01	2.544E-02	0.000E+00
4	-1	-7.060E-01	-4.025E-02	1.452E-03	-2.602E-18	7.060E-01	4.025E-02	-1.452E-03	0.000E+00
5	9.991E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01
6	3.957E-01	5.696E-01	-3.666E-01	-2.029E-01	2.776E-17	5.696E-01	-3.666E-01	-2.029E-01	0.000E+00
7	3.957E-01	-2.156E-01	-4.482E-01	2.047E-01	4.591E-01	-2.156E-01	-4.482E-01	2.047E-01	4.591E-01
8	3.957E-01	6.422E-02	-1.993E-01	5.403E-01	-4.053E-01	6.422E-02	-1.993E-01	5.403E-01	-4.053E-01

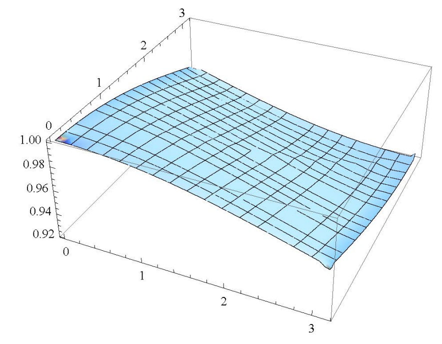
Table 3.2.12: Eigenvectors $\vec{a}$ for c=0.999, $\sigma_t\Delta x = \sigma_t\Delta y = 1$, $\tilde{\lambda}_x = \tilde{\lambda}_y = \frac{\pi}{2}$

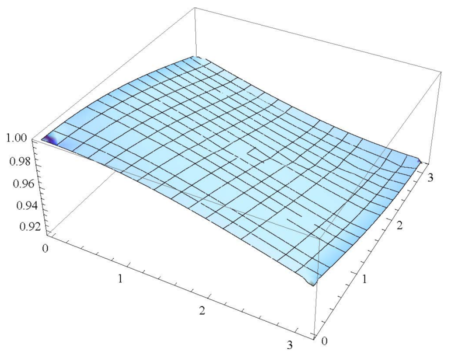
k	ω_k	$a_{x,1}$	$a_{x,2}$	$a_{x,3}$	$a_{x,4}$
1	i	-0.707107	0	0	0
2	-i	0	0	0.707107	0
3	-0.811703	-4.65483E-2 -i0.100065	0.487668	2.49754E-2+i1.07099E-2	-0.477245+i0.100287
4	0.811703	2.49754 E-2 -i0.107499	0.477245 -i0.100287	4.65483E-3+i0.100065	0.487668
5	-0.629016	0	0.450694 -i0.216506	0	-0.5
6	0.629016	0	0.450694 -i0.216507	0	-0.5
7	-i0.487234	0.685981	i0.147576	i7.11496E-2	-5.08811E-2
8	i0.487234	-i7.11496 E-2	-5.08811E-2	0.685981	-i0.147576

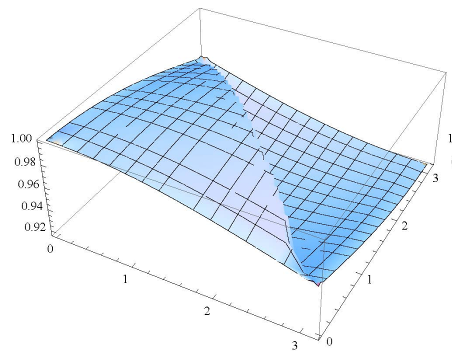
k	ω_k	$a_{y,1}$	$a_{y,2}$	$a_{y,3}$	$a_{y,4}$
1	i	0.707107	0	0	0
2	-i	0	0	-0.707107	0
3	-0.811703	-4.65483E-2 -i0.100065	-0.477245+i0.100287	2.49754E-2+i1.07099E-2	0.487668
4	0.811703	2.49754 E-2 -i0.107499	0.487668	4.65483E-3+i0.100065	0.477245 -i0.100287
5	-0.629016	0	0.5	0	-0.450694 +i0.216506
6	0.629016	0	0.5	0	0.450694 +i0.216507
7	-i0.487234	0.685981	-5.08811E-2	i7.11496 E-2	i0.147576
8	i0.487234	-i7.11496 E-2	-i0.147576	0.685981	-5.08811E-2

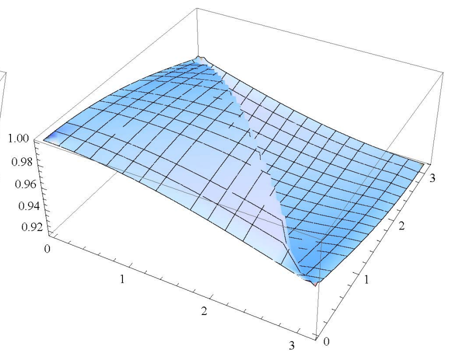
Table 3.2.13: Eigenvectors $\vec{a}$ for c=0.999, $\sigma_t\Delta x = \sigma_t\Delta y = 1$, $\tilde{\lambda}_x = \tilde{\lambda}_y = \pi$

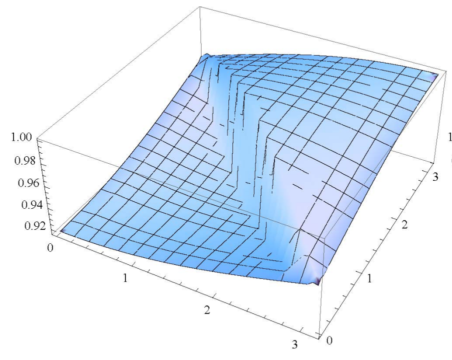
k	ω_k	$a_{x,1}$	$a_{x,2}$	$a_{x,3}$	$a_{x,4}$	$a_{y,1}$	$a_{y,2}$	$a_{y,3}$	$a_{y,4}$
1	1	1.883E-01	4.825E-01	-4.714E-01	9.720E-02	-1.883E-01	4.825E-01	4.714E-01	9.720E-02
2	1	-7.071E-01	-4.337E-19	8.457E-18	-3.730E-17	7.071E-01	-8.457E-18	-1.019E-17	1.540E-17
3	1	-3.498E-01	5.029E-01	3.513E-01	3.596E-02	3.498E-01	5.029E-01	-3.513E-01	3.596E-02
4	1	2.727E-01	-2.800E-01	-9.213E-02	5.820E-01	-2.727E-01	-2.800E-01	9.213E-02	5.820E-01
5	-0.999134	-3.536E-01	3.536E-01	3.536E-01	-3.536E-01	-3.536E-01	-3.536E-01	3.536E-01	3.536E-01
6	-0.395661	3.022E-01	-3.920E-01	4.307E-01	-2.636E-01	3.022E-01	3.920E-01	4.307E-01	2.636E-01
7	-0.395661	1.771E-01	-7.256E-02	5.897E-01	3.401E-01	1.771E-01	7.256E-02	5.897E-01	-3.401E-01
8	-0.39566	6.124E-01	2.041E-01	2.041E-01	-2.041E-01	6.124E-01	-2.041E-01	2.041E-01	2.041E-01

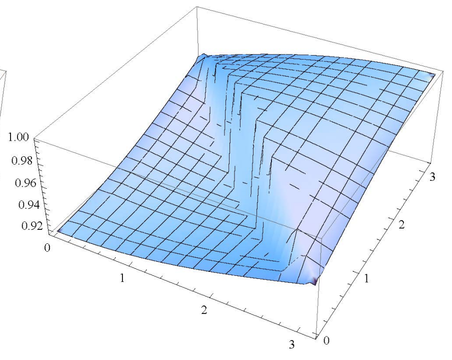

(a) $|\omega_1|$

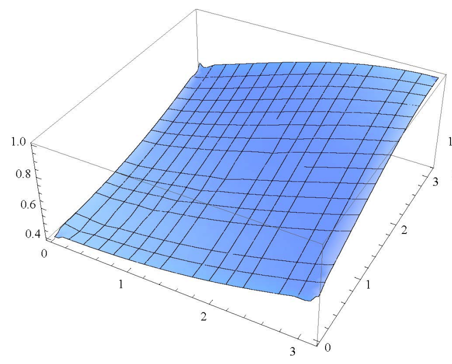

(b) $|\omega_2|$


(c) $|\omega_3|$


(d) $|\omega_4|$


(e) $|\omega_5|$


(f) $|\omega_6|$


(g) $|\omega_7|$

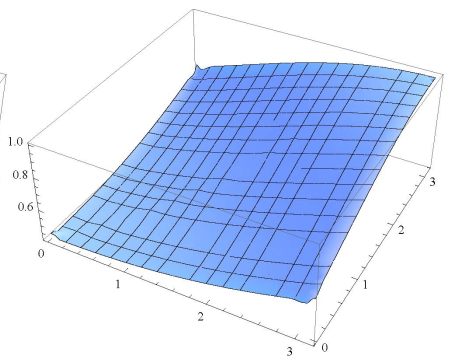
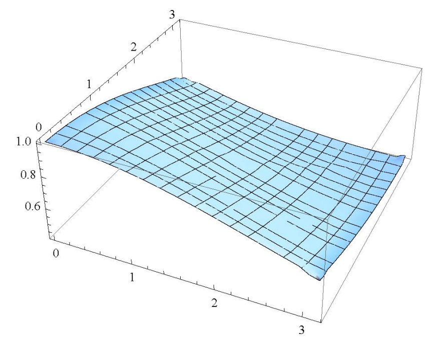
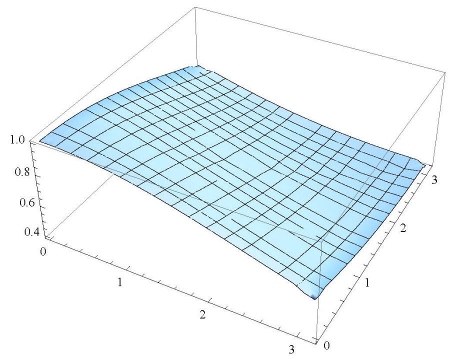
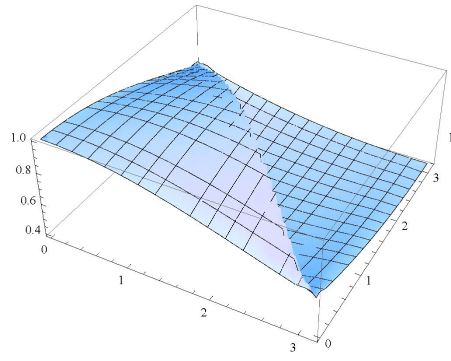
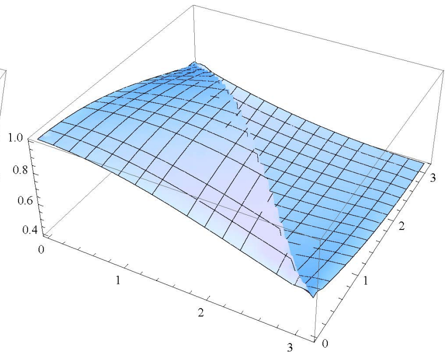

(h) $|\omega_8|$

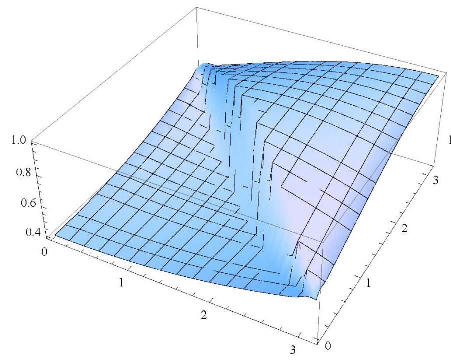
Figure 3.2.6: $\omega(\tilde{\lambda}_x, \tilde{\lambda}_y)$ for 2D, $c = 0.1$, $\sigma_t \Delta x$=$\sigma_t \Delta y$=0.1

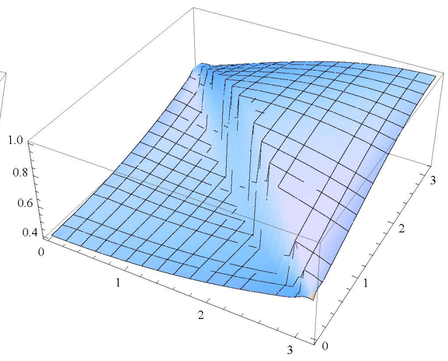

(a) $|\omega_1|$

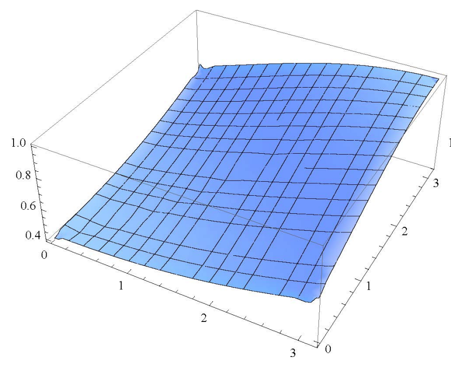

(b) $|\omega_2|$


(c) $|\omega_3|$


(d) $|\omega_4|$


(e) $|\omega_5|$


(f) $|\omega_6|$


(g) $|\omega_7|$

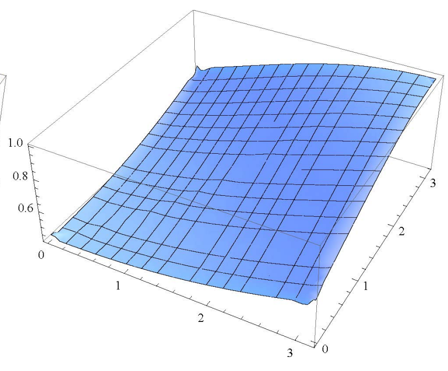
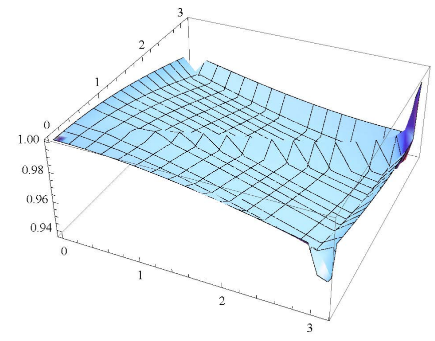
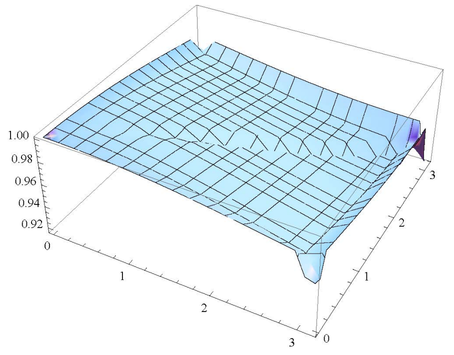
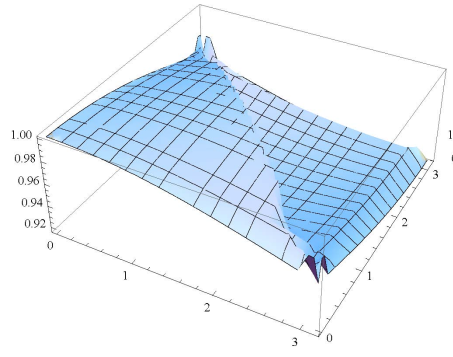
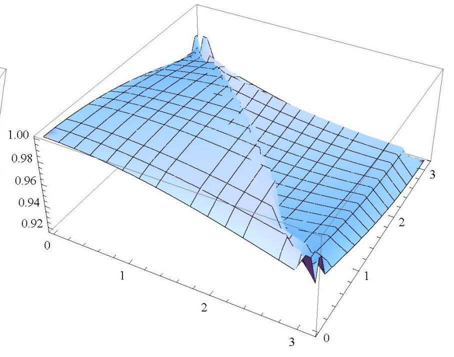

(h) $|\omega_8|$

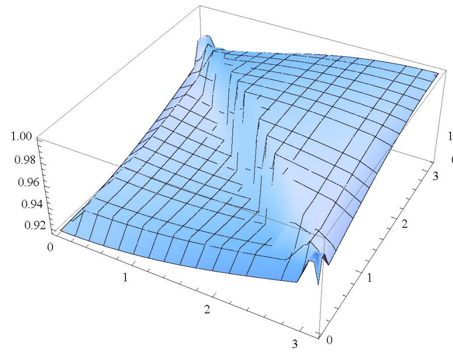
Figure 3.2.7: $\omega(\tilde{\lambda}_x, \tilde{\lambda}_y)$ for 2D, $c = 0.1$, $\sigma_t \Delta x$=$\sigma_t \Delta y$=1

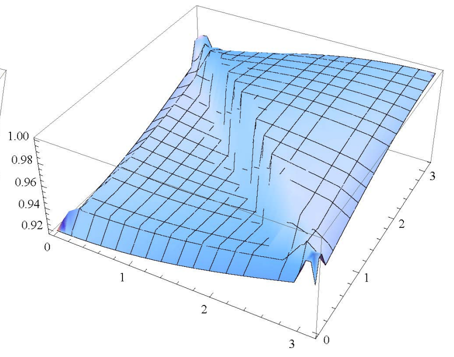

(a) $|\omega_1|$

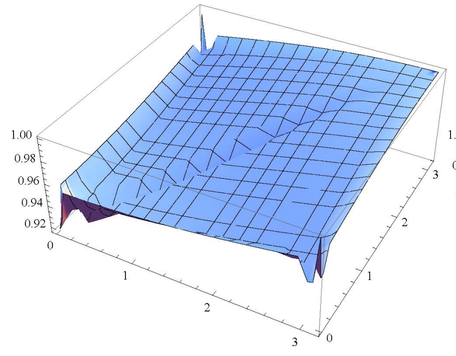

(b) $|\omega_2|$


(c) $|\omega_3|$


(d) $|\omega_4|$


(e) $|\omega_5|$


(f) $|\omega_6|$


(g) $|\omega_7|$

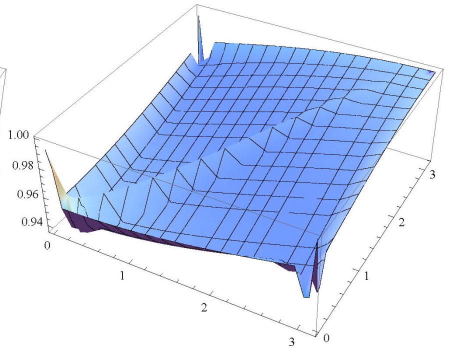
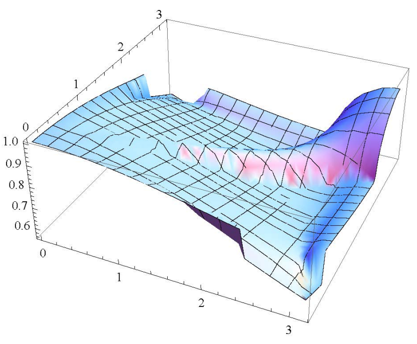
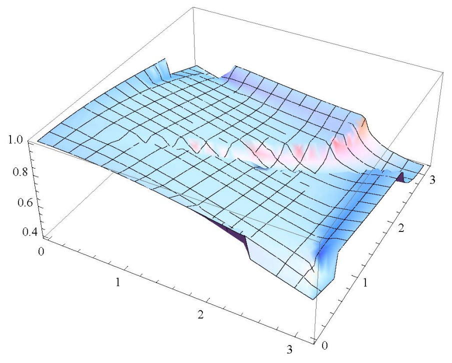
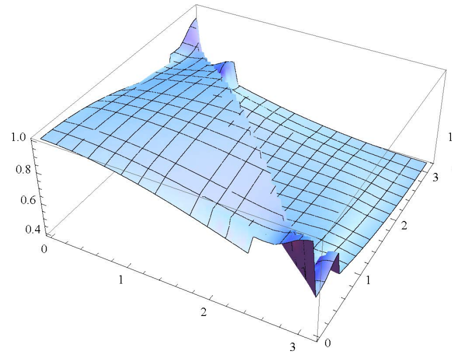
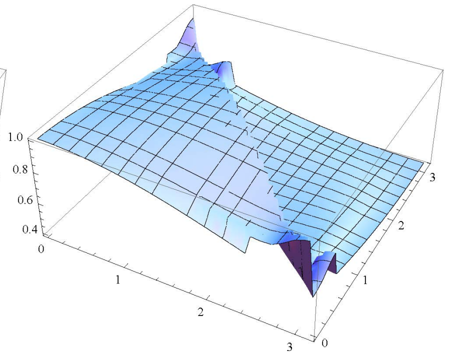

(h) $|\omega_8|$

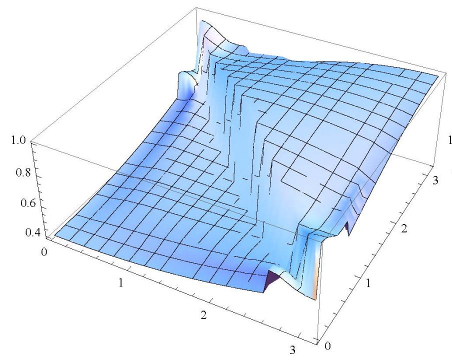
Figure 3.2.8: $\omega(\tilde{\lambda}_x, \tilde{\lambda}_y)$ for 2D, $c = 0.9$, $\sigma_t \Delta x$=$\sigma_t \Delta y$=0.1

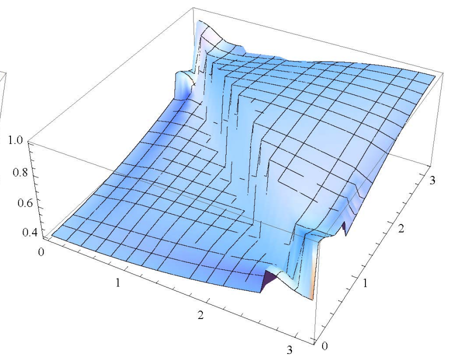

(a) $|\omega_1|$

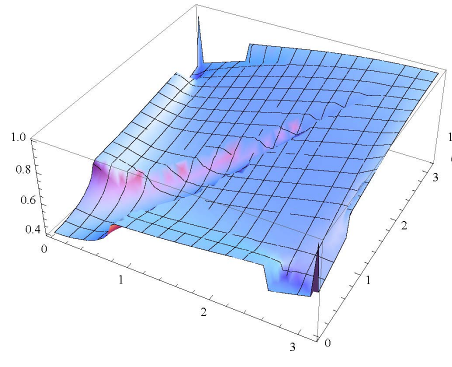

(b) $|\omega_2|$


(c) $|\omega_3|$


(d) $|\omega_4|$


(e) $|\omega_5|$


(f) $|\omega_6|$


(g) $|\omega_7|$

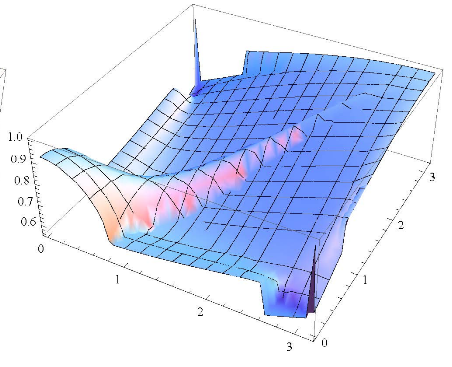
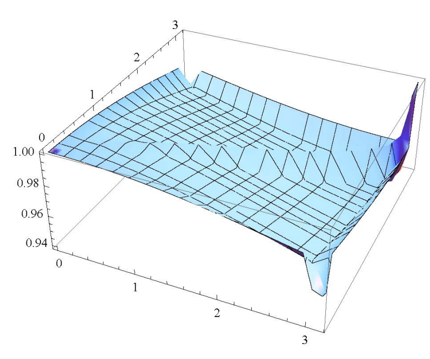
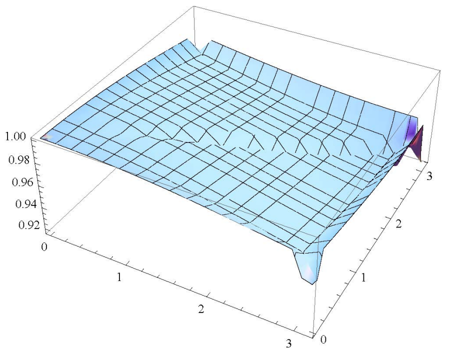
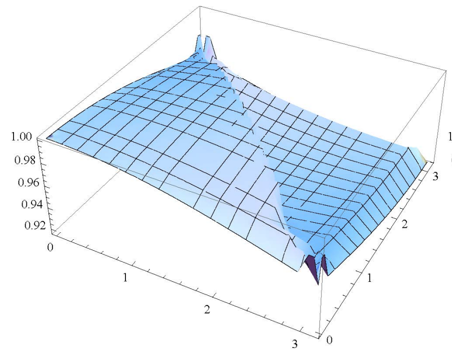
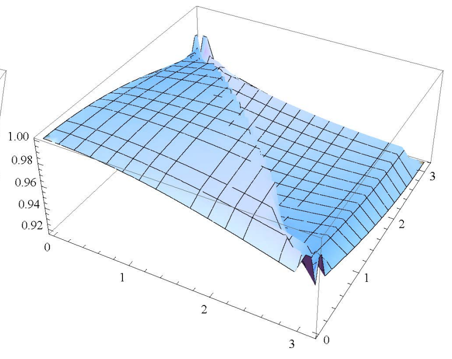

(h) $|\omega_8|$

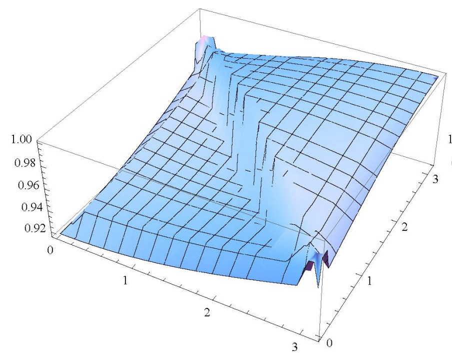
Figure 3.2.9: $\omega(\tilde{\lambda}_x, \tilde{\lambda}_y)$ for 2D, $c = 0.9$, $\sigma_t \Delta x$=$\sigma_t \Delta y$=1

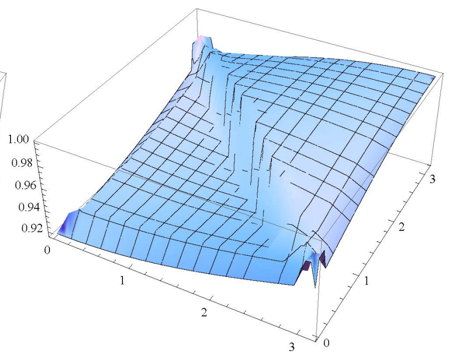

(a) $|\omega_1|$

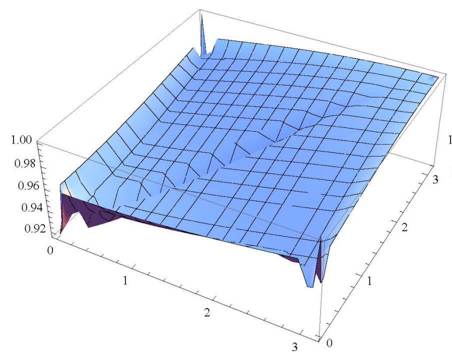

(b) $|\omega_2|$


(c) $|\omega_3|$


(d) $|\omega_4|$


(e) $|\omega_5|$


(f) $|\omega_6|$


(g) $|\omega_7|$

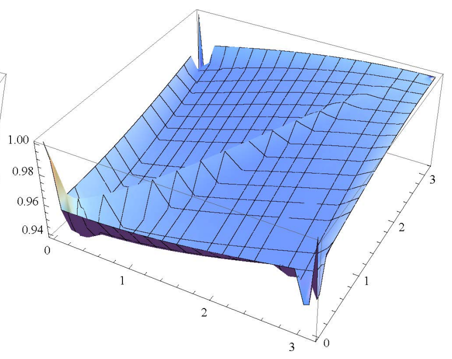
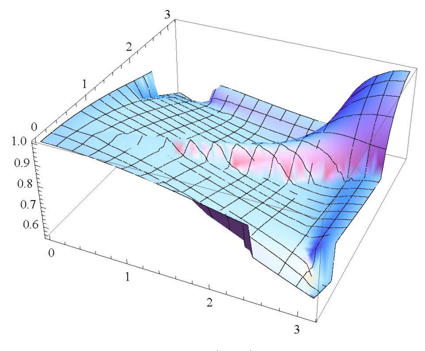
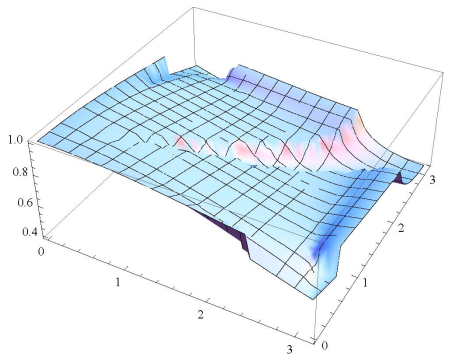
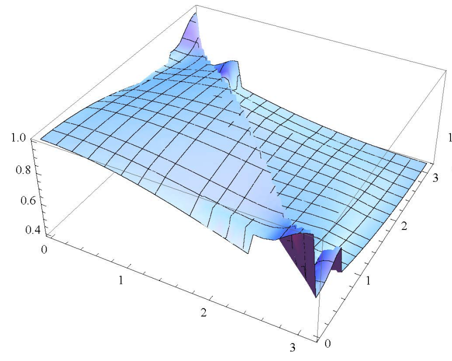
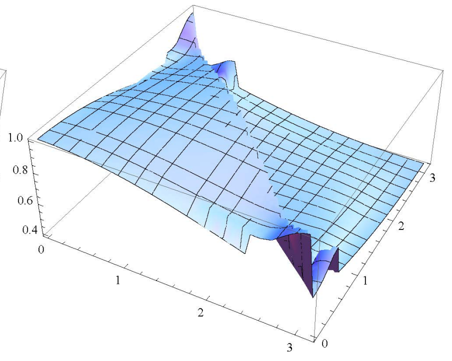

(h) $|\omega_8|$

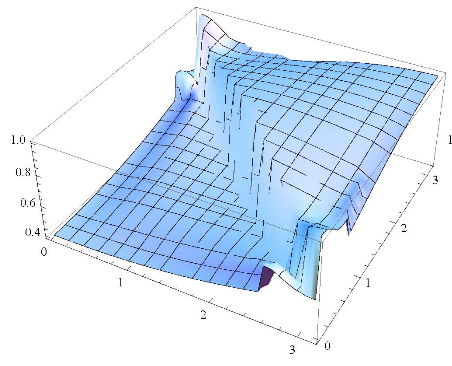
Figure 3.2.10: $\omega(\tilde{\lambda}_x, \tilde{\lambda}_y)$ for 2D, $c = 0.999$, $\sigma_t \Delta x$=$\sigma_t \Delta y$=0.1

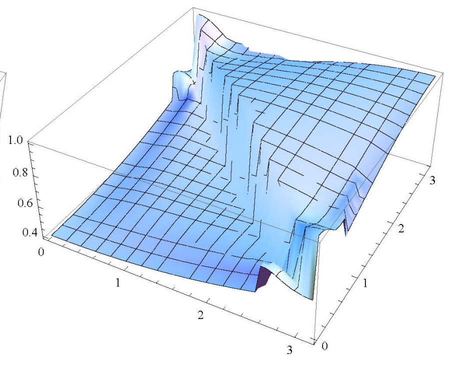

(a) $|\omega_1|$

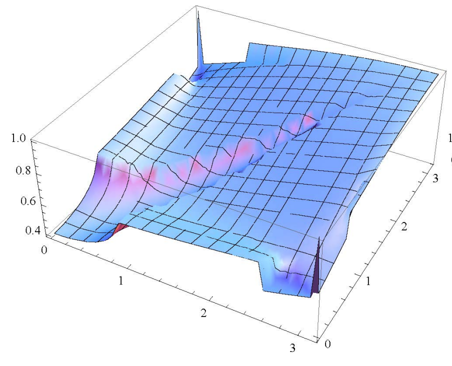

(b) $|\omega_2|$


(c) $|\omega_3|$


(d) $|\omega_4|$


(e) $|\omega_5|$


(f) $|\omega_6|$


(g) $|\omega_7|$

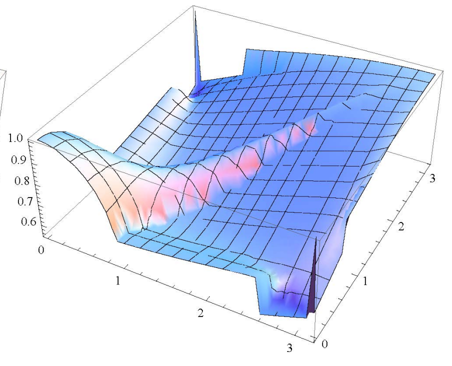

(h) $|\omega_8|$

Figure 3.2.11: $\omega(\tilde{\lambda}_x, \tilde{\lambda}_y)$ for 2D, $c = 0.999$, $\sigma_t \Delta x$=$\sigma_t \Delta y$=1

3.2.3 3D Cartesian Geometry

3.2.3.1 Formulation of the Computational Method

The DD method in 3D Cartesian geometry is defined as follows:

$$\sum_{\alpha=x,y,z} \nu_{\alpha,m}(\psi^{\alpha,+}_{m,i,j,k} - \psi^{\alpha,-}_{m,i,j,k}) + \psi_{m,i,j,k} = \frac{1}{4\pi}\left(c\phi_{i,j,k} + \frac{q}{\sigma_t}\right), \tag{3.2.112a}$$

$$\psi_{m,i,j,k} = \frac{1}{2}\left(\psi^{\alpha,+}_{m,i,j,k} + \psi^{\alpha,-}_{m,i,j,k}\right), \quad \alpha = x, y, z\,, \tag{3.2.112b}$$

$$\phi_{i,j,k} = \sum_{m=1}^{M} \psi_{m,i,j,k} w_m\,, \tag{3.2.112c}$$

$$\nu_{\alpha,m} = \frac{|\Omega_{\alpha,m}|}{\sigma_t \Delta\alpha}\,, \quad c = \frac{\sigma_s}{\sigma_t}\,, \tag{3.2.113}$$

$$\Omega_{x,m}>0,\ \Omega_{y,m}>0,\ \Omega_{z,m}>0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\pm1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\pm1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\pm1/2}, \tag{3.2.114a}$$

$$\Omega_{x,m}<0,\ \Omega_{y,m}>0,\ \Omega_{z,m}>0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\mp1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\pm1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\pm1/2}, \tag{3.2.114b}$$

$$\Omega_{x,m}<0,\ \Omega_{y,m}<0,\ \Omega_{z,m}>0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\mp1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\mp1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\pm1/2}, \tag{3.2.114c}$$

$$\Omega_{x,m}>0,\ \Omega_{y,m}<0,\ \Omega_{z,m}>0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\pm1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\mp1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\pm1/2}, \tag{3.2.114d}$$

$$\Omega_{x,m}>0,\ \Omega_{y,m}>0,\ \Omega_{z,m}<0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\pm1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\pm1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\mp1/2}, \tag{3.2.114e}$$

$$\Omega_{x,m}<0,\ \Omega_{y,m}>0,\ \Omega_{z,m}<0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\mp1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\pm1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\mp1/2}, \tag{3.2.114f}$$

$$\Omega_{x,m}<0,\ \Omega_{y,m}<0,\ \Omega_{z,m}<0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\mp1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\mp1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\mp1/2}, \tag{3.2.114g}$$

$$\Omega_{x,m}>0,\ \Omega_{y,m}<0,\ \Omega_{z,m}<0: \psi^{x,\pm}_{m,i,j,k}=\psi_{m,i\pm1/2,j,k},\ \psi^{y,\pm}_{m,i,j,k}=\psi_{m,i,j\mp1/2,k},\ \psi^{z,\pm}_{m,i,j,k}=\psi_{m,i,j,k\mp1/2}. \tag{3.2.114h}$$

The ITMM equations for iterative errors are the following:

$$\sum_{\alpha=x,y,z} \nu_{\alpha,m}\left(\delta\psi^{\alpha,+(s)}_{m,i,j,k} - \delta\psi^{\alpha,-(s-1)}_{m,i,j,k}\right) + \delta\psi^{(s)}_{m,i,j,k} = \frac{1}{4\pi} c\,\delta\phi^{(s)}_{i,j,k}\,, \tag{3.2.115a}$$

$$\delta\psi^{(s)}_{m,i,j,k} = \frac{1}{2}\left(\delta\psi^{\alpha,+(s)}_{m,i,j} + \delta\psi^{\alpha,-(s-1)}_{m,i,j}\right), \quad \alpha = x, y, z\,, \tag{3.2.115b}$$

$$m = 1, ..., M\,,$$

$$\delta\phi^{(s)}_{i,j,k} = \sum_{m=1}^{M} \delta\psi^{(s)}_{m,i,j,k} w_m\,. \tag{3.2.115c}$$

The system of equations for the iterative errors in face-averaged angular fluxes is given by

$$\sum_{\alpha=x,y,z}\left(p^{+}_{\alpha,m}\delta\psi^{\alpha,+(s)}_{m,i,j,k} + p^{-}_{\alpha,m}\delta\psi^{\alpha,-(s-1)}_{m,i,j,k}\right) = \frac{c}{4\pi}\sum_{m'\neq m} w_{m'} \sum_{\alpha'\neq\alpha}\left(\delta\psi^{\alpha,+(s)}_{m,i,j,k} + \delta\psi^{\alpha,-(s-1)}_{m,i,j,k}\right), \tag{3.2.116a}$$

$$\delta\psi_{m,i,j}^{\alpha,+(s)} + \delta\psi_{m,i,j,k}^{\alpha,-(s-1)} = \delta\psi_{m,i,j,k}^{\alpha',+(s)} + \delta\psi_{m,i,j,k}^{\alpha',-(s-1)}\,, \quad \alpha,\alpha' = x,y,z, \quad \alpha' \neq \alpha\,, \tag{3.2.116b}$$

$$m = 1, ..., M\,,$$

where

$$p_{\alpha,m}^{\pm} = 1 \pm 6\nu_{\alpha,m} - \frac{cw_m}{4\pi}\,, \quad \alpha = x,y,z\,. \tag{3.2.117}$$

3.2.3.2 Fourier Analysis of S$_2$ Case

We now consider S$_2$ case:

$$M = 8\,, \quad w_m = \frac{\pi}{2}\,, \quad |\Omega_{m,\alpha}| = \Omega^*\,. \tag{3.2.118}$$

The equations (3.2.116) give rise to

$$\sum_{\alpha=x,y,z}\left(p_\alpha^+\delta\psi_{m,i,j,k}^{\alpha,+(s)} + p_\alpha^-\delta\psi_{m,i,j,k}^{\alpha,-(s-1)}\right) = \tilde{c}\sum_{m'\neq m}\sum_{\alpha'\neq\alpha}\left(\delta\psi_{m,i,j,k}^{\alpha,+(s)} + \delta\psi_{m,i,j,k}^{\alpha,-(s-1)}\right), \tag{3.2.119a}$$

$$\delta\psi_{m,i,j}^{\alpha,+(s)} + \delta\psi_{m,i,j,k}^{\alpha,-(s-1)} = \delta\psi_{m,i,j,k}^{\alpha',+(s)} + \delta\psi_{m,i,j,k}^{\alpha',-(s-1)}\,, \quad \alpha,\alpha' = x,y,z, \quad \alpha' \neq \alpha\,, \tag{3.2.119b}$$

$$m = 1, ..., 8\,,$$

where

$$\tilde{c} = \frac{1}{8}c\,, \tag{3.2.120}$$

$$p_\alpha^{\pm} = 1 \pm 6\nu_\alpha - \tilde{c}\,, \quad \nu_\alpha = \frac{\Omega^*}{\sigma_t\Delta\alpha}\,, \quad \alpha = x,y,z\,. \tag{3.2.121}$$

We consider a single Fourier error mode with arbitrary λ_x, λ_y, and λ_z and introduce the Fourier ansatz of the following form:

$$\delta\psi_{m,i,j,k}^{\alpha,+(s)} = \omega^s(\lambda)a_{m,\alpha}e^{\mathbf{i}\sigma_t(\lambda_\alpha\alpha_n + \lambda_{\alpha'}\alpha'_{n'} + \lambda_{\alpha''}\alpha''_{n''})}e^{\mathbf{i}0.5\sigma_t\lambda_\alpha\chi_{\alpha,m}^{+}\Delta\alpha}\,, \tag{3.2.122a}$$

$$\delta\psi_{m,i,j,k}^{\alpha,-(s)} = \omega^s(\lambda)a_{m,\alpha}e^{\mathbf{i}\sigma_t(\lambda_\alpha\alpha_n + \lambda_{\alpha'}\alpha'_{n'} + \lambda_{\alpha''}\alpha''_{n''})}e^{\mathbf{i}0.5\sigma_t\lambda_\alpha\chi_{\alpha,m}^{-}\Delta\alpha}\,, \tag{3.2.122b}$$

where

$$\text{for} \quad \alpha = x: \quad n = i\,, \quad \alpha' = y\,, \quad n' = j\,, \quad \alpha'' = z\,, \quad n'' = k\,, \tag{3.2.123a}$$

$$\text{for} \quad \alpha = y: \quad n = j\,, \quad \alpha' = x\,, \quad n' = i\,, \quad \alpha'' = z\,, \quad n'' = k\,, \tag{3.2.123b}$$

$$\text{for} \quad \alpha = z: \quad n = k\,, \quad \alpha' = x\,, \quad n' = i\,, \quad \alpha'' = y\,, \quad n'' = j\,, \tag{3.2.123c}$$

$$\chi_{x,1}^{\pm} = \pm 1\,, \quad \chi_{x,2}^{\pm} = \mp 1\,, \quad \chi_{x,3}^{\pm} = \mp 1\,, \quad \chi_{x,4}^{\pm} = \pm 1\,, \tag{3.2.124a}$$

$$\chi_{x,5}^{\pm} = \pm 1\,, \quad \chi_{x,6}^{\pm} = \mp 1\,, \quad \chi_{x,7}^{\pm} = \mp 1\,, \quad \chi_{x,8}^{\pm} = \pm 1\,, \tag{3.2.124b}$$

$$\chi_{y,1}^{\pm} = \chi_{y,2}^{\pm} = \pm 1\,, \quad \chi_{y,3}^{\pm} = \chi_{y,4}^{\pm} = \mp 1\,, \tag{3.2.124c}$$

$$\chi_{y,5}^{\pm} = \chi_{y,6}^{\pm} = \pm 1\,, \quad \chi_{y,7}^{\pm} = \chi_{y,8}^{\pm} = \mp 1\,, \tag{3.2.124d}$$

$$\chi_{z,1}^{\pm} = \chi_{z,2}^{\pm} = \chi_{z,3}^{\pm} = \chi_{z,4}^{\pm} = \pm 1\,, \tag{3.2.124e}$$

$$\chi_{z,5}^{\pm} = \chi_{z,6}^{\pm} = \chi_{z,7}^{\pm} = \chi_{z,8}^{\pm} = \mp 1\,. \tag{3.2.124f}$$

This ansatz is similar to one in 2D geometry given by Eq. (3.2.76). Introducing Eq. (3.2.122) into Eq. (3.2.119), we obtain the system of equations for the eigenvalue ω and associated eigenvector

$$\vec{a} = (a_{x,1}, ..., a_{x,8}, a_{y,1}, ..., a_{y,8}, a_{z,1}, ..., a_{z,8})^T \tag{3.2.125}$$

of the following form:

$$\sum_{\alpha=x,y,z} \left(\omega p_\alpha^+ e^{\mathbf{i}0.5\sigma_t\lambda_\alpha\chi_{\alpha,m}^+\Delta\alpha} + p_\alpha^- e^{\mathbf{i}0.5\sigma_t\lambda_\alpha\chi_{\alpha,m}^-\Delta\alpha} \right) a_{m,\alpha} =$$
$$\tilde{c} \sum_{m'\neq m} \sum_{\alpha'\neq\alpha} \left(\omega e^{\mathbf{i}0.5\sigma_t\lambda_{\alpha'}\chi_{\alpha',m'}^+\Delta\alpha'} + e^{\mathbf{i}0.5\sigma_t\lambda_{\alpha'}\chi_{\alpha',m'}^-\Delta\alpha'} \right) a_{m',\alpha'} \,, \tag{3.2.126a}$$

$$\left(\omega e^{\mathbf{i}0.5\sigma_t\lambda_\beta\chi_{\beta,m}^+\Delta\beta} + e^{\mathbf{i}0.5\sigma_t\lambda_\beta\chi_{\beta,m}^-\Delta\beta} \right) a_{m,\beta}$$
$$- \left(\omega e^{\mathbf{i}0.5\sigma_t\lambda_{\beta'}\chi_{\beta',m}^+\Delta\beta'} + e^{\mathbf{i}0.5\sigma_t\lambda_{\beta'}\chi_{\beta',m}^-\Delta\beta'} \right) a_{m,\beta'} \,, \quad \beta' \neq \beta \,, \quad \beta,\beta' = x,y,z \,, \tag{3.2.126b}$$
$$m = 1, ..., 8 \,.$$

The eigenvalue is complex in general case. The results of Fourier analysis in 3D are similar to the results in 2D geometry.

$$\rho_{3D} = \sup_{\lambda_x,\lambda_y,\lambda_z} |\omega(\lambda_x,\lambda_y,\lambda_z)| = 1 \tag{3.2.127}$$

without regard to values of scattering ratio c, total cross section σ_t and cell sizes Δx, Δy, and Δz. For example, the Fourier analysis of the spatially flat mode, i.e., $\lambda_x = \lambda_y = \lambda_z = 0$ in case σ_t=0.6, c=$\frac{1}{6}$, $\Delta x = \Delta y = \Delta z = 1$ showed that there are three different eigenvalues

- ω_1=-1 is the eigenvalue of multiplicity 16,
- ω_2=0.704736 is the eigenvalue of multiplicity 1.
- ω_3=0.704732 is the eigenvalue of multiplicity 7.

These eigenvalues were also observed in numerical calculations of the corresponding test problem. The eigenvector $\vec{a}$ associated with ω_2=0.704736 is constant and hence the corresponding Fourier mode is isotropic in angle. The slowest converging harmonic associated with ω_1 is anisotropic and alternating sign. Further analysis showed that the spectral radius ρ_{3D} equals unity for any values of total cross section σ_t, scattering ratio c, mesh size Δx, Δy, Δz.

3.2.3.3 Summary

The main result of the analysis of ITMM in 3D Cartesian geometry is that the spectral radius

$$\rho_{3D} = 1 \,.$$

It does not depend on values of c, Δx, Δy, Δz.