A Parallel Version of A Multigrid Algorithm For Isotropic Transport Equations*

T. Manteuffel§, S. McCormick§, J. Morel†, S. Oliveira‡ and G. Yang§

Abstract.
The focus of this paper is on a parallel algorithm for solving the transport equations in a slab geometry using multigrid. The spatial discretization scheme used is a finite element method called Modified Linear Discontinuous scheme (MLD). The MLD scheme represents a lumped version of the standard Linear Discontinuous scheme (LD). The parallel algorithm was implemented on the Connection Machine 2 (CM2). Convergence rates and timings for this algorithm on the CM2 and Cray-YMP are shown.

1. Introduction.
The description of the neutron transport problem is given in previous papers [2], [1], [3]. For steady state problems within the same energy group for the isotropic case (by isotropic we mean that the probability of scattering for the particles is the same for all directions), the transport equation in a slab geometry of slab width b becomes

$$\mu\frac{\partial\psi}{\partial x}+\sigma_t\psi=\frac{1}{2}\sigma_s\int_{-1}^{1}\psi(x,\mu')d\mu'+q(x,u), \qquad (1.1)$$

for x ϵ (0,b), and μ ϵ [-1,1]. Here, $\psi(x,\mu)$ represents the flux of particles at position x traveling at an angle $\theta=\arccos(\mu)$ from the x-axis, $\sigma_t\,dx$ the expected number of interactions (absorptive or scattering) that a particle will have in traveling a distance dx, $\sigma_s\,dx$ the expected number of scattering interactions, $\sigma_a=\sigma_t-\sigma_s$ the expected number of absorptive interactions, and $q(x,\mu)$ the particle source. The boundary conditions prescribing particles entering the slab are

$$\psi(0,\mu)=g_0(\mu), \qquad \psi(b,-\mu)=g_1(\mu), \qquad \mu\in(0,1). \qquad (1.2)$$

This problem is difficult for conventional methods to solve in two cases of physical interest:

1. $\gamma=\frac{\sigma_s}{\sigma_t}=1$. (pure scattering, no absorption).
2. $\frac{1}{\sigma_t}\ll$ b (optically dense).

*This work was supported by AFOSR under grant AFOSR 86-0126, the NSF under grant DMS-8704169,the DOE under grant DE-FG02-90ER25086 and Los Alamos National Laboratory.
†Computer Research Group (C-3) Los Alamos National Laboratory, Los Alamos, NM
‡Centre for Mathematics and its Applications, Australian National University, Australia 2601
§Computational Mathematics Group, University of Colorado of Denver, Denver,CO

In fact, as $\sigma_t \to \infty$ and $\gamma \to 1$, the problem becomes singularly perturbed.

Standard discrete approximations to (1.1) and (1.2) will have operators with condition numbers on the order of at least σ_t^2, regardless of the mesh size [1] . This phenomenon presents problems for numerical solution techniques in general and multigrid in particular. In this paper, the discretization used, in the spatial dimension, is a special finite element method called the Modified Linear Discontinuous scheme (MLD) (described in the next section). To solve the linear system of equations, we use a suitable relaxation process, called two-cell μ-line relaxation, within a multigrid algorithm. The serial version of this algorithm was described in [3] and the convergence properties were analyzed. It was shown that for $\gamma = 1$ (pure scaterring), which is the case we examine in this paper, the algorithm yields a convergence factor on the order of $\mathrm{O}((\frac{1}{\sigma_t h})^2)$ when $\sigma_t h \gg 1$ and $\mathrm{O}((\sigma_t h)^3)$ when $\sigma_t h \ll 1$. For the two-angle case, it behaves like an exact solver. These convergence rates were proved for special cases in [3] and are substantiated by Fourier analysis in [12]. The Fourier analysis for multigrid based on red-black μ-line relaxation (with numerical results for both Jacobi and red-black Gauss-Seidel relaxation) is shown in [12]. The results show a close agreement between the Fourier analysis and the computational results presented here. The serial algorithm for the case in which $\gamma < 1$ is examined in [4]. The multigrid algorithms were compared to a version of the Diffusion Synthetic Acceleration (DSA) method in [3] and [4]. It was shown to be faster than the DSA in all regimes.

Multigrid algorithms for the transport equations have been examined in [8] [9] [10] [11]. The algorithm here takes advantage of the rank-one form of the isotropic scattering operator in the implementation of the two-cell μ-line relaxation (see section 3). In this form the two-cell μ-line relaxation is efficient, effective and parallel. Together with efficient interpolation and restriction operators this algorithm yields the rates mentioned above (for complete details see [3]). We also mention that a parallel implementation of the DSA algorithm was examined in [13].

In this paper we describe a parallel version of the algorithm. The main benefit of the SIMD architecture of the CM2 is attained at the relaxation step of the multigrid algorithm, where we take advantage of the structure of the matrix, matching the nonzero entries of very sparse matrices with the relevant processors.

An outline of the remainder of this paper is as follows. In Section 2 we describe the discretization scheme used. In Section 3 we show how to accomplish the inversion of the relaxation matrix in order to take advantage of the SIMD architecture on the CM2. In Section 4 we describe the multigrid scheme, the interpolation and the restriction operators used. In Section 5 we discuss the storage and data structure. In Section 6 we develop the algorithm for the parallel relaxation and a few implementation details. In Section 7 we consider a few implementation details of our CM2 multigrid code. In Section 8 we report on convergence rates and compare these results with the Fourier analysis prediction. In Section 9 we show the timings obtained with our CM2 parallel codes and compare them with a sequential version of our algorithm on a cray Y-MP. Finally, in Section 10 we make a few conclusions.

2. Modified Linear Discontinuous Scheme.

The angular discretization is accomplished by expanding the angular dependence in Legendre polynomials, and is known as the S_N approximation when the first N Legendre polynomials are used. This results in a semidiscrete set of equations that resemble collocation at N Gauss quadrature points, μ_j, $j = 1, \ldots, N$, with weights w_j, $j = 1, \ldots, N$. Since the quadrature points and weights are symmetric about zero, we reformulate the problem in terms of the positive values,

μ_j, $j = 1, \ldots, n$, where $n = N/2$. We define $\psi_j^+ = \psi(x, \mu_j)$ and $\psi_j^- = \psi(x, -\mu_j)$ for $j = 1, \ldots, n$. The spatial discretization is accomplished by the MLD scheme, which uses elements that are linear across each cell and discontinuous in the upwind direction. In our grid representation, the variable $\psi_{ij}^{+(-)}$ denotes the flux of particles at position x_i in the direction μ_j $(-\mu_j)$. Assuming γ=1, the nodal equations are

$$\frac{\mu_j}{\sigma_t} \frac{\psi^+_{i+\frac{1}{2},j} - \psi^+_{i-\frac{1}{2},j}}{h_i} + \psi^+_{i,j} = \sum_{k=1}^{n} \omega_k (\psi^+_{i,k} + \psi^-_{i,k}) + q^+_{i,j}, \tag{2.1}$$

$$2\frac{\mu_j}{\sigma_t} \frac{\psi^+_{i+\frac{1}{2},j} - \psi^+_{i,j}}{h_i} + \psi^+_{i+\frac{1}{2},j} = \sum_{k=1}^{n} \omega_k (\psi^+_{i+\frac{1}{2},k} + 2\psi^-_{i,k} - \psi^-_{i-\frac{1}{2},k}) + q^+_{i,j}, \tag{2.2}$$

$$\frac{\mu_j}{\sigma_t} \frac{\psi^-_{i-\frac{1}{2},j} - \psi^-_{i+\frac{1}{2},j}}{h_i} + \psi^-_{i,j} = \sum_{k=1}^{n} \omega_k (\psi^+_{i,k} + \psi^-_{i,k}) + q^-_{i,j}, \tag{2.3}$$

and

$$2\frac{\mu_j}{\sigma_t} \frac{\psi^-_{i-\frac{1}{2},j} - \psi^-_{i,j}}{h_i} + \psi^-_{i-\frac{1}{2},j} = \sum_{k=1}^{n} \omega_k (\psi^-_{i-\frac{1}{2},k} + 2\psi^+_{i,k} - \psi^+_{i+\frac{1}{2},k}) + q^-_{i,j}, \tag{2.4}$$

$j = 1,\ldots, n$, $i = 1,\ldots, m$, with boundary conditions

$$\psi^+_{\frac{1}{2},j} = g^+_{0,j},\ \psi^-_{m+\frac{1}{2},j} = g^-_{1,j}\ . \tag{2.5}$$

$j = 1,\ldots, n$.

In our model, $x_{i+\frac{1}{2}}$ and $x_{i-\frac{1}{2}}$ are cell edges, $x_i = \frac{1}{2}(x_{i+\frac{1}{2}} + x_{i-\frac{1}{2}})$ is the cell center, and $h_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$ is the cell width, $1 \leq i \leq m$. Equations (2.1) and (2.3) are called balance equations and (2.2) and (2.4) are called edge equations. In block matrix form equations (2.1)-(2.5) can be written respectively as

$$B_i(\underline{\psi}^+_{i+\frac{1}{2}} - \underline{\psi}^+_{i-\frac{1}{2}}) + \underline{\psi}^+_i = R(\underline{\psi}^+_i + \underline{\psi}^-_i) + \underline{q}^+_i, \tag{2.6}$$

$$2B_i(\underline{\psi}^+_{i+\frac{1}{2}} - \underline{\psi}^+_i) + \underline{\psi}^+_{i+\frac{1}{2}} = R(\underline{\psi}^+_{i+\frac{1}{2}} + 2\underline{\psi}^-_i - \underline{\psi}^-_{i-\frac{1}{2}}) + \underline{q}^+_{i+\frac{1}{2}}, \tag{2.7}$$

$$B_i(\underline{\psi}^-_{i-\frac{1}{2}} - \underline{\psi}^-_{i+\frac{1}{2}}) + \underline{\psi}^-_i = R(\underline{\psi}^+_i + \underline{\psi}^-_i) + \underline{q}^-_i, \tag{2.8}$$

$$2B_i(\underline{\psi}^-_{i-\frac{1}{2}} - \underline{\psi}^-_i) + \underline{\psi}^-_{i-\frac{1}{2}} = R(\underline{\psi}^-_{i-\frac{1}{2}} + 2\underline{\psi}^+_i - \underline{\psi}^+_{i+\frac{1}{2}}) + \underline{q}^-_{i-\frac{1}{2}}, \tag{2.9}$$

$$\underline{\psi}^+_{\frac{1}{2}} = \underline{g}^+_0, \underline{\psi}^-_{m+\frac{1}{2}} = \underline{g}^-_1. \tag{2.10}$$

$i = 1,\ldots, m$. Here,

$$B_i = \begin{bmatrix} \frac{\mu_1}{\sigma_t h_i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\mu_n}{\sigma_t h_i} \end{bmatrix} \text{ and } R = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} \omega_1 & \cdots & \omega_n \end{bmatrix}, \tag{2.11}$$

where $\mu_1, \mu_2, \ldots, \mu_n$ are the positive Gauss quadrature points, $w_1, w_2, \ldots, w_n$ are the Gauss quadrature weights and $\underline{\psi}_i^{+(-)}$ is an n-vector: $\underline{\psi}_i^{+(-)} = (\psi_{i1}^{+(-)}, ..., \psi_{in}^{+(-)})^T$.

In the computational grid the inflow for positive angles is on the left of each cell and for the negative angles is on the right. For a μ-line relaxation the inflows of each cell are assumed known. This is the same as using the values of these variables from the previous iteration. Figure 1 shows the computational domain with $2m+1$ spatial points and n angles.

Consider cell i. In μ-line relaxation cell centers, $\underline{\psi}_i^+$ and $\underline{\psi}_i^-$, together with the outflow variables, $\underline{\psi}_{i-\frac{1}{2}}^-$ and $\underline{\psi}_{i+\frac{1}{2}}^+$ will be updated using the following matrix equation

$$\begin{bmatrix} I+2B_i-R & -2R & -2B_i & R \\ 0 & I-R & -R & B_i \\ B_i & -R & I-R & 0 \\ R & -2B_i & -2R & I+2B_i-R \end{bmatrix} \begin{bmatrix} \underline{\psi}_{i-\frac{1}{2}}^- \\ \underline{\psi}_i^+ \\ \underline{\psi}_i^- \\ \underline{\psi}_{i+\frac{1}{2}}^+ \end{bmatrix} = \begin{bmatrix} 0 \\ B_i\underline{\psi}_{i-\frac{1}{2}}^+ \\ B_i\underline{\psi}_{i+\frac{1}{2}}^- \\ 0 \end{bmatrix} + \begin{bmatrix} \underline{q}_{i-\frac{1}{2}}^- \\ \underline{q}_i^+ \\ \underline{q}_i^- \\ \underline{q}_{i+\frac{1}{2}}^+ \end{bmatrix}. \quad (2.12)$$

Solving this matrix equation corresponds to performing a μ-line relaxation. In our implementations we perform two-cell relaxations for the whole domain. In this kind of relaxation we consider pair of cells coupled together. This relaxation yields an error that is linear, independent of angle and continuous across two cells up to $O(\frac{1}{\sigma_t h})$ accuracy when $\sigma_t h \gg 1$ [3]. Instead of updating four variables with a relaxation scheme, we update eight variables. For example, in Figure 1 variables $\underline{\psi}_{i-\frac{1}{2}}^-$, $\underline{\psi}_i^-$, $\underline{\psi}_i^+$, $\underline{\psi}_{i+\frac{1}{2}}^-$, $\underline{\psi}_{i+\frac{1}{2}}^+$, $\underline{\psi}_{i+1}^-$, $\underline{\psi}_{i+1}^+$ and $\underline{\psi}_{i+\frac{3}{2}}^+$, will be updated. Notice that, the inflow variables $\underline{\psi}_{i-\frac{1}{2}}^+$ and $\underline{\psi}_{i+\frac{3}{2}}^-$ will be used from the previous iteration.

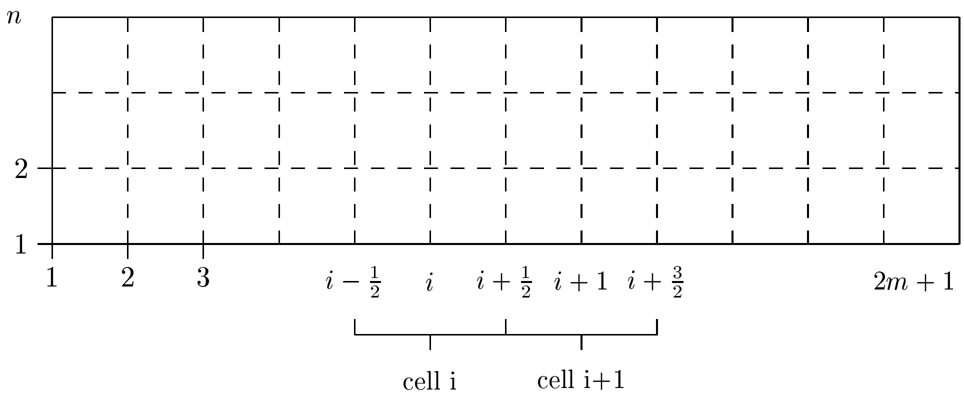

Figure 1. Computational Grid.

The two-cell μ-line relaxation involves inversion of the $8n \times 8n$ matrix

$$\begin{bmatrix} I+2B_i-R & -2R & -2B_i & R & 0 & 0 & 0 & 0 \\ 0 & I-R & -R & B_i & 0 & 0 & 0 & 0 \\ B_i & -R & I-R & 0 & -B_i & 0 & 0 & 0 \\ R & -2B_i & -2R & I+2B_i-R & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I+2B_{i+1}-R & -2R & -2B_{i+1} & R \\ 0 & 0 & 0 & -B_{i+1} & 0 & I-R & -R & B_{i+1} \\ 0 & 0 & 0 & 0 & B_{i+1} & -R & I-R & 0 \\ 0 & 0 & 0 & 0 & R & -2B_{i+1} & -2R & I+2B_{i+1}-R \end{bmatrix}. \tag{2.13}$$

The order of variables for this matrix are

$$\underline{\psi} = (\underline{\psi}^-_{i-\frac{1}{2}}, \underline{\psi}^+_i, \underline{\psi}^-_i, \underline{\psi}^+_{i+\frac{1}{2}}, \underline{\psi}^-_{i+\frac{1}{2}}, \underline{\psi}^+_{i+1}, \underline{\psi}^-_{i+1}, \underline{\psi}^+_{i+\frac{3}{2}})^T. \tag{2.14}$$

The right-hand side for the two-cell μ-line relaxation is given by

$$(0, B_i\underline{\psi}^+_{i-\frac{1}{2}}, 0, 0, 0, 0, B_{i+1}\underline{\psi}^-_{i+\frac{3}{2}}, 0)^T. \tag{2.15}$$

In our implementations we developed a multigrid scheme that uses either a block Jacobi relaxation or a block red-black Gauss-Seidel relaxation, in parallel. For the Jacobi relaxation all of the two-cell relaxations will be performed simultaneously for the whole domain. For red-black relaxation, we update half of the total number of cell-pairs during the first (red) stage of the algorithm, and the other half during second (black) stage of the algorithm, each time skipping neighboring cell-pairs. To illustrate, Figure 2 shows 4 cells. The even numbers represent the cell centers, the odd numbers the cell edges. For simplicity, we omitted the vertical lines (passing through each spatial grid point) that contain the Gauss quadrature points (angles). For a Jacobi type of relaxation process, variables at x points 5 through 9 will be updated at the same time as points 1 through 5. Each pair of cells will use the relaxation matrix just described. On a parallel machine like the CM2 red-black relaxation takes approximately twice as much CPU time as a Jacobi relaxation. This is partially offset by a better convergence rate (refer to the Fourier analysis in [12] and numerical results in Section 8).

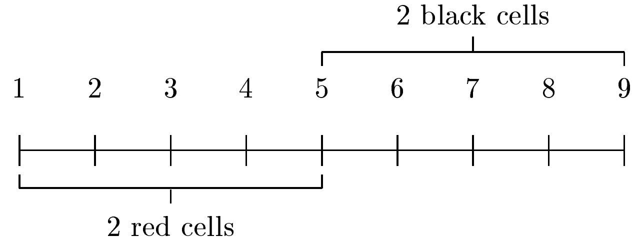

Figure 2. Red-black relaxation for 4 cells

3. Two-cell Inversion.

The inversion of the two-cell relaxation matrix can be done in a very suitable way for SIMD computers like the CM2. First notice that the matrix can be written as the difference of an easily inverted matrix, that we call A, and a rank four matrix, which we write as VW^T. The matrix A has a special structure that allows its inverse to be calculated basically in place. It is given by

$$A = \begin{bmatrix} I+2B_i & 0 & -2B_i & 0 & & & & \\ 0 & I & 0 & B_i & & & & \\ B_i & 0 & I & 0 & -B_i & & & \\ 0 & -2B_i & 0 & I+2B_i & & & & \\ & & & & I+2B_{i+1} & 0 & -2B_{i+1} & 0 \\ & & & -B_{i+1} & 0 & I & 0 & B_{i+1} \\ & & & & B_{i+1} & 0 & I & 0 \\ & & & & 0 & -2B_{i+1} & 0 & I+2B_{i+1} \end{bmatrix}_{(8n\times 8n)} \tag{3.16}$$

This is a block matrix with $n \times n$ blocks . Each nonzero block is diagonal. The rank four matrix VW^T is defined by

$$V = \begin{bmatrix} \underline{2} & \underline{0} & \underline{0} & \underline{0} \\ \underline{1} & \underline{1} & \underline{0} & \underline{0} \\ \underline{1} & \underline{1} & \underline{0} & \underline{0} \\ \underline{0} & \underline{2} & \underline{0} & \underline{0} \\ \underline{0} & \underline{0} & \underline{0} & \underline{2} \\ \underline{0} & \underline{0} & \underline{1} & \underline{1} \\ \underline{0} & \underline{0} & \underline{1} & \underline{1} \\ \underline{0} & \underline{0} & \underline{2} & \underline{0} \end{bmatrix}_{(8n\times 4)} \tag{3.17}$$

and

$$W^T = \begin{bmatrix} -\frac{1}{2}\underline{w}^T & \underline{w}^T & 0 & \frac{1}{2}\underline{w}^T & 0 & 0 & 0 & 0 \\ \frac{1}{2}\underline{w}^T & 0 & \underline{w}^T & -\frac{1}{2}\underline{w}^T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2}\underline{w}^T & \underline{w}^T & 0 & \frac{1}{2}\underline{w}^T \\ 0 & 0 & 0 & 0 & \frac{1}{2}\underline{w}^T & 0 & \underline{w}^T & -\frac{1}{2}\underline{w}^T \end{bmatrix}_{(4\times 8n)} . \tag{3.18}$$

where $\underline{1} = (1,1,\ldots,1)^T$, $\underline{0} = (0,0,\ldots,0)^T$, $\underline{2} = (2,2,\ldots,2)^T$ and $\underline{w}^T = (\omega_1, \omega_2, \ldots, \omega_n)$.

We can use the Sherman-Morrison formula to calculate the inverse of the two-cell μ-line relaxation matrix $M = A - VW^T$:

$$M^{-1} = A^{-1} + A^{-1}VE^{-1}W^TA^{-1}, \tag{3.19}$$

where $E = I - W^TA^{-1}V$. If $\underline{y}$ is an n-vector, we have $M^{-1}\underline{y} = (I + A^{-1}VEW^T)A^{-1}\underline{y}$.

The connectivity of A is two interleaved 4×4 block matrices. Thus, inverting A amounts to solving $2n$ tridiagonal matrices, each of size 4×4, which can be done in parallel. Since all matrices are diagonal, and thus commute, we make use of the notation $\frac{M_1}{M_2} = M_1M_2^{-1}$. We have $A^{-1} =$

$$
\begin{bmatrix}
\frac{I}{D_i} & 0 & \frac{2B_i}{D_i} & 0 & \frac{2B_i^2}{D_iD_{i+1}} & 0 & \frac{4B_i^2B_{i+1}}{D_iD_{i+1}} & 0 \\
0 & \frac{I+2B_i}{D_i} & 0 & \frac{-B_i}{D_i} & 0 & 0 & 0 & 0 \\
\frac{-B_i}{D_i} & 0 & \frac{I+2B_i}{D_i} & 0 & \frac{B_i+2B_i^2}{D_iD_{i+1}} & 0 & \frac{(B_i+2B_i^2)(2B_{i+1})}{D_iD_{i+1}} & \\
0 & \frac{2B_i}{D_i} & 0 & \frac{I}{D_i} & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & \frac{I}{D_{i+1}} & 0 & \frac{2B_{i+1}}{D_{i+1}} & 0 \\
0 & \frac{(B_{i+1}+2B_{i+1}^2)2B_i}{D_iD_{i+1}} & 0 & \frac{B_{i+1}+2B_{i+1}^2}{D_iD_{i+1}} & 0 & \frac{I+2B_{i+1}}{D_{i+1}} & 0 & \frac{-B_{i+1}}{D_{i+1}} \\
0 & 0 & 0 & 0 & \frac{-B_{i+1}}{D_{i+1}} & 0 & \frac{I+2B_{i+1}}{D_{i+1}} & 0 \\
0 & \frac{4B_iB_{i+1}^2}{D_iD_{i+1}} & 0 & \frac{2B_{i+1}^2}{D_iD_{i+1}} & 0 & \frac{2Bi+1}{D_{i+1}} & 0 & \frac{I}{D_{i+1}}
\end{bmatrix}
\tag{3.20}
$$

where $B_i = diag(\ldots, b_{ij}, \ldots)$, $D_i = diag(\ldots, 1 + 2b_{ij} + 2b_{ij}^2, \ldots)$ and $b_{ij} = \frac{\mu_j}{\sigma_t h_i}$. The matrix $E = I - W^T A^{-1} V$ is a 4 $\times$ 4 matrix of the form

$$
I - W^T A_0^{-1} V = \begin{bmatrix} d_1 & 0 & a_{11} & b_1 \\ 0 & d_1 & c_1 & a_1 \\ a_{22} & b_2 & d_2 & 0 \\ c_2 & a_2 & 0 & d_2 \end{bmatrix}, \tag{3.21}
$$

where

$$
d_1 = 1 - 2\sum_{j=1}^{N} \frac{\omega_j + \omega_j b_{ij}}{d_{ij}}, \tag{3.22}
$$

$$
d_2 = 1 - 2\sum_{j=1}^{N} \frac{\omega_j + \omega_j b_{i+1j}}{d_{i+1j}}, \tag{3.23}
$$

$$
a_{11} = 2\sum_{j=1}^{N} \frac{\omega_j b_{ij} b_{i+1j} + \omega_j b_{ij}^2 b_{i+1j}}{d_{ij} d_{i+1j}}, \tag{3.24}
$$

$$
b_1 = -2\sum_{j=1}^{N} \frac{\omega_j b_{ij} + \omega_j b_{ij} b_{i+1j} + \omega_j b_{ij}^2 + \omega_j b_{ij}^2 b_{i+1j}}{d_{ij} d_{i+1j}}, \tag{3.25}
$$

$$
c_1 = -2\sum_{j=1}^{N} \frac{\omega_j b_{ij}^2 b_{i+1j}}{d_{ij} d_{i+1j}}, \tag{3.26}
$$

$$
a_1 = -2\sum_{j=1}^{N} \frac{\omega_j b_{ij}^2 + \omega_j b_{ij}^2 b_{i+1j}}{d_{ij} d_{i+1j}}, \tag{3.27}
$$

$$
a_{22} = -2\sum_{j=1}^{N} \frac{\omega_j b_{i+1j}^2 + \omega_j b_{ij} b_{i+1j}^2}{d_{ij} d_{i+1j}}, \tag{3.28}
$$

$$b_2 = -2 \sum_{j=1}^{N} \frac{\omega_j b_{ij} b_{i+1j}^2}{d_{ij} d_{i+1j}}, \tag{3.29}$$

$$c_2 = -2 \sum_{j=1}^{N} \frac{\omega_j b_{i+1j} + \omega_j b_{ij} b_{i+1j} + \omega_j b_{i+1j}^2 + \omega_j b_{ij} b_{i+1j}^2}{d_{ij} d_{i+1j}}, \tag{3.30}$$

$$a_2 = -2 \sum_{j=1}^{N} \frac{\omega_j b_{ij} b_{i+1j} + \omega_j b_{ij} b_{i+1j}^2}{d_{ij} d_{i+1j}}, \tag{3.31}$$

and E^{-1} is also a 4×4 matrix and can be calculated analytically. Its entries are shown in Appendix A.

4. Multigrid.

To illustrate the multigrid scheme we consider it in a two grid level form. Let L^h denote the fine grid operator, L^{2h} the coarse grid operator, and I_{2h}^h and I_h^{2h} the interpolation and restriction operators, respectively. Let ν_1 and ν_2 be small integers (e.g., $\nu_1 = \nu_2 = 1$), which determine the number of relaxation sweeps performed before and after the coarse grid correction. Then one multigrid $V(\nu_1, \nu_2)$ cycle is represented (in two-grid form) by the following:

1. Relax ν_1 times on $L^h u^h = f^h$.
2. Calculate the residual $r^h = f^h - L^h u^h$.
3. Solve approximately $L^{2h} u^{2h} = I_h^{2h} r^h$.
4. Replace $u^h \leftarrow u^h + I_{2h}^h u^{2h}$.
5. Relax ν_2 times on $L^h u^h = f^h$.

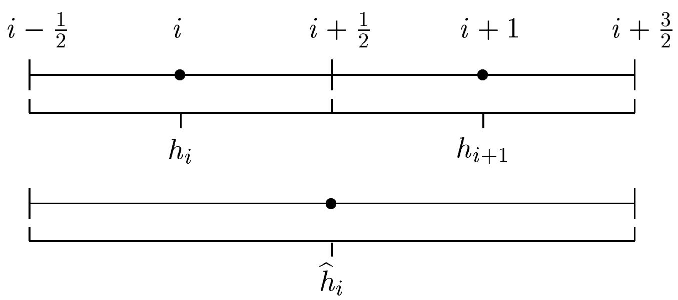

Figure 3: Fine and coarse grid.

Our multigrid scheme is applied with regard to the spatial variable only. For a multilevel scheme in angle see [7]. Figure 3 illustrates grid points on the fine grid and on the coarse grid after the a restriction operator is applied to the residual. The interpolation and restriction operators used here are based on the same finite element principle as in the derivation of the MLD scheme. They are defined as follows:

$$I_h^{2h} = \begin{bmatrix} S_{1,1} & S_{1,2} & & & & & \\ & & S_{3,1} & S_{3,2} & & & \\ & & & & \ddots & \ddots & \\ & & & & & S_{m-1,1} & S_{m-1,2} \end{bmatrix}, \qquad (4.32)$$

where

$$S_{i,1} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & \frac{h_i}{h_{i+1}+h_i}I & 0 & 0 \\ 0 & 0 & \frac{h_i}{h_{i+1}+h_i}I & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } S_{i,2} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{h_{i+1}}{h_{i+1}+h_i}I & 0 & 0 \\ 0 & 0 & \frac{h_{i+1}}{h_{i+1}+h_i}I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}$$

and

$$I_{2h}^h = \begin{bmatrix} T_{1,1} & & & \\ T_{2,1} & & & \\ & T_{1,3} & & \\ & T_{2,3} & & \\ & & \ddots & \\ & & \ddots & \\ & & & T_{1,m-1} \\ & & & T_{2,m-1} \end{bmatrix}, \qquad (4.33)$$

where

$$T_{1,i} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & \frac{2h_{i+1}+h_i}{h_{i+1}+h_i}I & 0 & \frac{-h_{i+1}}{h_{i+1}+h_i}I \\ \frac{h_{i+1}}{h_{i+1}+h_i}I & 0 & \frac{h_i}{h_{i+1}+h_i}I & 0 \\ 0 & \frac{2h_{i+1}}{h_{i+1}+h_i}I & 0 & \frac{h_i-h_{i+1}}{h_{i+1}+h_i}I \end{bmatrix}$$

and

$$T_{2,i} = \begin{bmatrix} \frac{h_{i+1}-h_i}{h_{i+1}+h_i}I & 0 & \frac{2h_i}{h_{i+1}+h_i}I & 0 \\ 0 & \frac{h_{i+1}}{h_{i+1}+h_i}I & 0 & \frac{h_i}{h_{i+1}+h_i}I \\ \frac{-h_i}{h_{i+1}+h_i}I & 0 & \frac{2h_i+h_{i+1}}{h_{i+1}+h_i}I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}.$$

The order of variables for these operators are the same as in (2.14). The coarse grid operator, L^{2h}, is defined as

$$L^{2h} = I_{2h}^h L^h I_h^{2h},$$

where L^h is given by (2.6)-(2.9). The coarse grid operator L^{2h} has the same form as L^h, but on a new grid.

We use the notation $2h$ to indicate a coarse grid, although our grids are not really assumed to be uniform. In the general nonuniform case, the mesh size at cell i on the coarse grid is given by $\widehat{h}_i = h_{2i-1} + h_{2i}$.

5. Storage and Data Structure.

Implementation of this scheme on the CM2 involves the following considerations:

- We assign one processor to each point (i.e., for each (x_i, μ_j) pair) in the computational mesh, assuming there are at least $(2m+1)n$ processors. If we are using less than $(2m+1)n$ physical processors, the CM2 will re-use the physical processors, creating virtual processors to store the additional grid points.

- Variables that are defined at different grid levels have an index that represents the grid level. We store these variables such that if they represent the same spatial and angular coordinate point they will be assigned to the same processor even if their grid levels are distinct. This avoids communication when data is accessed from different grid levels for a given spatial and angle coordinate variable. For example, variable $\underline{\psi}$ appears as ψ(:serial,:news,:news) in the code. Here the first index represents the grid level, and the remaining indices represent spatial and angular coordinates.

- In the relaxation for the nonuniform grid (variable h), some processors will require products of variables that belong to different cells, as can be seen, for example, in any row of (3.20). In particular, look at the first row of this matrix. Even though this row will have its elements stored in a processor in the first cell of the two-cell pair, it uses data from both cells (i.e. $B_i^2 B_{i+1}$). To avoid extra complication in the parallel algorithm, we store these variables in the proper processors at the outset. For example, we have an array b that stores the values of matrix B for the first cell, and an array $bp1$ that stores the values of matrix B for the second cell of the pair. This way all the processors representing grid points of a two-cell pair will contain data from both cells of the pair.

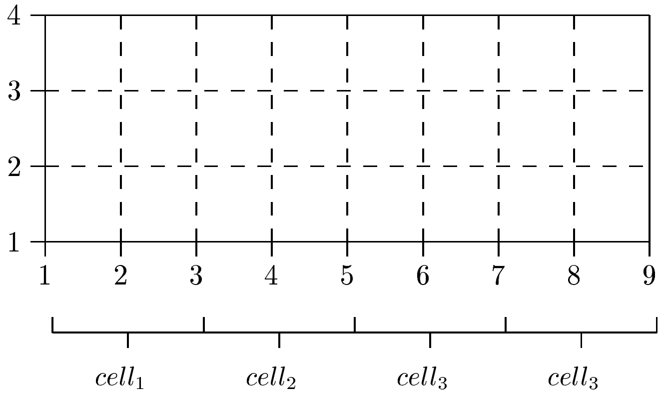

Figure 4. Computational mesh for four cells.

Figure 4 illustrates the computational mesh for four cells (horizontal axis) and four angles (vertical axis). Vertical lines 1-5 represent the variables associated with $cell_1$ and $cell_2$, i.e., $(\underline{\psi}^{+(-)}_{-\frac{1}{2}}, \underline{\psi}^{+(-)}_{1}, \underline{\psi}^{+(-)}_{\frac{3}{2}}, \underline{\psi}^{+(-)}_{2}, \underline{\psi}^{+(-)}_{\frac{5}{2}})$, while vertical lines 5-9 represent the variables associated with $cell_3$ and $cell_4$, $(\underline{\psi}^{+(-)}_{\frac{5}{2}}, \underline{\psi}^{+(-)}_{3}, \underline{\psi}^{+(-)}_{\frac{7}{2}}, \underline{\psi}^{+(-)}_{4}, \underline{\psi}^{+(-)}_{\frac{9}{2}})$. Processors represented by vertical lines 1-4 will contain all the information associated with $cell_1$ and $cell_2$ while processors represented by vertical lines 5-8 will contain all the information associated with $cell_3$ and $cell_4$. Consider $cell_1$ and $cell_2$. Array b contains the values of

$b(i,j) = \mu(j)/(\sigma_t(cell_1)h(cell_1))$. Thus $bp(1,3) = bp(2,3) = bp(3,3) = bp(4,3)$. Array $bp1$ contains the values of $bp1(i,j) = \mu(j)/(\sigma_t(cell_2)h(cell_2))$. Thus $bp1(1,3) = bp1(2,3) = bp1(3,3) = bp1(4,3)$. However $b(1,1) \neq b(1,2)$, $bp1(1,1) \neq bp1(1,2)$, and so on. The arrays b and $bp1$ are constant on the same subsets of the computational mesh. We also use some replication of data in a slightly different way, for the mesh size variable h. The elements of array h contain the cell size h_i, the array $hm1$ contains h_{i-1} and the array $hp1$ contains h_{i+1}. These three arrays are constant on vertical lines in Figure 4. This data structure is the same for all the grid levels, but with a different number of cells for each grid level.

- Some communication, of course, is necessary between processors. In fact, between the boundaries of every pair of cells we will have some shifting of data. For example, consider the variables (2.14) updated with the appropriate two-cell relaxation matrix ($cell_i$ and $cell_{i+1}$). Notice that $\underline{\psi}^+_{i+\frac{3}{2}}$ will be updated by this matrix, but the variables $\underline{\psi}^-_{i+\frac{3}{2}}$ will be updated by the next two-cell pair relaxation matrix ($cell_{i+2}$ and $cell_{i+3}$). Again consider Figure 4. Vertical line 5 represents the variables $\underline{\psi}^{(+)(-)}_{\frac{5}{2}}$. The two-cell relaxation matrix for $cell_1$ and $cell_2$ will be stored on the processors associated with vertical lines 1-4. So to update variables $\underline{\psi}^+_{\frac{5}{2}}$ the processors on line 5 will need to access data from line 4.

- In the CM2, conformable arrays (arrays with the same shape and size) are always stored in the same set of processors in the same order. To compute expressions like the elements of the matrix E in (3.21) without the need for communication we store the variables w_j (a one dimension array) as a two-dimension array that is conformable with array b (Figure 4). Consequently, w is constant over horizontal lines in Figure 4. Note that the elements of E^{-1} in (3.21) will also be conformable with variables w and b.

6. Relaxation in Parallel.

A two-cell relaxation step consists of applying M^{-1} (M is given by (2.13)) to a right-hand side vector which we call $\underline{y}$. What follows is the process to perform a two-cell relaxation step. If the current grid has k cells, then $\frac{k}{2}$ two-cell relaxations will be performed simultaneously for Jacobi, while $\frac{k}{4}$ two-cell relaxations steps will be performed in each of two stages for red-black Gauss-Seidel. Each two-cell relaxation can be performed through the following steps:

1. Form $\underline{z} = A^{-1}\underline{y}$.

   This step is done at the same time for all the cells, using the matrix A^{-1} as shown in (3.20). In order to update any variable, the associated processor needs only one row of this matrix. For example, with the order of variables used and shown in (2.14) , to update the variable $\underline{\psi}^+_i$, we need only the second row of A^{-1}, so, the processor that updates this variable needs only to store this row's entries. Note, though, that this row's entries will multiply data that are allocated in other processors. Through the use of the CM2 intrinsic function *eoshift* , we can perform a move of data in the x direction (index i), and then multiply by the appropriate entry of the matrix. To update all the variables, we thus have the following combination of shifts:

   $\psi^+_i = A_2\psi^+_i + A_1\psi^+_{i-\frac{1}{2}} + A_3\psi^+_{i+\frac{1}{2}} + A_4\psi^+_{i+1} + A_5\psi^+_{i+\frac{3}{2}}$

for j $=1,\dots,n$ (at the same time). Here, $A_1,\ A_2,\ A_3,\ A_4$ and A_5 are the elements of the appropriate row. Note that processor (i,j) does not have variables other than the ones with indexes i,j. It will therefore use the CM2 intrinsic function *eoshift* [1] in the following way:

$temp = \psi_{i+\frac{1}{2}} = eoshift(\psi_i, 1, 1)$,

$\psi_{i+1} = eoshift(temp, 1, 1)$,

and so on. Each new shift will take advantage of the previous one. The big advantage of this step is that this updating occurs at the same time for all the processors, making this matrix multiplication a job that requires only 8 communications. Note that, for each cell, the vector $\underline{z}$ has length $8n$.

2. Form $VE^{-1}W^T\underline{z}$.

   In this step we take advantage of the fact that the $8n \times 8n$ matrix $VE^{-1}W^T$ can be written as the tensor product $F \otimes R = F \otimes \underline{1}\underline{w}^t$. Remember that $\underline{1}$ and $\underline{w}^t$ are n-vectors. The 8×8 matrix F is derived from a combination of rows and columns of the matrix E^{-1} and is given by

$$F = \begin{bmatrix} 2 & 0 & & \\ 1 & 1 & & \\ 1 & 1 & & \\ 0 & 2 & & \\ & & 2 & 0 \\ & & 1 & 1 \\ & & 1 & 1 \\ & & 0 & 2 \end{bmatrix} E^{-1} \begin{bmatrix} \frac{1}{2} & 1 & 0 & -\frac{1}{2} & & & & \\ -\frac{1}{2} & 0 & 1 & \frac{1}{2} & & & & \\ & & & & \frac{1}{2} & 1 & 0 & -\frac{1}{2} \\ & & & & -\frac{1}{2} & 0 & 1 & \frac{1}{2} \end{bmatrix}. \tag{6.34}$$

Notice that if the grids are nonuniform the matrices E^{-1} and F will have their elements as arrays and will be different for each distinct pair of cells. Notice that we can write $F \otimes \underline{1}\underline{w}^t = (I_8 \otimes \underline{1})F(I_8 \otimes \underline{w}^t)$, where I_8 is the 8×8 identity matrix. Thus, multiplication by the matrix $F \otimes \underline{1}\underline{w}^t$ can thus be done in three steps:

2.1 Form $z^* = (I_8 \otimes \underline{w}^T)\underline{z}$

This step consists of a dot product of $\underline{w}^t$ and each n-vector that makes up the $8 \times n$-vector $\underline{z}$. Remember, vector $\underline{z}$ was obtained in Step 1 and $\underline{w}^t$ is an n-vector. So, first we multiply each entry of $\underline{w}^t$ by each corresponding entry of $\underline{z}$: $w_{ij} \times z_{ij}$ which is performed for all processors at the same time because $\underline{w}$ is conformable to $\underline{z}$. The i index for variable z_{ij} goes from 1 to 8 for each cell. The second stage of the dot product is to perform a summation in the angle index direction. This is done with the use of the CM2 intrinsic function *sum*. The vector $\underline{z}^*$ has length 8.

2.2 Form $\underline{\eta} =$ F $\underline{z}^*$

This step is a multiplication of a vector by a matrix, and can be performed in the CM2 similar to the way Step 1 of this relaxation was performed. Depending on which point the processor is representing, it will have stored different rows of the matrix F. Its

[1] The first argument of the eoshift function is the array name, the second is the array index that is going to be shifted, and the last is the stride of the shift.

variables will be updated through the use of the CM2 intrinsic function *eoshift* only (in this case, 16 of them).

2.3 Form $\Theta = (I_8 \otimes \underline{1})\underline{\eta}$.

This spreads the result of Step 2.1 across all the angles (for all spatial points at the same time). This is done with the use of the CM2 intrinsic function *spread*.

3. Form $A^{-1}\underline{\Theta}$ as in step 1 and add the result to $\underline{z}$.

A special remark should be made here. It is not hard to see that Steps 2.2 and 2.3 can have their order interchanged. In fact, since all the CM2 processors will execute the same instruction simultaneously, unless instructed otherwise by the use of masks, we make use of this property of the algorithm. We spread vector z^* throughout all the angles represented by different processors. Since all the processors had the appropriate row elements of matrix F from the outset, we perform Step 2.2 last. This will make the use of masks necessary only for distinction between the processors regarding the kind of spatial point it represents, not the kind of angle.

It is easy to show that, if we take advantage of the structure of the various matrices and perform the matrix multiplication steps as described, the total operation count and the number of communications per processor for one relaxation sweep are both $O(n)$, more precisely $2(n-1)$. Details are given in [12].

7. Multigrid in Parallel.

The parallel relaxation process of the multigrid algorithm was explained in Section 6. For calculation of the residual on the finer grid, again we use the CM2 intrinsic function *eoshift*. For example, to calculate the residual at an edge grid point for a negative angle, we use equation (2.4). Note that the processor that contains the variable $\psi^-_{i,j}$ will not contain other variables necessary for the calculation of the residual at this point. Therefore, we perform a move of data:

$\psi^-_{i+\frac{1}{2}} = eoshift(\psi^-_i, 1, 1)$,

$\psi^-_{i-\frac{1}{2}} = eoshift(\psi^-_i, 1, -1)$.

For calculation of the right-hand side of this equation, we perform a one-step multiplication for the conformable arrays w and ψ^+ (the same for ψ^-), and then, using the CM2 intrinsic function *sum*, we perform a summation in the angle direction for the resultant products.

After the residual is calculated, in order to solve for the error on the coarser grid, we have to estimate the residual on the coarser grid. This is done by multiplying r^h, residual on the fine grid by the restriction operator I_h^{2h} shown in (4.32). In some processors there will be no operation or communication at all, because as in Section 3 the grid level index is serial, and some processors will contain all the information they need. For example, for a negative angle at the left-hand side of a two-cell pair (negative angle outflow), the restriction consists of $r^{2h}_{i-\frac{1}{2}} = r^h_{i-\frac{1}{2}}$ (see the first row of (4.32)). This requires no communication.

For processors that represent cell centers on the coarse grid (second and third row of (4.32)), we use the CM2 intrinsic function *eoshift* in a way that is similar to what was done in Step 1 of relaxation:

$temp1 = r^h_{i+1} = eoshift(r^h_i, 1, 1)$,

$temp2 = r^h_{i-1} = eoshift(r^h_i, 1, -1)$.

We then calculate the residual on the coarse grid:

$r_i^{2h} = \frac{hm1}{(hm1+hi)} temp1 + \frac{hi}{(hi+hm1)} temp2$

After forming r^{2h}, we perform parallel relaxation again, this time to estimate the error on the coarser grid (u^{2h}).

In order to use the coarser grid quantities to correct the solution on a finer grid, we multiply u^{2h} by the matrix I_{2h}^{h} shown in (4.33). This multiplication is performed similarly to the multiplication of r^h by I_h^{2h}.

8.Numerical Results.

The CM2 codes were run for different values of $\sigma_t h$ on the fine grid. In Tables 1 and 2 we chose these values to be the worst cases predicted by the Fourier analysis shown in [12]. All of the convergence factors (i.e., the ratio of Euclidean norms of the residuals before and after a multigrid V-cycle) shown here are for the case with no absorption ($\gamma = 1$). The results are shown for the two kinds of relaxation (Jacobi and red-black Gauss-Seidel) used in our multigrid scheme. The convergence factors shown for the CM2 codes were obtained for one V-cycle after 5 V-cycles. The convergence factors were close to the ones predicted by the Fourier analysis as Tables 1 and 2 show. Tables 1 and 2 show results for uniform grids for different number of angles, e.g., S_4 means 4 scattering angles (n=2) and so on. Table 3 shows convergence factors obtained for nonuniform grids; these grids were generated randomly with h varying between 1 and 100. Table 4 shows a comparison of three relaxation methods, discussed in the next section, and the behavior of the convergence factor for varying $\sigma_t h$ for a two-cell Jacobi ($V(1,1)$ and $V(2,2)$) or two-cell red-black Gauss-Seidel($V(1,1)$) relaxation. Note that the convergence factor appears to be $\mathrm{O}((\frac{1}{\sigma_t h})^2)$ when $\sigma_t h \gg 1$ and $\mathrm{O}((\sigma_t h)^3)$ when $\sigma_t h \ll 1$ for all three relaxation methods. To make a fair comparison between the different relaxations we have to consider the difference in time spent to attain the above convergence factors. This will be analyzed in the next section.

Table 1. Worst case convergence factor for multigrid with Jacobi V(1,1) relaxation, Fourier analysis and CM2 results (for $m = 512$).

$angles$	$S4$	$S8$	$S16$	$S32$
$\sigma_t h$	.230	.100	.150	.400
Fa	.012	.012	.013	.13
$CM2$	.016	.015	.016	.017

Table 2. Worst case convergence factor for multigrid with red-black Gauss-Seidel relaxation, Fourier analysis and CM2 results ($m = 512$).

$angles$	$S4$	$S8$	$S16$	$S32$
$\sigma_t h$	.150	.240	.100	.200
Fa	.0042	.0045	.0043	.0045
$CM2$	.0063	.0065	.0086	.0045

Table 3. Convergence factors for nonuniform cells
(Jacobi and red-black Gauss-Seidel relaxation on the CM2),
$1. \leq h \leq 100.$ and $\sigma_t = 1.$

$angles$	$S4$	$S8$	$S16$
$Jacobi$	1.6×10^{-4}	1.3×10^{-4}	5.6×10^{-4}
$red-black$	1.3×10^{-4}	1.0×10^{-4}	9.7×10^{-5}

Table 4. Convergence factors for the Jacobi ($V(1,1)$ and $V(2,2)$)
and red-black Gauss-Seidel ($V(1,1)$), for various values of $\sigma_t h$.
$S4$ case and $m = 512$.

$\sigma_t h$	$Jacobi\ V(1,1)$	$red-black\ V(1,1)$	$Jacobi\ V(2,2)$
10^{-5}	9.4×10^{-10}	3.3×10^{-10}	1.47×10^{-10}
10^{-4}	8.7×10^{-7}	2.8×10^{-7}	1.38×10^{-7}
10^{-3}	4.1×10^{-4}	7.2×10^{-5}	7.40×10^{-5}
10^{-2}	1.0×10^{-2}	4.3×10^{-3}	2.41×10^{-3}
10^{-1}	1.5×10^{-2}	8.0×10^{-3}	2.71×10^{-3}
1.	4.2×10^{-3}	1.1×10^{-3}	1.03×10^{-3}
10^{1}	3.9×10^{-5}	3.2×10^{-5}	1.86×10^{-5}
10^{2}	3.4×10^{-7}	2.5×10^{-7}	4.16×10^{-7}

In table 5 we make $\sigma_t h$ equal to .1 and vary the number of cells (m). For a fixed grid mesh size the convergence factor is basically constant. The proof of convergence through Fourier analysis for uniform meshs and additional results are shown in [12]. An analytical proof of convergence is also shown in [3].

Table 5. Convergence rates for Jacobi ($V(1,1)$ and $V(2,2)$ cycle)
and red-black Gauss-Seidel (V(1,1)), on the CM2.
$S4$ case and $\sigma_t h = .1$

m	$Jacobi\ V(1,1)$	$red-black\ V(1,1)$	$Jacobi\ V(2,2)$
512	1.5×10^{-2}	9.5×10^{-3}	2.7×10^{-3}
1024	1.5×10^{-2}	9.7×10^{-3}	2.8×10^{-3}
2048	1.5×10^{-2}	9.8×10^{-3}	2.9×10^{-3}
4096	1.5×10^{-2}	9.9×10^{-3}	3.0×10^{-3}
32768	1.6×10^{-2}	9.9×10^{-3}	3.2×10^{-3}
65536	1.6×10^{-2}	9.9×10^{-3}	3.1×10^{-3}

9. Timings.

First we show the cost behavior when we keep n constant (equal to one) and vary the spatial dimension m (Figures 5 and 6 and Table 6). Second, we analyze the cost when n varies and the spatial dimension m is kept constant (Figure 7). All the results shown in this section were measured for a $V(1,1)$ or $V(2,2)$ cycle. There are two codes, one for uniform grids and the other for nonuniform grids. The uniform grid code was used for the timings in this paper. However similar results have been obtained with the nonuniform grid code, but with all of the timings roughly doubled. The sequential timings were measured using one processor of a Cray Y-MP, which has a vector architecture.

When we increase the number of grid points of the computational grid, we should have no increase on the time spent for our relaxation step on computers like the CM2 until all the processors are being used. However, in our multigrid relaxation scheme, the number of levels should be defined such that the coarsest grid consists of two cells. Hence the number of grid levels will always be $log_2\frac{m}{2}$, so, every time we double the number of cells (m) in our finest grid, we have new computations at the new added grid level since the grid level index is serial. This increase in time corresponds only to the new calculations. This can be observed in Figure 5 for m between 2^6 and 2^9 specifically, where we notice that the increase in time when we double m is linearly proportional to $\log_2 m$, while after $m = 2^9$ there is a bigger increase in the slope of the graph every time we double m because the processors are saturated.

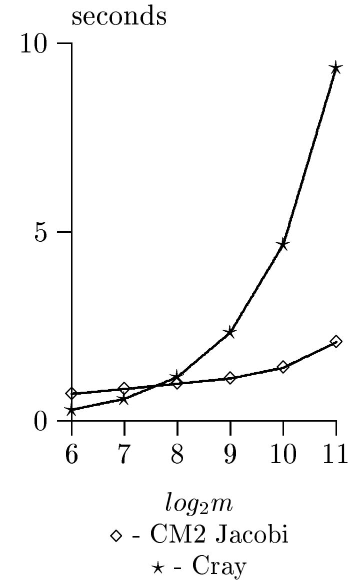

Figure 5. Timings for a $V(1,1)$ cycle
for small m and $n = 1$

For our example, when m reaches the value of 2^{10}, we have begun to saturate the CM2. Here the number of grid points $(2m+1)$, is 2049. When these timings were measured we used one sequencer, which consists of 2^{12} processors with 512 floating point units (FPU). Notice that 2048 is four times the number of FPU's, where four is the vector-length at each FPU. This explains the beginning of increase for the slopes of the piecewise linear graph in Figure 5, since we will be re-using these FPU's. Also, when we vary m between 2^{10} and 2^{11}, we are changing the number of grid points by 2048, which causes an even bigger increase on the slopes of Figure 5. In fact, the timings will double (disregarding the overhead) every time we further increase the number of cells $(2m+1)$ by multiples of 2048 (the number of FPU $\times$ 4).

If we look at the timings and dimensions for larger m, the parallel code starts increasing its time proportionally to the increase of m, similarly to the way a sequential code does. Consider the points $m = 2^{15}(32768)$ and $m = 2^{16}(65536)$ in Figure 6. This is explained by the fact that, for this range of m, every time we double m we will be re-using FPU's.

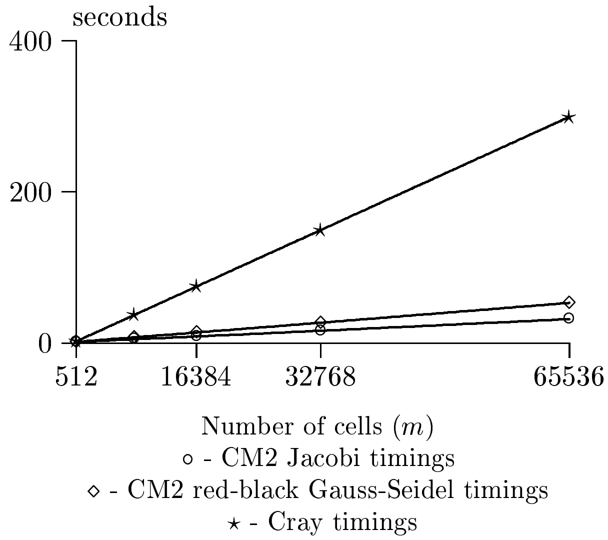

Figure 6.Timings for a $V(1,1)$
for large m and n =1

Figure 7 shows the variation of cost when n changes. The multigrid in this paper was applied to the spatial variable m, so none of the increase in cost is due to the multilevel scheme used. The increase of cost here is due to the fact of using the CM2 intrinsic functions *sum* and *spread* in the angular variable direction. This happens for example in step 2 of the parallel relaxation. Of course when we increase n to the point of re-using the FPU's, the time cost will increase similarly to the way it did when m increased.

Figure 6 also shows the additional time spent if we use a red-black Gauss-Seidel relaxation instead of a Jacobi. The timings for red-black Gauss-Seidel (V(1,1)) and Jacobi (V(1,1) and V(2,2)) two-cell relaxation are compared to the Cray timings on Table 6. The Cray timings were measured for a V(1,1) cycle. Notice that since the code on the Cray is sequential, it is irrelevant for timing purposes whether we use the Jacobi or red-black Gauss-Seidel relaxation since both relaxations will take approximately the same time, $\frac{m}{2}$ sequential steps.

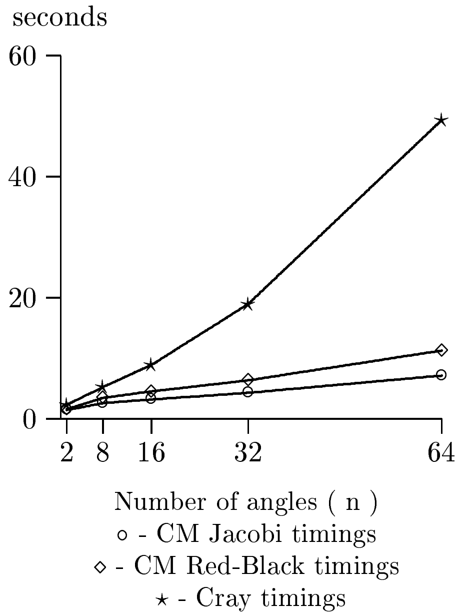

Figure 7. Timings for a $V(1,1)$ cycle varying (n) and $m = 512$.

Table 6. Timings for Jacobi (V(1,1) and V(2,2)) and red-black (V(1,1)) on the CM2 and Cray Y-MP. $S4$ case, $\sigma_t h = .1$

m	Jacobi $V(1,1)$	redblack $V(1,1)$	Jacobi $V(2,2)$	Cray
512	1.19	1.78	1.69	2.9
1024	1.93	3.00	2.82	5.85
2048	2.78	4.54	4.35	11.71
4096	4.60	7.57	7.41	23.43
32768	31.00	53.53	52.69	187.5
65536	63.14	109.54	108.6	376.0

In the CM2 for the same number of V-cycles the Jacobi relaxation in a $V(1,1)$ cycle is much faster than the red-black Gauss-Seidel relaxation in a $V(1,1)$ cycle but its convergence factor is in general worse than the red-black $V(1,1)$ convergence rate (Table 4). If we consider the Jacobi relaxation for the $V(2,2)$ cycle the timings will be slightly better than the red-black Gauss Seidel $V(1,1)$ cycle and the convergence factors in this case will be comparable. We show here that if the additional time used in doing a V-cycle with red-black Gauss-Seidel relaxation was used to perform more of the Jacobi V-cycles the convergence factor (both $V(1,1)$ and $V(2,2)$) would be in general better.

For a fixed number of V-cycles Table 7 compares Jacobi $V(1,1)$ and $V(2,2)$ with red-black Gauss-Seidel $V(1,1)$. In this table we compare the convergence factors for each relaxation using the time for a red-black $V(1,1)$ as the standard unit. For example, if red-black Gauss-Seidel

$V(1,1)$ took twice as long as Jacobi $V(1,1)$, then two Jacobi $V(1,1)$ cycles could be performed in the same time as one red-black $V(1,1)$. Using the time for red-black $V(1,1)$ as the standard unit of time, the proper convergence factor for Jacobi $V(1,1)$ would be $(\rho_{j1})^2$. We define

$$\begin{aligned} t_{rb} &= \text{time for 1 } V(1,1) \text{ cycle with red} - \text{black Gauss} - \text{Seidel relaxation.} \\ t_{j1} &= \text{time for 1 } V(1,1) \text{ cycle with Jacobi relaxation.} \\ t_{j2} &= \text{time for 1 } V(2,2) \text{ cycle with Jacobi relaxation.} \end{aligned}$$

Likewise we define the convergence factors ρ_{rb}, ρ_{j1} and ρ_{j2}. The correct comparisons are:

$$\rho_{rb} \ \ vs. \ \ (\rho_{j1})^{\frac{t_{rb}}{t_{j1}}} \ \ vs. \ \ (\rho_{j2})^{\frac{t_{rb}}{t_{j2}}}.$$

These comparisons are shown in Table 7. It is obvious that both Jacobi V(1,1) and V(2,2) are superior to red-black Gauss-Seidel. If we compare the Jacobi V(1,1) and V(2,2) relaxations we conclude that Jacobi V(1,1) gives, in general, a better convergence factor in this parallel environment for the same CPU time.

Table 7. Comparison between Jacobi V(1,1), Jacobi V(2,2), and red-black V(1,1) convergence factors (ρ_{j1}, ρ_{j2} and ρ_{rb1}) for various values of $\sigma_t h$ (S4 case).

$\sigma_t h$	$(\rho_{j1})^{\frac{t_{rb1}}{t_{j1}}}$	ρ_{rb1}	$(\rho_{j2})^{\frac{t_{rb1}}{t_{j2}}}$
10^{-5}	2.8×10^{-14}	3.3×10^{-10}	4.4×10^{-11}
10^{-4}	8.1×10^{-10}	2.8×10^{-7}	5.9×10^{-10}
10^{-3}	8.6×10^{-6}	7.2×10^{-5}	4.5×10^{-5}
10^{-2}	1.0×10^{-3}	4.3×10^{-3}	1.7×10^{-3}
10^{-1}	1.8×10^{-3}	8.0×10^{-3}	2.0×10^{-3}
1.	2.7×10^{-4}	1.1×10^{-3}	7.1×10^{-4}
10^{1}	2.4×10^{-7}	3.2×10^{-5}	1.0×10^{-5}
10^{2}	2.0×10^{-10}	2.5×10^{-7}	1.9×10^{-7}

10. Conclusions.

In this paper we have shown a parallel algorithm for a multigrid scheme for solving the transport equations in slab geometry. This algorithm is suitable for SIMD computers and takes advantage of this kind of architecture during the different stages of the multigrid scheme. It was implemented on the CM2 and was much faster than a sequential version of the same algorithm on the Cray Y-MP (using only one processor), especially when comparing large grid sizes. For the relaxation step of the multigrid algorithm we used the Sherman-Morisson formula and developed an efficient relaxation with the use of a few intrinsic functions on the CM2. The interpolation and restriction operations, for some grid points, were able to be performed without communication. The increase in the CPU time spent in a V-cycle when either one of the grid dimensions was increased was small before saturation. For the spatial dimension this increase is due to the addition of a new grid level, and for the angular dimension it is due to the use of the CM2 intrinsic functions. After saturation of the CM2 is reached, we have shown that the sharper increase in timings are caused by re-use of FPU's. In fact, when the CM2 becomes saturated the parallel timings start doubling when one of the grid dimensions is doubled. This, of course, always happens in the sequential timings for any doubling of m or n.

The convergence rates attained per V-cycle were extremely good and matched theoretical results. We implemented three different kind of relaxations: Jacobi ($V(1,1)$ and $V(2,2)$) and red-black Gauss-Seidel V(1,1) in two kinds of implementation each, uniform and nonuniform meshes. We have shown that in this context, the Jacobi V(1,1)-cycle was superior to Jacobi $V(2,2)$-cycle and red-black Gauss-Seidel $V(1,1)$-cycle. The timings for the nonuniform grid code do not appear in this paper, since they were obtained under a timesharing environment which were not reliably comparable to the uniform grid code which were executed under the dedicated environment. However similar results and conclusions seem to hold, with the exception that all of the times are roughly doubled.

The results shown in this paper show the good performance of our parallel algorithms for solving the isotropic form of the transport equation on SIMD architectures. For the anisotropic scattering we are developing an algorithm suitable for SIMD computers [12] using an angular multigrid developed in [7].

Appendix A:

The inverse of the matrix E in (3.21) is given by

$$E^{-1} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix}, \tag{1.35}$$

where

$$e_{11} = \frac{(a_{11}b_2 + b_1a_2)d_2}{det_1}, \tag{1.36}$$

$$e_{12} = \frac{(a_{11}b_2 + b_1a_2)d_2}{det_1}, \tag{1.37}$$

$$e_{13} = \frac{[d_1d_2 - (c_1b_2 + a_1a_2)](-a_1) + (a_{11}b_2 + b_1a_2)(-c_1)}{det_1}, \tag{1.38}$$

$$e_{14} = \frac{[d_1d_2 - (c_1b_2 + a_1a_2)](-b_1) + (a_11b_2 + b_1a_2)(-a_1)}{det_1}, \tag{1.39}$$

$$e_{21} = \frac{(c_1a_{22} + a_1c_2)d_2}{det_1}, \tag{1.40}$$

$$e_{22} = \frac{[d_1d_2 - (a_{11}a_{22} + b_1c_2)]d_2}{det_1}, \tag{1.41}$$

$$e_{23} = \frac{[c_1a_{22} + a_1c_2](-a_{11}) + [d_1d_2 - (a_{11}a_{22} + b_1c_2)](-c_1)}{det_1}, \tag{1.42}$$

$$e_{24} = \frac{[c_1a_{22} + a_1c_2](-b_1) + [d_1d_2 - (a_{11}a_{22} + b_1c_2)](-a_1)}{det_1}, \tag{1.43}$$

$$e_{31} = \frac{[d_1d_2 - (c_2b_1 + a_1a_2)](-a_{22}) + (a_{22}b_1 + b_2a_1)(-c_2)}{det_2}, \tag{1.44}$$

$$e_{32} = \frac{[d_1d_2 - (c_2b_1 + a_1a_2)](-b_2) + (a_{22}b_1 + b_2a_1)(-a_2)}{det_2}, \tag{1.45}$$

$$e_{33} = \frac{[d_1d_2 - (c_2b_1 + a_1a_2)](d_1)}{det_2}, \tag{1.46}$$

$$e_{34} = \frac{(a_{22}b_1 + a_1b_2)](d_1)}{det_2}, \tag{1.47}$$

$$e_{41} = \frac{(c_2a_{11} + a_2c_1)(-a_{22}) + [d_1d_2 - (a_{22}a_{11} + b_2c_1)](-c_2)}{det_2}, \tag{1.48}$$

$$e_{42} = \frac{(c_2a_{11} + a_2c_1)(-b_2) + [d_1d_2 - (a_{22}a_{11} + b_2c_1)](-a_2)}{det_2}, \tag{1.49}$$

$$e_{43} = \frac{(c_2a_{11} + a_2c_1)(d_1)}{det_2}, \tag{1.50}$$

$$e_{44} = \frac{d_1d_2 - (a_{11}a_{22} + b_2c_1)(d_1)}{det_2}, \tag{1.51}$$

$$det_1 = [d_1d_2 - (c_1b_2 + a_1a_2)][d_1d_2 - (a_{11}a_{22} + b_1c_2)] - (c_1a_{22} + a_1c_2)(a_{11}b_2 + b_1a_2), \tag{1.52}$$

and

$$det_2 = [d_1d_2 - (c_2b_1 + a_1a_2)][d_1d_2 - (a_{11}a_{22} + b_2c_1)] - (c_2a_{11} + a_2c_1)(a_{22}b_1 + b_2a_1). \tag{1.53}$$

References

[1] V. Faber, T. A. Manteuffel. *Neutron transport from the viewpoint of linear algebra*, Lecture Notes in Pure and Applied Mathematics, April 1989.

[2] E. E. Lewis and W. F. Miller. *Computational Methods of Neutron Transport*, John Wiley and Sons, New York, 1984.

[3] T. Manteuffel, S. McCormick, J. Morel, S. Oliveira and G. Yang. *A Fast Multigrid Algorithm for Isotropic Transport Problems*, submitted to SIAM J. of Sci. Stat. Comp., Oct 1992.

[4] T. Manteuffel, S. McCormick, J. Morel and G. Yang. *A Fast Multigrid Solver for Isotropic Transport Problems with Absorption*, submitted to SIAM J. of Sci. Stat. Comp., March 1993.

[5] T. Manteuffel, S. McCormick, J. Morel, and S. Oliveira. *A Parallel Multilevel Algorithm for Anisotropic Transport Equations*, to be submitted.

[6] J. E. Morel and E. W. Larson. *A New class of S_N Spatial Differencing Schemes*, Nucl. Sci. Eng., 1988.

[7] J. E. Morel and T. A. Manteuffel *An Angular Multigrid Acceleration Technique for the S_n Equations With Highly Forwarded-Peaked Scattering*, Nucl. Sci. Eng., Vol. 107, pp 330-342, 1991.

[8] A. Nowak, J. E. Morel and D.R.Harris, *A Multigrid Acceleration Method for one dimensional S_N Equations with Anisotropic Scattering*, Nucl. Sci. Eng., 102,1 1989

[9] P. F. Nowak, " A Coupled Synthetic and Multigrid Acceleration Method for Two-Dimensional Transport Calculations PhD thesis, Department of Nuclear Engineering, University of Michigan, 1988.

[10] P. F. Nowak, E. W. Larsen, *Multigrid problems for S_N problems*, Transactions of the American Nuclear Society, vol. 57, pp. 355-356, 1987.

[11] P. F. Nowak, *A Multigrid Method for S_N Calculations in x-y Geometry*, Transactions of the American Nuclear Society, vol. 56 pp. 291-292, 1988.

[12] S. Oliveira, *Parallel Multilevel Methods For Transport Equations* PhD thesis, Mathematics Department, University of Colorado at Denver, May 1993.

[13] Yavuz and E. Larsen, *S_N Methods on Parallel Computers*, Nucl. Sci. Eng., 112, **32** (1992).