Multicoloring of Grid-Structured PDE Solvers for Parallel Execution on Multiprocessor

H. C. Wang and Kai Hwang

CENG Technical Report 93-19

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4470

May 1993

Multicoloring of Grid-Structured PDE Solvers for Parallel Execution on Multiprocessors

H.C. Wang and Kai Hwang
University of Southern California
Los Angeles, CA 90089-2562

Contents

Captions of Figures

Figure 1. Storage of the sparse matrix in Eq. (1) with different formats.

Figure 2. Stencils used for discretizing 2D and 3D differential operators.

Figure 3. 2D and 3D grids resulting from finite-difference discretization. Each point is referenced by its coordinates (r, s) in a 2D grid and (r, s, t) in a 3D grid.

Figure 4. Reduction and extension operation on a 2D grid.

Figure 5. Value of residual norm (r, r) for the first 10 CGNR iterations in different modes of execution.

Figure 6. Four multicoloring schemes with different numbers of colors. Each diagram shows the ranges of grid points in the same color. The coloring in (a), (b), and (c) is valid, that in (d) is not.

Figure 7. Color assignment on a 9×8 grid using a 6-coloring scheme. Each rhombic area represents the turf of a circled point.

Figure 8. Partition of the state space into equivalence classes. The dot in each class stands for the representative mapping.

Figure 9. Residual norm versus CGNR iterations with the use of $\langle 1, 1, 1 \rangle$ coloring scheme.

Figure 10. The range and turf of a 2D grid generated by 9-point stencils.

Figure 11. The color of each grid point using a 5-coloring defined by the mapping $\langle 1, 3 \rangle$ and the ordering of grid points based on the coloring.

Figure 12. The matrix resulting from the ordering shown in Figure 11b.

Figure 13. Mflops achieved for different numbers of processors.

Figure 14. Mflops achieved for different grid sizes using 8 processors.

Figure 15. Mflops rates versus the grid size used to illustrate the effect of cache saturation on performance.

Figure 16. Mflops rates versus number of colors used.

Multicoloring of Grid-Structured PDE Solvers for Parallel Execution on Multiprocessors *

H.C. Wang and Kai Hwang
University of Southern California
Los Angeles, CA 90089-2562
{hcwang,kaihwang}@aloha.usc.edu

Abstract: In order to execute parallel PDE (partial differential equation) solvers on a multiprocessor, we have to avoid memory conflicts in accessing multidimensional data grids from the shared memory. A new multicoloring technique is proposed for sparse matrix operations which dominate the main computing cost of iterative PDE solvers. The new technique enables parallel access of grid-structured data elements in the shared memory without causing conflicts. The coloring scheme is formulated as an algebraic mapping which can be easily implemented with low overhead on commercial multiprocessors.

The proposed multicoloring scheme has been tested on an Alliant FX/80 multiprocessor for solving 2D and 3D problems using the CGNR method. Compared to the results reported by Saad on an identical Alliant system, our results show a factor of more than 30 times higher performance in Mflops. Multicoloring also leads to sparse matrices with *diagonal diagonal block* (DDB) structure, allowing parallel LU decomposition in solving PDE problems. This multicoloring technique can be extended to solve other scientific problems characterized by sparse matrices.

Index terms: Parallel processing, conjugate gradient methods, multicoloring, sparse matrix, PDE solvers, memory access conflicts, multiprocessor performance.

 Manuscript submitted to *IEEE Trans. Parallel and Distributed Systems* on May 12, 1993. This paper is also available as *Technical Report 93-19*, Department of Electrical Engineering - Systems, University of Southern California, Los Angeles, CA 90089-2562. A preliminary version of the paper will be presented in the *International Conference on Parallel Processing*, St. Charles, Illinois, August 16–20, 1993.

1. Introduction

Parallel solution of *partial differential equations* (PDE) has attracted much attention in recent years. A number of studies were reported in [5, 9, 13, 17, 15, 21, 25, 26]. In this paper, we present a new multicoloring technique which allows the parallelization of *generalized conjugate gradient* (GCG) methods on shared-memory multiprocessor systems. These methods are used to solve nonsymmetric systems of equations and are characterized by the need to perform multiplication of vectors by a matrix and its transpose. Such operations dominate the main computing cost of the PDE algorithms. The sparse matrix operations are highly desirable for parallel processing.

With most storage formats for sparse matrices that have been adopted in numerical packages for scientific computations, the matrix-vector multiplications form a duality of reduction-extension operations. It is illustrated that without coloring, the extension operation in a parallel processing environment either does not show much improvement in performance over serial execution, or causes the algorithms to diverge due to simultaneous updates to the same memory location by more than one processor. Both problems must be avoided. To cope with the problems, a multicoloring technique for grid-structured PDE problems is developed. In this approach, coloring is formulated as a mapping from the coordinates of a grid point to a certain color. Sufficient and necessary conditions for a mapping to generate a valid coloring scheme are derived. The minimum number of colors needed for different discretization stencils is also established.

Numerical experiments were conducted on a shared-memory Alliant FX/80 multiprocessor system for solving 2D and 3D PDE problems. The timing results demonstrate a significant improvement in performance. Indeed, the results obtained for extension type of operation show an appreciable gain in speed over those reported in the past. With the use of the proposed coloring scheme, these operations also exhibit better scalability with respect to machine size and problem size.

The fundamental idea of coloring is to decouple the connections among grid points. It has been used extensively to improve parallel processing efficiency. The best known coloring technique is the two-coloring developed for the parallel processing of Gauss-Seidel and SOR methods. Diagonal coloring has been used to allow a wave-front sweeping along diagonals. Coloring has also been used with irregular grid structures [19].

Another use of coloring is to form long vectors in order to reduce the startup cost of

pipeline processing on some vector processors. This class of methods is exemplified by the *continuous coloring* scheme proposed by Poole and Ortega in [22]. Related work can also be found in [1, 12, 24]. The main objective of these schemes is to obtain a matrix the diagonal blocks of which are all diagonal. Such a matrix is said to possess a DDB (diagonal diagonal blocks) structure [23].

The methods have been successfully used to vectorize the preconditioning phase of conjugate-gradient methods, in particular, preconditioning techniques based on incomplete (block) factorization of the coefficient matrix. These multicoloring techniques share some similarity with the method described in this paper. However, the objectives are different, and the outcome also differs. For instance, the continuous coloring method does not allow any point connected to a point P to have the same color as P.

This is inadequate for concurrent processing in the context of extension type of operations. In fact, we introduce the concept of the *turf* of a point P as an area within which no other point is allowed to have the same color as P, and show that the area needs to be extended beyond those points directly connected to pint P. Moreover, the number of colors used in Poole's continuous coloring scheme varies with the number of grid points along each coordinate direction and may require trial-and-error to determine a suitable coloring. Our method enumerates the valid coloring schemes, thereby eliminating the potentially tedious process.

The remainder of the paper is organized as follows. In Section 2, several storage formats for storing sparse matrix-vector multiplications are described. In Section 3, two iterative solvers for nonself-adjoint PDEs and their properties are described. In Section 4, matrix-vector multiplication operations are characterized as reduction and extension operations, depending on the storage scheme used. In Section 5, the coloring technique for a 2D grid is discussed. Four color assignment strategies and the resulting coloring are illustrated pictorially. In Section 6, the coloring of a 3D grid is formulated as a linear mapping. A rigorous analysis is provided which leads to useful rules for coloring the grid points. Based on the rules, some results presented in Section 5 are given a formal proof.

In Section 7, the other important property of matrices obtained from the proposed multicoloring technique is derived. Specifically, it is shown that such matrices have a DDB structure. In Section 8, the numerical experiments performed on an Alliant FX/80 shared-memory system are described. In Section 9, performance data obtained from the numerical experiments are analyzed and compared with previous results. Finally, we conclude with

research contributions and application requirements.

2. Storage Schemes for Sparse Matrices

The manipulation of sparse matrices is quite different from that of dense matrices. Special data structures are needed for the storage of such matrices to conserve memory space. Several commonly used formats are described below. The corresponding code for matrix-vector multiplications is then specified. These operations are required in GCG methods.

Suppose A is a sparse $N \times N$ real matrix. N is usually large, effective storage of the matrix is essential. Irrespective of the format used, the basic information required is the value of each nonzero element and its row and column indices in the matrix. Several storage schemes are compared below:

- *Diagonal format.* For matrices having a banded structure with a small bandwidth or matrices with nonzero elements clustered along fixed diagonals, the nonzero elements can be stored in an array VA of dimension $N \times Nd$. The value of Nd depends on the characteristic of the problem under consideration. A small vector IA of dimension Nd stores the offset of each element relative to the main diagonal.
- *Rowwise format.* A vector VA of size NZ (the total number of nonzero elements) is used to store the nonzero elements of the matrix row by row. A second vector JA of the same length as A stores the column index of each nonzero element. A third vector SA of size $N+1$ indicates where in vector A each row of the sparse matrix starts, with the exception that $SA(N+1)$ points to the first free memory location at the end of the matrix.
- *Columnwise format.* This is the counterpart of rowwise format. Similar to rowwise format, a vector VA holds the nonzero elements. Vector IA stores the row index of each nonzero elements, and vector SA stores the the starting location of each column in A. This format is used in Harwell-Boeing sparse matrix collection.
- *ITPACK format.* A 2D array VA of size $N \times J_{\max}$ is used to store the nonzero elements of A, where $J_{\max}$ is the largest number of nonzero elements in any row. Another array JA of the same size indicates the column index of each nonzero element. The format has been used in numerical packages ELLPACK and ITPACK.
- *Jagged diagonal format.* The format is similar to rowwise format except that the rows are sorted by the numbers of nonzero elements. The matrix is stored in stripes by

picking the first nonzero element in each row, followed by the second nonzero element in each row, etc. The data structures required are similar to rowwise format except SA now stores the starting location of each column stripe. It also needs an auxiliary vector IDX to store the correct row index of each shuffled row. The format was proposed in [19] to improve vector processing efficiency. Of course, it is possible to reorder the matrix according to the number of nonzero elements in each column.

Figure 1 shows the use of the various formats for storing the matrix

$$\begin{pmatrix} 8 & -1 & 0 & 0 & 0 \\ 1 & 10 & 11 & 0 & 0 \\ 0 & 4 & 9 & 3 & 0 \\ 0 & 0 & 7 & 5 & 6 \\ 0 & 0 & 0 & -3 & 2 \end{pmatrix} \tag{1}$$

In each case, the data structures required are indicated. Note that the jagged diagonal format results in longer stripes than the other formats.

Other schemes have also been used in the storage of sparse matrices. For instance, slightly different schemes have been used in MADPACK (Multigrid Aggregation and Disaggregation package) [8], where a columnwise storage format is used with diagonal elements preceding the other nonzero elements. Furthermore, a special format is used for restriction and prolongation operators which can often be represented as repeated patterns of fixed templates. Linked lists have also been used for the storage of large sparse matrices. Because tracking the chain of list pointers is essentially a sequential process, list structures are not amenable to concurrent operations.

3. Grid-Structured PDE Solvers

We limit the discussion to a system of linear equations arising from the discretization of a PDE-like problem defined over a regular region. For a 2D problem, the grid size is $m \times n$ with a 5-star discretization stencil; for a 3D problem, a grid of size $m \times n \times p$ is used with a 7-star stencil. See Figure 2. The discretization leads to 2D and 3D grids as shown in Figure 3, with the grid points numbered in a natural ordering and each grid point referred to by its coordinates. These problems will be referred to as the *model problems* with $N = mn$ and $N = mnp$, respectively. The resulting coefficient matrix A is sparse. Since the model problems have regular structures, most storage formats discussed in the previous section can be used.

VA:

0	8	−1
1	10	11
4	9	3
7	5	6
−3	2	0

IA:

−1	0	1

(a) Diagonal format

VA:

8	−1	0
1	10	11
4	9	3
7	5	6
−3	2	0

JA:

2	3	0
1	2	3
2	3	4
3	4	5
4	5	0

(b) ITPACK format

VA:	8	−1	1	10	11	4	9	3	7	5	6	−3	2
JA:	1	2	1	2	3	2	3	4	3	4	5	4	5
SA:	1	3	6	9	12	14							

(c) Rowwise format

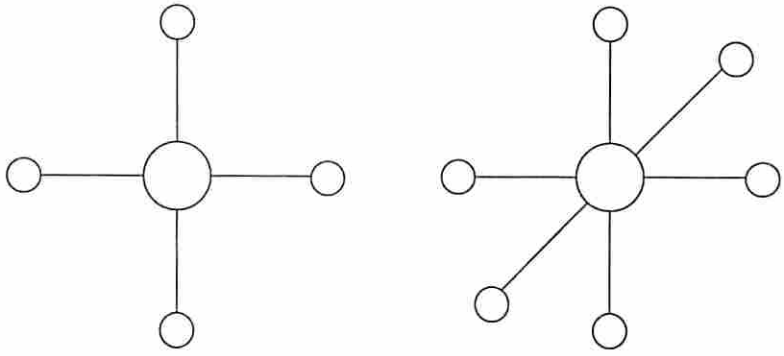
VA:	8	1	−1	10	4	11	9	7	3	5	−3	6	2
IA:	1	2	1	2	3	2	3	4	3	4	5	4	5
SA:	1	3	6	9	12	14							

(d) Columnwise format

VA:	1	4	7	8	−3	10	9	5	−1	2	11	3	6
JA:	1	2	3	1	4	2	3	4	5	2	5	3	4
SA:	1	6	11	14									
IDX:	2	3	4	1	5								

(e) Jagged diagonal format

Figure 1: Storage of the sparse matrix in Eq. (1) with different formats.



(a) 5-star stencil (b) 7-star stencil

Figure 2: Stencils used for discretizing 2D and 3D differential operators.

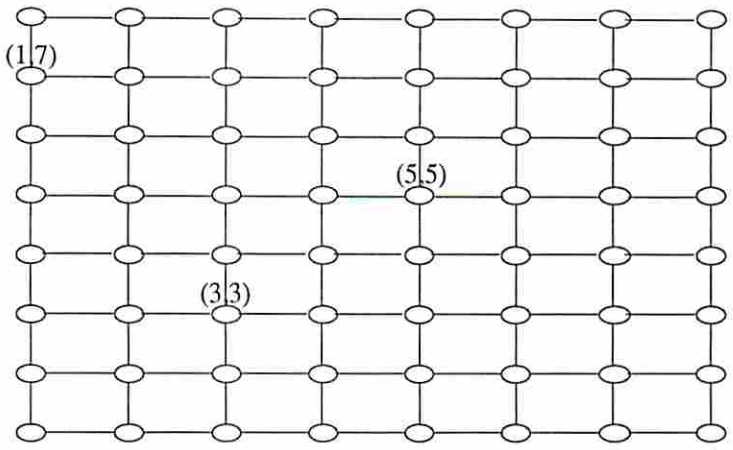
A difference between the model problems considered and typical PDE problems is that the coefficients in the stencil are randomly generated positive numbers between 0 and 1. A ramification is that the corresponding matrices are neither M-matrix [1] nor diagonally dominant. Therefore, the algorithms used tend to converge more slowly. Nevertheless, the model problems capture the essence of nonsymmetric PDE problems for which GCG methods have to be used.

Two such methods are *CG method applied to normal equations* (CGNR) and *biconjugate-gradient* (BCG) methods. These methods have been shown by examples in [20] to be suitable for solving different types of linear systems. In particular, since CGNR solves the normal equation $A^T Au = A^T f$ by conjugate gradient method, the rate of convergence is determined by the singular values of matrix A. Detailed analysis of these methods can be found in [11]. Algorithms 1 and 2 show the computation steps of CGNR and BCG methods for solving the linear system $Au = f$. One important property of CGNR method is that it converges monotonically for any nonsingular coefficient matrix. In other words, the residual norm should be reduced with each iteration.

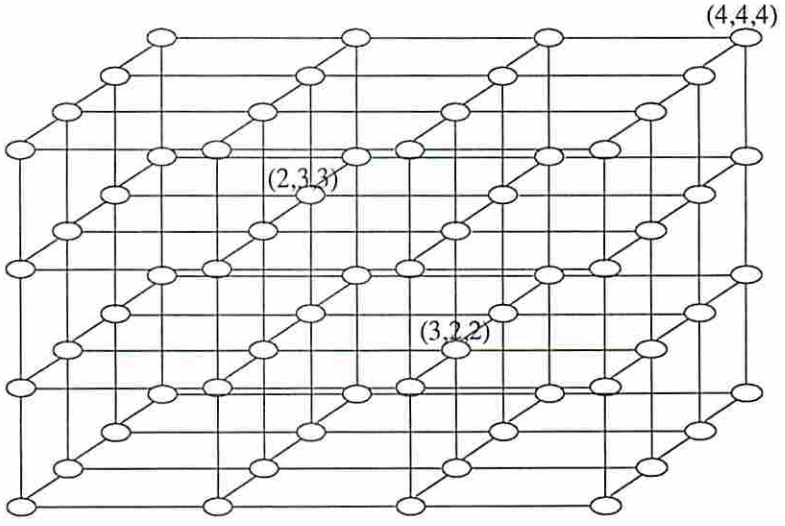
Algorithm 1 (CGNR method).

1. Initialization: Choose u_0 as an initial guess and compute $r_0 = f - Au_0$ and $p_0 = A^T r_0$.
2. **For** $k = 0, 1, 2, \ldots$ until convergence **do**

[1] A matrix is an M-matrix if all the diagonal elements are positive, all the nonzero off-diagonal elements are negative, and all the elements of its inverse are positive.



(a) An 8x8 square grid



(b) A 4x4x4 cubic grid

Figure 3: 2D and 3D grids resulting from finite-difference discretization. Each point is referenced by its coordinates (r, s) in a 2D grid and (r, s, t) in a 3D grid.

- Compute $\alpha_k = (A^T r_k, A^T r_k)/(Ap_k, Ap_k)$.
- Update solution vector $u_{k+1} = u_k + \alpha_k p_k$.
- Compute residual $r_{k+1} = r_k - \alpha_k A p_k$.
- Compute $\beta_k = (A^T r_{k+1}, A^T r_{k+1})/(A^T r_k, A^T r_k)$.
- Update direction vector $p_{k+1} = A^T r_{k+1} + \beta_k p_k$.

Algorithm 2 (Biconjugate gradient method).

1. Initialization: Choose u_0 as an initial guess and compute $r_0 = r_0^* = p_0 = p_0^* = f - Au_0$.
2. **For** $k = 0, 1, 2, \ldots$ until convergence **do**

- Compute $\alpha_k = (p_k, p_k^*)/(p_k, Ap_k^*)$.
- Update solution vector $u_{k+1} = u_k + \alpha_k p_k$.
- Compute residual vectors $r_{k+1} = r_k - \alpha_k A p_k$ and $r_{k+1}^* = r_k^* - \alpha_k A^T p_k^*$.
- Compute $\beta_k = (r_{k+1}, r_{k+1}^*)/(r_k, r_k^*)$.
- Update direction vectors $p_{k+1} = r_{k+1} + \beta_k p_k$ and $p_{k+1}^* = r_{k+1}^* + \beta_k p_k^*$

4. Reduction and Extension Operations

Both CGNR and BCG involve the multiplication of vectors by both A and A^T. It has been suggested that performing both multiplications in the same algorithm can be inefficient because of the different properties of the two operations. Suppose v is an $N \times 1$ column vector. Av is a *reduction* operation in which two vectors are combined to generate a single scalar quantity. On the contrary, operation $A^T v$ is an *extension* operation whereby the action of a scalar quantity can affect the values of several others. The concept is illustrated in Figure 4.

More accurately, the two operations form a duality. That is, the reduction and extension nature of the operations can be reversed, depending on the storage scheme used. For instance, if columnwise storage is used, the operation $A^T v$ is a reduction operation and Av is an extension operation. In contrast, Av is a reduction and $A^T v$ an extension if rowwise format is used.

The basic operation in the matrix-vector multiplication $u = Av$, where u and v are

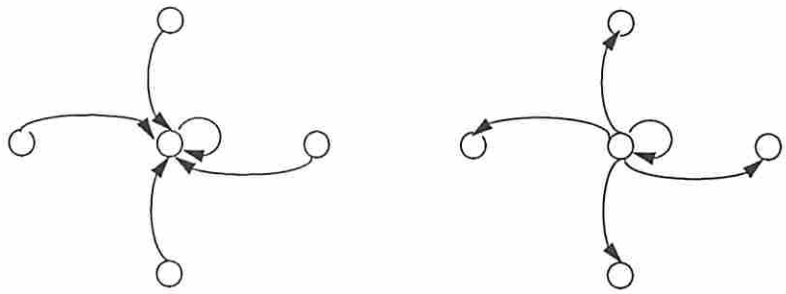


Figure 4: Reduction and extension operation on a 2D grid.

column vectors of length N, is the following:

$$u_i = \sum_{j=1}^{NZ_i} A_{ij} v_j, \tag{2}$$

where NZ_i is the number of nonzero elements in row i of A. Similarly, the basic step in $u = A^T v$ is

$$u_i = \sum_{j=1}^{NZ_i} (A^T)_{ij} v_j = \sum_{j=1}^{NZ_i} A_{ji} v_j. \tag{3}$$

It is convenient to interchange the subscripts in Eq. (3) and rewrite it as

$$u_j = \sum_{i=1}^{NZ_j} A_{ij} v_i. \tag{4}$$

In the sequel, we describe the two types of multiplications for rowwise storage format in the following pseudo code. Data structures described in Section 2 are used in the following algorithms. Computations for the other storage schemes can be similarly formulated.

- **Operation** $u = Av$:

```
For i = 1 to N do
  For j = SA(i) to SA(i+1) − 1 do
    u(i) = u(i) + VA(j) ∗ v(JA(j))
  Enddo
Enddo
```

- **Operation** $u = A^T v$:

```
For i = 1 to N do
  For j = SA(i) to SA(i+1) − 1 do
    u(JA(j)) = u(JA(j)) + VA(j) ∗ v(i)
  Enddo
Enddo
```

Consider the operation $A^T v$ for rowwise storage format. The code consists of an outer loop indexed by i and an inner loop indexed by j. At compile time, it is impossible to know the value of $JA(j)$ for a particular combination of i and j. Therefore, the code is generally not optimized for parallel execution by a compiler. However, since vector v is read-only,

Table 1: Sequential and parallel execution times (in ms) for the operation $u = A^T v$ using rowwise storage format.

Grid size	Seq.	CI	CO	6-coloring
15×15	14.8	4.5	1.2	2.3
63×63	275.6	71.8	12.0	17.2
255×255	4512.6	1183.0	207.3	292.5

the access order to its elements is irrelevant. Moreover, for the model problem it is known that each $JA(j)$ corresponds to one of the neighbors that the grid point indexed by i is adjacent to. Hence, the values of $JA(j)$ are all different for a given i. Consequently, it is permissible to carry out concurrent update operations in the inner loop. This will be referred to as *concurrent inner* (CI) mode of operation. Many systems provide compiler directives to assist the compiler in generating appropriate code for execution in this mode.

On the other hand, different combinations of i and j may lead to the same value of $JA(j)$. This is an example of *aliasing* problem in which two logical items refer to the same physical entity. For the model PDE problem, the aliasing problem can occur at adjacent grid points. Thus, if two iterations in the outer loop are allowed to proceed concurrently, they will create a *write-after-write* (WAW) hazard condition. This mode of execution is termed *concurrent outer* (CO) mode. The hazard is likely to deteriorate the performance of the GCG algorithms. In Table 1 and Figure 5, we show the execution times of the Av operation and the residual norm for the first few iterations when CGNR method is used to solve a 2D model problem.

The sequential execution time is obtained on an Alliant FX/80 detached processor running in scalar mode, and the parallel execution time is obtained on a 4-processor cluster with vector processing enabled. The curve for the sequential and CI mode shows a monotonic decrease in the residual norm as required of the CGNR method, whereas that for the CO mode shows a slower convergence at first and then diverges. The hazard condition observed for rowwise format also exists for diagonal storage format.

The hazard condition observed for rowwise format also holds for diagonal storage format. Note that it is not always possible to get rid of the hazard conditions by merely choosing a different storage scheme. As mentioned earlier, for CGNR and BCG methods, we need to perform both types of matrix-vector multiplications, which form a reduction-extension

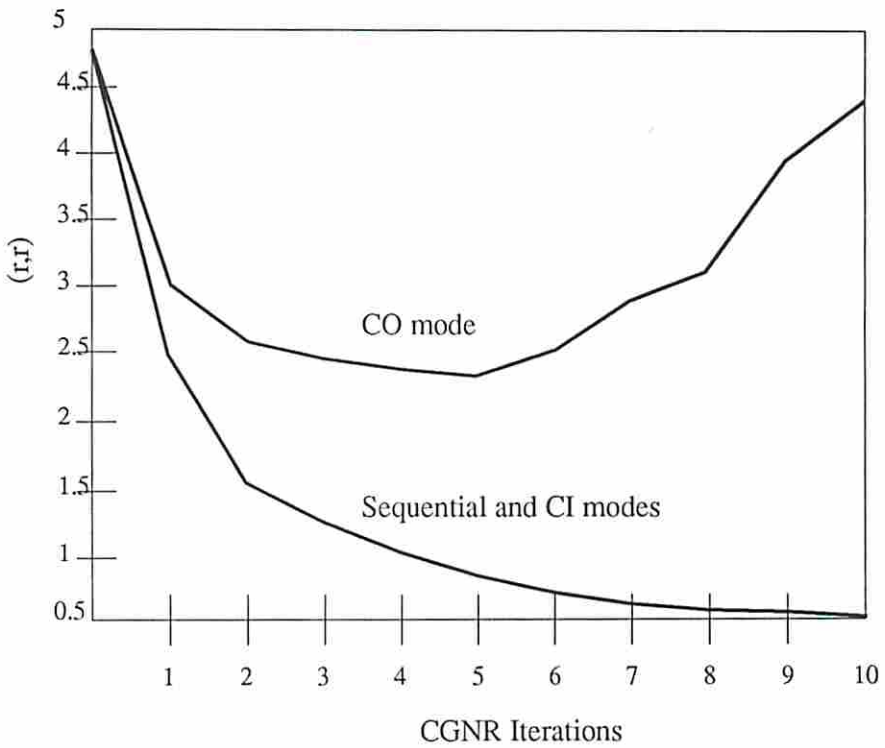


Figure 5: Value of residual norm (r, r) for the first 10 CGNR iterations in different modes of execution.

duality regardless of the storage format used.

5. Multicoloring of 2D Grids

To improve parallel performance while avoiding the hazard conditions, a coloring technique is devised. Define the *range* (or output set) R_{src} of an operation on index *src* as the set of points whose values are affected by the operation. From the perspective of the outer loop in the code for 2D extension operation, the range consists of the five grid points in the neighborhood of the point corresponding to index i. In this case, the range of a grid point coincides with the discretization stencil centered at it.

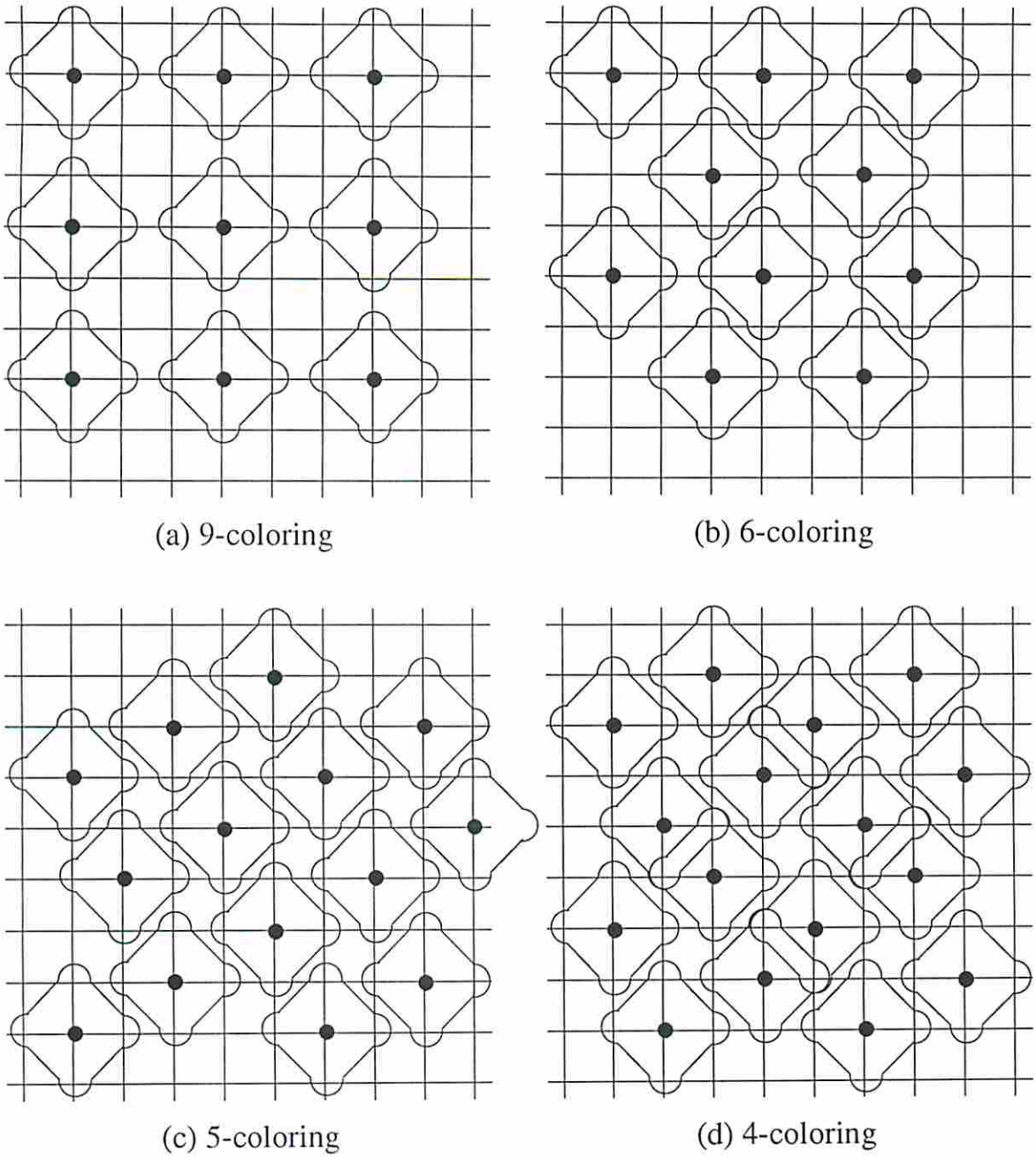
To circumvent the concurrent write conflicts, it is important to avoid simultaneous operations at grid points indexed by different i's whose ranges intersect with each other. In other words, two grid points indexed by i_1 and i_2 can be updated simultaneously only if $R_{i_1} \cap R_{i_2} = \phi$. This condition is simply a restatement of Bernstein's third condition for parallel processing [3].

Since the range of a grid point is a 5-star stencil centered at point (r, s), a two-coloring scheme does not work. Suppose two points P at (r_1, s_1) and Q at (r_2, s_2) have the same color. Then an examination of their ranges indicates that the coordinates must satisfy the condition

$$|r_1 - r_2| + |s_1 - s_2| \geq 3 \tag{5}$$

in order for P and Q to be updated concurrently. Graphically, a rhombic region surrounding point P can be drawn, which will be referred to as the *turf* of point P and denoted $\mathcal{T}_p$. Equation (5) then requires that no other point in $\mathcal{T}_p$ be allowed to have the same color as P. The turf associated with a point always contains the range of the point in an extension operation. Consequently, none of the neighbors of a grid point will have the same color as it.

Clearly, a 9-coloring scheme, in which points separated by a distance of 3 along each coordinate have the same color, satisfies Eq. (5). More compact 6-coloring and 5-coloring schemes are also allowed. With 6-coloring, points in the same color are separated by 3 along x direction and by 2 along y direction. Figure 6d shows that a 4-coloring scheme, in which points at a distance of 4 along one direction and 1 along the other are assigned the same color, leads to write interference and is not allowed. This can be clearly seen from the diagram. A proof can also be furnished by verifying that the condition embodied in Eq. (5)



(a) 9-coloring

(b) 6-coloring

(c) 5-coloring

(d) 4-coloring

Figure 6: Four multicoloring schemes with different numbers of colors. Each diagram shows the ranges of grid points in the same color. The coloring in (a), (b), and (c) is valid, that in (d) is not.

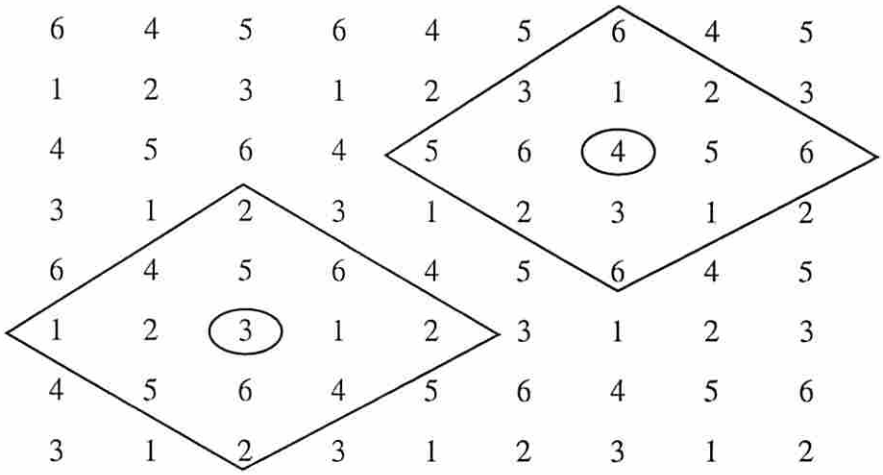


Figure 7: Color assignment on a 9×8 grid using a 6-coloring scheme. Each rhombic area represents the turf of a circled point.

is violated.

From the diagrams, it is easy to determine the set of points belonging to a certain color for each of the legitimate coloring schemes. It is also easy to decide the color of a grid point from its coordinates. For instance, in the 9-coloring scheme, the color for a point at (r, s) is specified by $3(s \bmod 3) + r \bmod 3$. In the case of 5-coloring, the same point is painted in color $(3s+r) \bmod 5$. For the 6-coloring scheme, we have the following color pattern selection rules:

$$s \bmod 4 = \begin{cases} 1 \rightarrow (1,2,3) \\ 2 \rightarrow (4,5,6) \\ 3 \rightarrow (3,1,2) \\ 0 \rightarrow (6,4,5) \end{cases} .$$

One of the patterns to the right of the arrow is chosen according to the the value of (s mod 4), and the pattern is repeated until the points on line s are exhausted. In practice, the assignment of colors can be carried out by modulus operations which can be performed in parallel. The process is illustrated in Figure 7 in which the color of each grid point on a 9×8 grid is shown. The turfs associated with two points (circled) in the grid are also indicated in the figure. In fact, if a rhombic mask similar to those shown in the figure is moved around the grid, the same observation can be made at any grid point.

Using the 6-coloring, we obtain the timing results shown in the last column of Table 1. It is interesting to compare the results with those of the CI and CO modes. As the problem

size is increased, the performance of the 6-coloring scheme improves steadily relative to that of the concurrent outer mode. This is so because the number of grid points in each color is larger and concurrent processing is more efficient. CI mode shows little improvement in speedup because of the limited amount of parallelism.

6. Multicoloring of 3D Grids

For 3D grids using 7-star discretization stencils, we expect seven colors to be sufficient to avoid the concurrent write conflicts. The geometric rendition useful for 2D grids is difficult to depict for a 3D grid. In the following, an algebraic approach to coloring is described. We use a seven color scheme as an illustration and later generalize to other numbers of colors.

We formulate a coloring scheme as a linear mapping M denoted by a 3-tuple $\langle m_1, m_2, m_3 \rangle$. M assigns color c to a grid point at (r, s, t) according to the following formula:

$$c = (m_1 r + m_2 s + m_3 t) \bmod 7, \ 1 \leq m_1, m_2, m_3 \leq 6. \tag{6}$$

The integers m_1, m_2, m_3 are the *weights* of the mapping. In general, once the number of colors is fixed, a coloring scheme is uniquely determined by the weights used in the mapping and can be represented by the 3-tuple with the weights.

Conceptually the determination of valid coloring can be envisioned as searching a state space comprising various combinations of the weights. For 7-coloring, there is a total of $6^3 = 216$ possible states. An exhaustive search for valid combinations is tedious. Since our concern is to group grid points by different colors and color assignment is invariant with regard to coordinates, the number of distinct mappings is much smaller. In fact, if the coordinate-invariance property is taken into account, the number is reduced to 56. In general, for a b-coloring scheme, a simple analysis shows that $(b-1)b(b+1)/6$ distinct combinations are obtained after considering the invariance property. For large b, exhaustive search is still time-consuming. Therefore, it is desirable to determine valid mappings systematically. To this end, we first introduce the concept of equivalent mappings.

Definition 1. *Two mappings are equivalent if two grid points assigned the same color c_1 by one mapping are assigned the same color c_2 by the other mapping.*

Formally, mappings M_1 and M_2 are equivalent if for any $i \neq j$, $M_1(r_i, s_i, t_i) = c_1$, $M_1(r_j, s_j, t_j) = c_1$, and $M_2(r_i, s_i, t_i) = c_2$, then $M_2(r_j, s_j, t_j) = c_2$.

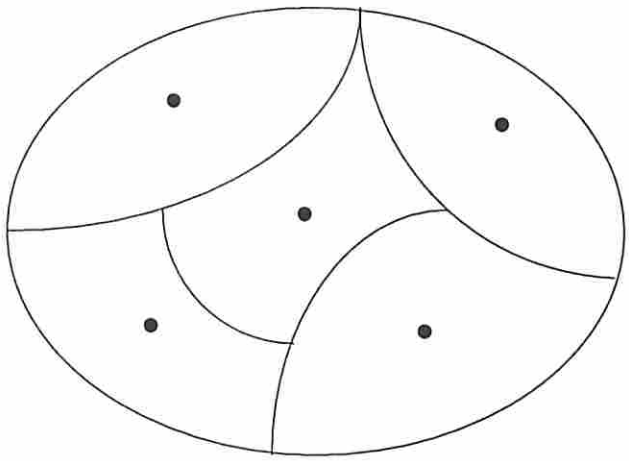


Figure 8: Partition of the state space into equivalence classes. The dot in each class stands for the representative mapping.

Equivalence relation facilitates the partitioning of the state space into equivalence classes. Each class can be represented by a single state to be determined in the follows. The concept is illustrated in Figure 8. Equivalent mappings allow us to search for the representative states instead of inspecting all the individual states, thereby reducing the amount of time required.

Based on the above definition, we can derive the following lemma.

Lemma 1. *Suppose mapping M_1 is defined by the 3-tuple $\langle m_1, m_2, m_3 \rangle$ and M_2 by $\langle Sm_1, Sm_2, Sm_3 \rangle$ with the scaling factor S an integer, $1 \leq S \leq 6$. Then the mappings M_1 and M_2 are equivalent.*

Proof: Suppose $M_1(r_1, s_1, t_1) = c_1$, $M_1(r_2, s_2, t_2) = c_1$, and $M_3(r_1, s_1, t_1) = c_2$. That is,

$$(m_1 r_1 + m_2 s_1 + m_3 t_1) \bmod 7 = c_1, \tag{7}$$

$$(m_1 r_2 + m_2 s_2 + m_3 t_2) \bmod 7 = c_1, \tag{8}$$

$$S(m_1 r_2 + m_2 s_2 + m_3 t_2) \bmod 7 = c_2. \tag{9}$$

From Eq. (7) and (8), we have

$$[(m_1 r_1 + m_2 s_1 + m_3 t_1) - (m_1 r_2 + m_2 s_2 + m_3 t_2)] \bmod 7 = 0. \tag{10}$$

Hence,

$$S[(m_1 r_1 + m_2 s_1 + m_3 t_1) - (m_1 r_2 + m_2 s_2 + m_3 t_2)] \bmod 7 = 0. \tag{11}$$

Therefore,

$$S(m_1r_1 + m_2s_1 + m_3t_1) \bmod 7 = S(m_1r_2 + m_2s_2 + m_3t_2) \bmod 7, \tag{12}$$

which establishes the equivalence relation. □

In fact, it can be shown that the set $O = \{0, 1, 2, 3, 4, 5, 6\}$ and the associated operations $+_7$ and $\times_7$ defined by $+_7(a, b) = (a + b) \bmod 7$, and $\times_7(a, b) = (a \times b) \bmod 7$, respectively, for $a, b \in O$, forms a group [7]. Through this connection, the lemma can also be proved using group theory.

Theorem 1. *Suppose mapping M_1 is defined by the 3-tuple $\langle m_1, m_2, m_3\rangle$ and M_2 by $\langle Sm_1 \bmod 7, Sm_2 \bmod 7, Sm_3 \bmod 7\rangle$ with S an integer, $1 \leq S \leq 6$. Then mappings M_1 and M_2 are equivalent.*

Proof: First observe the relation

$$Sm_1r_2 \bmod 7 = (Sm_1 \bmod 7)r_2 \bmod 7,$$

which follows from the fact $Sm_1 = \lfloor Sm_1/7\rfloor \times 7 + (Sm_1) \bmod 7$. Similarly,

$$Sm_2s_2 \bmod 7 = (Sm_2 \bmod 7)s_2 \bmod 7,$$

and

$$Sm_3t_2 \bmod 7 = (Sm_3 \bmod 7)t_2 \bmod 7.$$

Hence,

$$\begin{aligned} & [(Sm_1 \bmod 7)r_2 + (Sm_2 \bmod 7)s_2 + (Sm_3 \bmod 7)t_2] \bmod 7 \\ = \ & (Sm_1r_2 + Sm_2s_2 + Sm_3t_2) \bmod 7 \\ = \ & c_2 \qquad \text{by Lemma 1.} \end{aligned}$$

□

An immediate result of the theorem is stated in the following corollary.

Corollary 1. *In 7-coloring, all the mappings $\langle Sm_1, Sm_2, Sm_3\rangle$ for $S = 1, 2, 3, 4, 5, 6$ are equivalent.*

For instance, mappings $\langle 1,1,2\rangle$, $\langle 2,2,4\rangle$, $\langle 3,3,6\rangle$, $\langle 4,4,1\rangle$, $\langle 5,5,3\rangle$, and $\langle 6,6,5\rangle$ are all equivalent, and we say $\langle 1,1,2\rangle$ *covers* the other equivalent mappings. By the equivalence theorem, the number of unique mappings can be cut down to 22. Compared to the original 216 possible combinations, the reduction in complexity is significant, which makes the search for legitimate coloring schemes much less expensive. The set comprising the unique mappings is called the *minimum cover set.*

In the 3-tuple notation, the minimum cover set for 7-coloring consists of the following mappings: $\langle 1,1,1\rangle$, $\langle 1,1,1\rangle$, $\langle 1,1,2\rangle$, $\langle 1,1,3\rangle$, $\langle 1,1,4\rangle$, $\langle 1,1,5\rangle$, $\langle 1,1,6\rangle$, $\langle 1,2,3\rangle$, $\langle 1,2,4\rangle$, $\langle 1,2,5\rangle$, $\langle 1,2,6\rangle$, $\langle 1,3,4\rangle$, $\langle 1,3,5\rangle$, $\langle 1,3,6\rangle$, $\langle 1,4,5\rangle$, $\langle 1,4,6\rangle$, $\langle 1,5,6\rangle$, $\langle 2,3,4\rangle$, $\langle 2,3,6\rangle$, $\langle 2,5,6\rangle$, $\langle 3,4,5\rangle$, $\langle 3,4,6\rangle$, and $\langle 3,5,6\rangle$.

However, not all these 22 mappings will lead to colorings which prevent concurrent write conflicts. For example, Figure 9 shows the values of residual norm as a function of CGNR iterations applied to a 3D grid when the mapping $\langle 1,1,1\rangle$ is used to color the grid points. Four processors are used in the execution. Obviously, the mapping is not valid for parallel processing.

Thus, the final step in the coloring process is to determine valid mappings in the minimum cover set by eliminating invalid mappings from the set. For this purpose, we resort to a combination of geometric and algebraic approaches. Similar to 2D cases, the turf of a grid point P at (r,s,t) is a rhomboid with a side length of 3 surrounding P; i.e., it is defined by the following inequality:

$$\mathcal{T}_p = \{Q : |\triangle r| + |\triangle s| + |\triangle t| \leq 2\}, \tag{13}$$

where $\triangle$ stands for the distance along each direction between point P and another point Q in the 3D grid. Any other grid point in $\mathcal{T}_p$ is not allowed to have the same color as P.

From the definition of color mapping, the distance between two grid points in the same color satisfies the condition

$$(m_1 \triangle r + m_2 \triangle s + m_3 \triangle t) \bmod 7 = 0. \tag{14}$$

Hence, the goal is to find values for m_1, m_2, m_3 such that any point $Q \in \mathcal{T}_p$, $Q \neq P$ will have a different color than P. From Eq. (13), a point in $\mathcal{T}_p$ other than P satisfies one of the following conditions:

- One of $|\triangle r|$, $|\triangle s|$, $|\triangle t|$ is 2, the others are 0;

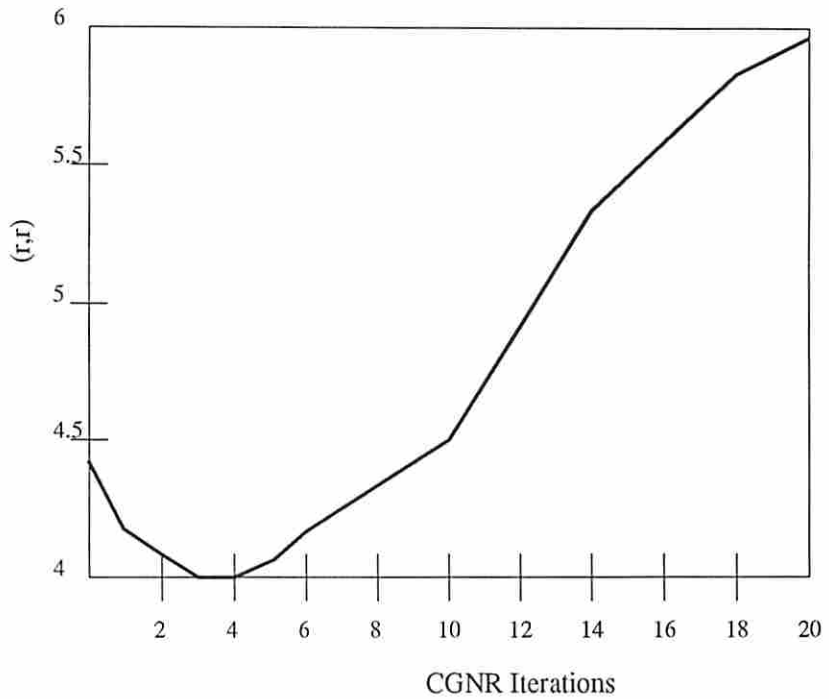


Figure 9: Residual norm versus CGNR iterations in parallel execution with the use of $\langle 1, 1, 1 \rangle$ 7-coloring scheme.

- One of $|\triangle r|$, $|\triangle s|$, $|\triangle t|$ is 1, the others are 0;
- Two of $|\triangle r|$, $|\triangle s|$, $|\triangle t|$ are 1, the third is 0.

In the first two cases, since $1 \leq m_1, m_2, m_3 \leq 6$, the condition specified by Eq. (14) is always violated, hence Q and P will always have different colors. In the last case, Eq. (14) is satisfied only if

$$(m_i \pm m_j) \bmod 7 = 0 \text{ for } i \neq j. \tag{15}$$

Because $1 \leq m_i \leq 6$, the condition Eq. (15) holds if

$$m_i = m_j \text{ or } m_i + m_j = 7 \text{ for } i \neq j. \tag{16}$$

A direct consequence of Eq. (16) is that any coloring scheme with two or more identical weights will give rise to WAW conflicts and should be eliminated from the minimum cover set. Likewise, mappings with two weights that add up to 7 are also prohibited. Based on these criteria, only five coloring schemes, $\langle 1,2,3\rangle$, $\langle 1,2,4\rangle$, $\langle 1,3,5\rangle$, $\langle 2,3,6\rangle$, and $\langle 3,5,6\rangle$, and their equivalent mappings will produce valid 7-coloring.

Although the foregoing discussion has been focused on 7-coloring applied to a 3D grid, more general rules can be formulated for a b-coloring scheme when a 5-star or 7-star stencil is used on a 2D or 3D grid. Let $\langle m_1, ..., m_d\rangle$ be the coloring scheme where $d = 2$ or 3. We have the following rules on the selection of the weights:

1. $0 \leq m_i \leq b$ for all i.
2. $2m_i \neq b$ for all i.
3. $m_i + m_j \neq b$, $1 \leq i, j \leq d$.
4. $m_i \neq m_j$, $1 \leq i, j \leq d$.

Based on the rules, it is straightforward to determine valid choices for the weights. Moreover, using these rules, it can be shown that both 4-coloring for 2D grids and 6-coloring for 3D grids are not valid for avoiding WAW hazards. For instance, if 6-coloring is applied to a 3D grid, the possible choices for any of the three weights are 1 through 5. Value 3 is disallowed since it violates rule 2. The pair 1 and 5 cannot appear together since rule 3 will be violated, neither can 2 and 4. Also, rule 4 disallows any number to be chosen more than once. Thus, it is impossible to form any legitimate 3-tuple, and hence 6-coloring

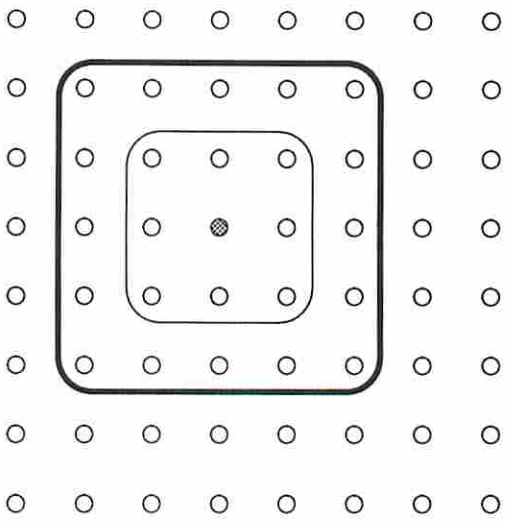


Figure 10: The range and turf of a 2D grid generated by 9-point stencils.

is not feasible. Consequently, 5 and 7 are the minimum numbers of colors needed for 2D and 3D grids, respectively. The results are consistent with intuitive reasoning. Aside from theoretical interest, larger values of b may have practical applications. For example, in some memory interleaving schemes, b can be chosen to improve memory-access efficiency.

The above rules reflect the similarity in the way turfs for 5-and 7-star stencils are constructed. For different stencils, such as the 9-point stencil used for 2D PDEs with mixed differential terms, these rules will need to be modified. Figure 10 shows the range and turf of a grid point on a 2D grid when a 9-point stencil is used.

7. Multicoloring to Generate DDB Matrices

Because of the periodicity property of the modulus operation used in the multicoloring, the grid points are uniformly distributed among the colors for a regular grid with a sufficiently large number of grid points. For instance, if b colors are used on a 2D grid with $m \times n$ grid points, the number of grid points in each color will be $\lceil mn/b \rceil$ for (mn mod b) of the colors and $\lfloor mn/b \rfloor$ for the rest of the colors.

Furthermore, matrices resulting from the use of multicoloring have a *diagonal diagonal block* (DDB) structure [23]. The grid points are numbered according to the colors. We number the grid points in color 1 first, followed by all the grid points in color 2, and so on. Within each color, the grid points are numbered by the natural ordering. This

4	2	5	3	1	4	2	5
5	3	1	4	2	5	3	1
1	4	2	5	3	1	4	2
2	5	3	1	4	2	5	3
3	1	4	2	5	3	1	4
4	2	5	3	1	4	2	5
5	3	1	4	2	5	3	1
1	4	2	5	3	1	4	2

(a) Coloring of grid points

39	14	52	27	1	40	15	53
54	28	2	41	16	55	29	3
4	42	17	56	30	5	43	18
19	57	31	6	44	20	58	32
33	7	45	21	59	34	8	46
47	22	60	35	9	48	23	61
62	36	10	49	24	63	37	11
12	50	25	64	38	13	51	26

(b) Ordering of grid points based on the coloring

Figure 11: The color of each grid point using a 5-coloring defined by the mapping $\langle 1, 3 \rangle$ and the ordering of grid points based on the coloring.

numbering scheme assigns to each grid point a unique number. As a result, the grid points are partitioned into disjoint sets denoted by G_c for $1 \leq c \leq b$.

Consider a 2D grid. We denote the color of a grid point at (r, s) by $C(r, s)$ and its order in the coloring scheme by $ord(r, s)$. Figure 11a shows the color of each grid point in the 5-coloring of an 8×8 grid by the mapping $\langle 1, 3 \rangle$, i.e.,

$$C(r, s) = (r + 3s) \bmod 5.$$

Note that there are 13 grid points in colors 1, 2, 4, and 5 each, and 12 points in color 3. The fact reflects the uniform distribution property. Figure 11b shows the ordering of the grid points corresponding to the coloring.

The DDB property essentially states that each diagonal block corresponding to a group G_c of grid points should be diagonal. A criterion established in [22] (Theorem 3.2) can be used to test whether a matrix satisfies the DDB property. The criterion can be paraphrased in the following theorem:

Theorem 2. *A matrix A satisfies the DDB property if and only if for any grid point $P \in G_c$, none of its neighbors in the stencil belongs to the same set G_c.*

Proof: Consider a diagonal block submatrix A_c corresponding to the grid points in G_c for any c in the range of 1 through b. If there is a grid point of which at least one neighbor

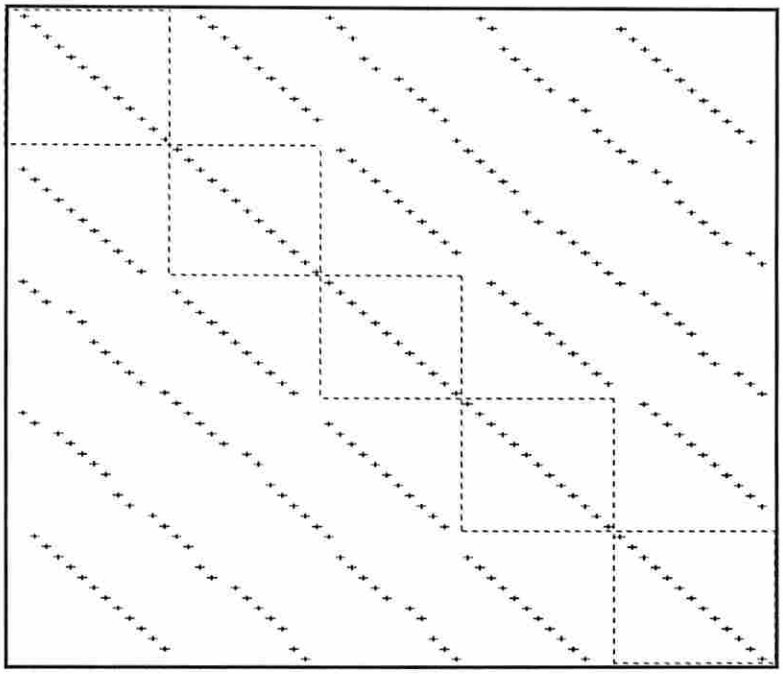


Figure 12: The matrix resulting from the ordering shown in Figure 11b.

also belongs to G_c, then there will be a nonzero off-diagonal element in A_c on the row corresponding to the grid point. Hence, A_c is not diagonal. Therefore, if A is DDB, then the condition must hold.

Conversely, if the condition holds, then there will be no off-diagonal nonzero element in any of the diagonal block matrices A_c for any color c. Therefore, each of the diagonal block matrices is diagonal. Thus, A has a DDB structure. A similar proof can be found in [22]. □

As we have mentioned in Sections 5 and 6, for 2D and 3D grids, the neighbors of a grid point are all in different colors than it. (Actually, our multicoloring scheme satisfies an even stronger condition.) Therefore, the DDB property is satisfied with the use of the multicoloring technique.

In Figure 12, we show the coefficient matrix corresponding to the grid in Figure 11. Each plus sign "+" in the diagram represents a nonzero element in the matrix. Clearly, the matrix satisfies the DDB property. In fact, as mentioned in [23], besides the implied meaning of using more than two colors, multicoloring has been used to represent the class of general coloring schemes which give rise to matrices with DDB structure. In light of this, our use of the term is consistent with the convention.

The DDB structure has an important application in the parallel processing of preconditioned conjugate gradient algorithms. In particular, it is very useful in the parallel implementation of preconditioners based on incomplete LU decomposition of the coefficient matrix [6, 10, 18]. The idea is to use a lower triangular matrix L and an upper triangular matrix U, both of which have the same sparsity structure as the corresponding parts of the coefficient matrix A. Since A has a DDB structure, both L and U also have DDB structure. This eliminates a major bottleneck in the forward and backward sweeps during the solution of the preconditioner system, making the sweeps more parallelizable. For a more detailed discussion, see [23]. Experimental results were also reported in [22].

In summary, we have proved the DDB property of matrices obtained using multicoloring and link it to another important application in the parallel computation of the preconditioning step. In the use of conjugate gradient methods, preconditioning is an integral part of these algorithms. In fact, preconditioning can be used to significantly improve the convergence rate of such algorithms [4]. Thus, in addition to our original goal of resolving the concurrent write conflicts in extension type of operations, our multicoloring technique can also be used to improve the parallel performance of the preconditioning operation.

8. Multiprocessor Experiments

Numerical experiments have been conducted on an Alliant FX/80 with multiple vector processors. The machine has a memory hierarchy consisting of cache and main memory, both of which are shared among processors. As in typical memory hierarchy, the cache serves a small-capacity, fast-access buffer storage. The system also provides compiler directives and other support to facilitate execution in different modes [2].

The problems tested are the model problems as described in Section 3. Only the performance data for matrix-vector multiplications are collected and expressed in Mflops (millions of floating-point operations per second) in the ensuing discussions. 5-star and 7-star stencils are used for 2D and 3D grids, respectively.

The total number of nonzero elements in the coefficient matrix obtained from discretization is $5mn-2(m+n)$ on an $m \times n$ 2D grid and $7mnp-2(mn+mp+np)$ on an $m \times n \times p$ 3D grid. For each of these elements, two arithmetic operations, one multiplication and one addition, are performed. Therefore, the number of floating-point operations is $10mn - 4(m+n)$ and $14mnp - 4(mn + mp + np)$ for 2D and 3D model problems. This information together

with the execution time allows us to estimate the Mflops rate.

Processing at grid points proceeds by colors. In this manner, the colors also define synchronization points for the processing. In other words, all grid points in one color must have been processed before grid points in the next color are processed. This allows us to define the *degree of parallelism* (DOP) as the number of grid points that can be operated upon simultaneously without causing instability to the algorithms used. For the extension operation, the DOP is equal to the number of grid points divided by b when a b-coloring scheme is used. DOP represents the software or algorithmic parallelism. When the algorithm is executed on an Alliant FX/80, the software parallelism is split to match hardware parallelism which is a combination of vector processing within each processor and concurrent processing across multiple processors. Color assignment is performed only once at the beginning with the information stored for subsequent use.

9. Performance Results and Analysis

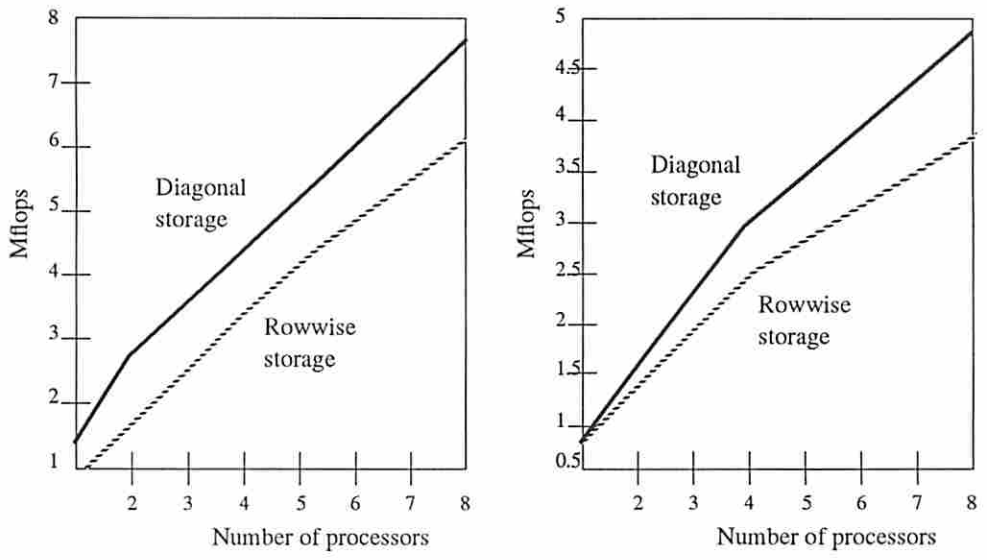
In the following, we investigate several factors that may affect the performance of matrix-vector multiplication operations, such as problem size and machine size. In general, we expect the performance to improve with an increasing number of grid points, since a higher degree of parallelism is achieved which can better exploit the computing resources. Likewise, when hardware parallelism is increased In general, with the use of more processors, performance should improve. But certain hardware constraints may limit the gain derived from grid size increase. Comparison with previous results is presented where appropriate.

A. Measured Mflops Performance

In Figure 13 we show the measured Mflops versus the number of processors for grid sizes of 63×63 and $31 \times 31 \times 31$, respectively. Vector processing is enabled in all cases. In each figure, solid curves represent results for diagonal storage format and dashed curves correspond to those for rowwise format.

Four different machine sizes are compared with the number of processors equal to 1, 2, 4, and 8, respectively. It is clear that with the increase in hardware parallelism, the Mflops rate increases monotonically. It also indicates that coloring has generated enough parallelism to utilize the available computing resources.

Figure 14 shows the Mflops rate obtained in performing the extension operation using 8 processors for different grid sizes. As before, vector processing is enabled. When the grid



(a) A 63 x 63 2D grid

(b) A 31 x31 x 31 3D grid

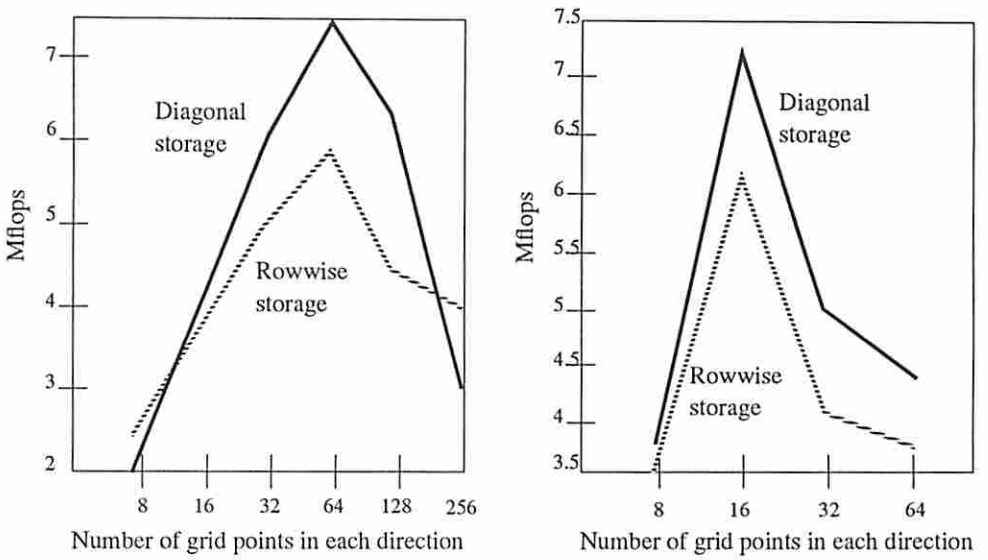
Figure 13: Mflops achieved for different numbers of processors.

size is small, there is a mismatch between hardware and software parallelism. Hence the Mflops rate is moderate. As the grid size is increased, the Mflops rate improves rapidly for both storage formats. However, when the grid size is increased beyond some threshold values, the Mflops drops sharply. The effect can be attributed to the fact that the storage space requirement of large grids exceeds the cache capacity. This phenomenon is referred to as *cache saturation.*

When cache saturation takes place, data have to be loaded from main memory frequently, prompting the replacement of those already residing in the cache. This data thrashing slows down processing speed considerably. Indeed, at the onset of cache saturation, the time spent on memory access may dominate the overall execution time. A possible way to overcome cache saturation is to apply strip mining or tiling [27] technique to improve the reuse of a data item once it is brought into the cache instead of reloading the same data from main memory.

In Figure 15, the diagram shows a more detailed variation of Mflops rates as a function of the grid size for a 2D grid when 5-coloring is used in conjunction with rowwise storage format. There is a general trend of declining Mflops rates with increasing grid sizes.

The idea of strip mining is to divide the vectors into strips so as to better fit the fast



(a) 2D square grids of different sizes

(b) 3D cubic grids of different sizes

Figure 14: Mflops achieved for different grid sizes using 8 processors.

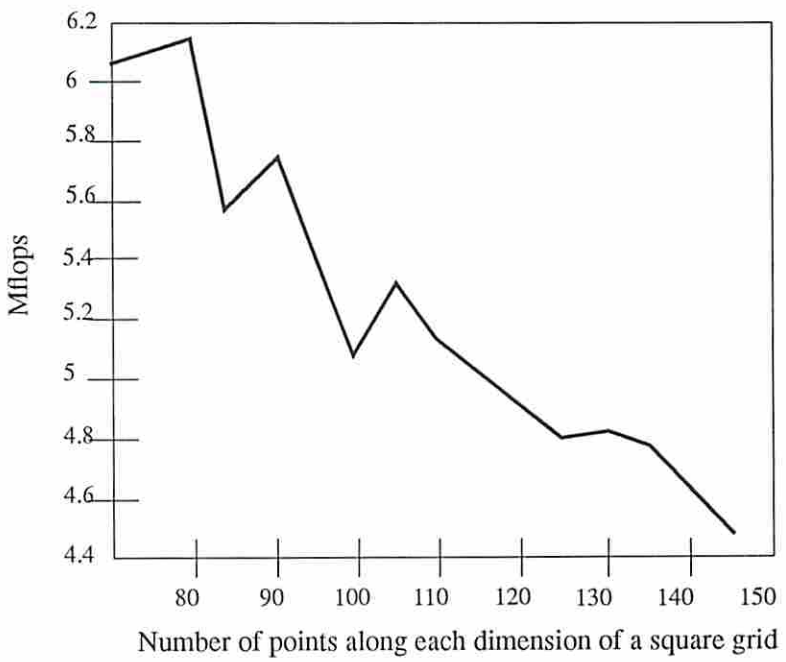


Figure 15: Mflops rates versus the grid size used to illustrate the effect of cache saturation on performance.

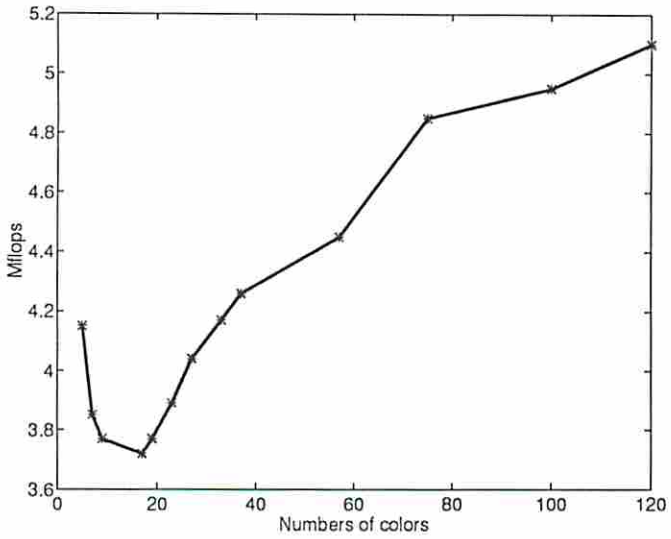


Figure 16: Mflops rates versus number of colors used.

memory such as vector registers or cache memory. In fact, strip mining can be realized by using a larger number of colors. In Figure 16, we show the Mflops rates obtained for a 2D grid with 255×255 points using various numbers of colors defined by the mapping $(r + 2s) \bmod b$ with b the number of colors used. Note that the coloring scheme is valid for any of the numbers of colors shown in the diagram according to the discussion in Section 6.

The figure shows that the Mflops rate improves when the number of colors is increased. With the use of more colors, both the source and destination vectors are divided into more segments which can be loaded to the same vector registers or high-speed memory locations without interfering with other vectors, thereby reducing the detrimental effect of data thrashing.

Clearly, there is a tradeoff as to the optimal number of colors to use. As discussed before, if the number is large, the number of grid points in each color becomes smaller and the degree of parallelism dwindles. The choice of a proper number depends on the size of grids. The goal is to strike a balance between maintaining a high degree of parallelism and ameliorating the damaging effect of data thrashing.

B. Comparison with Saad's Results

We also compare the results with those reported in [23] for Av operation using columnwise

Table 2: Comparison of Mflops rates for Av multiplication using columnwise storage.

Grid size	Saad's results (no coloring)	Our results (with coloring)
20×20	0.19	3.50
30×30	0.19	4.78
$20 \times 20 \times 10$	0.21	7.28
$30 \times 30 \times 10$	0.21	6.27

Table 3: Comparison of Mflops rates for reduction and extension operations on 2D and 3D grids.

Grid size	Av	A^Tv
$7 \times 7 \times 7$	3.97	3.53
$15 \times 15 \times 15$	7.37	6.14
$31 \times 31 \times 31$	5.16	4.12
$63 \times 63 \times 63$	4.44	3.70

storage. The multiplication, as noted before, is an extension operation. Table 2 lists the Mflops rates from the experiments. Their results were obtained on an Alliant FX/80 using 8 processors. The same environment is used in our experiments. Clearly, coloring has significantly improved the performance and shows better scalability with respect to grid size before cache saturation sets in.

C. Comparison of Reduction and Extension Operations

Finally, we compare the performance results of reduction and extension operations for the same storage format. Table 3 shows the Mflops rates for the operations Av and A^Tv using diagonal storage format. The results were obtained for a 3D grid with different sizes using 8 processors with vector processing. In all cases, the reduction operation outperforms the extension operation by 10% to 20%. The difference may be ascribed to the higher DOP of reduction operation and the need for extra memory accesses to retrieve the coloring information in the extension operation. A similar observation has been made for rowwise storage format.

10. Conclusions

We have characterized matrix-vector multiplications as reduction and extension operations depending on the sparse matrix storage scheme used. For extension type of operation, concurrent writes can cause the numerical algorithm to diverge. The multicoloring technique is developed to avoid hazard conditions and improve the performance. The geometric rendition through the introduction of ranges and turfs gives a clear illustration of the hazard conditions and their avoidance.

By recasting multicoloring in an algebraic mapping setting, a systematic approach has been derived for assigning colors to the grids points in a straightforward manner. Using this method, we have been able to obtain performance results far superior to those without coloring. Another feature of the multicoloring scheme is that it imposes little overhead compared to other parallelization techniques. The property ensures high quality of parallel computation [14, 16] with the proposed coloring technique.

In fact, two types of overhead are incurred, one with the determination of colors for individual grid points, and the other for the access of the coloring information during execution. Both costs are negligible compared to the main computations involved in the algorithms. The low overhead associated with the approach makes it attractive compared to existing coloring techniques.

We have also shown that the proposed multicoloring scheme leads to matrices with the desirable DDB structure, making it suitable for use in conjunction with the preconditioning step of most CG-like methods. By numbering the grid points according to their colors and coordinates rather than by coordinates alone (as in natural ordering), we have shown a method for obtaining DDB matrices with the use of the multicoloring technique. The proposed multicoloring can be generalized to parallelize any grid-structured sparse-matrix problems such as for large database manipulation or for scientific simulation modeling.

References

[1] L. Adams. m-step preconditioned conjugate gradient methods. *SIAM J. Sci. Stat. Comp.*, 6:452–463, 1985.

[2] Alliant Computer Systems Corporation. *FX/Series Product Summary*, 1987.

[3] A.J. Bernstein. Analysis of programs for parallel processing. *IEEE Trans. Elec. Computers*, pages 746–757, October 1966.

[4] J.H. Bramble, J.E. Pasciak, and A.H. Schatz. The construction of preconditioners for elliptic problems by substructuring. I. *Math. Comp.*, 47(175):103–134, 1986.

[5] T.F. Chan and Y. Saad. Multigrid algorithms on the hypercube multiprocessor. *IEEE Trans. Computers.*, pages 969–977, November, 1986.

[6] P. Concus, G.H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Stat. Comp.*, 6:309–332, 1985. Also Report LBL-14856, Lawrence Berkeley Laboratory, 1982.

[7] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*, pages 813–820. MIT Press, Cambridge, Mass.; McGraw-Hill, New York, 1990.

[8] C.C. Douglas. *MADPACK (Version 2) Users' Guide.* Mathematical Science Department, IBM Research Division, 1990.

[9] C.C. Douglas and W.L. Miranker. Constructive interference in parallel algorithms. *SIAM J. Numer. Anal.*, 25:376–398, 1988.

[10] T. Dupont, R.P. Kendall, and H.H. Rachford, Jr. An approximate factorization procedure for solving self-adjoint difference equations. *SIAM J. Numer. Anal.*, 5:559–573, 1968.

[11] H.C. Elman. *Iterative Methods for Large, Sparse, Nonsymmetric Systems of Linear Equations.* PhD thesis, Yale University, 1982.

[12] H.C. Elman and E. Agron. Ordering techniques for the preconditioning of conjugate gradient methods on parallel computers. Technical Report UMIACS-TR-88-53, UMIACS, University of Maryland, 1988.

[13] I. Garcia, J.J. Merelo, J.D. Bruguera, and E.L. Zapata. Parallel quadrant interlocking factorization on hypercube computers. *Parallel Computing*, 15:87–100, 1990.

[14] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, pages 105–113. McGraw-Hill, New York, 1993.

[15] K. Hwang and H.C. Wang. A multigrid Schwarz alternating method for parallel solution of elliptic PDE problems. In D.J. Evans et al., editor, *Proc. Int. Conf. on Advances in Parallel Computing*, pages 105–120, 1989.

[16] R.B. Lee. Empirical results on the speedup, efficiency, redundancy, and quality of parallel computations. In *Proc. Int. Conf. on Parallel Processing*, pages 91–96, August 1980.

[17] O.A. McBryan, P.O. Frederickson, J. Linden, A. Schuller, K. Solchenbach, K. Stuben, C. A. Thole, and U. Trottenberg. Multigrid methods on parallel computers: A survey of recent developments. *Impact Comput. Sci. Eng.*, 3:1–75, 1991.

[18] J.A. Meijerink and H.A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp*, 31:148–162, 1977.

[19] R. Melhem and K. Ramarao. Multicolor reordering of sparse matrices resulting from irregular grids. *ACM Trans. Math. Software*, 14(2):117–138, 1988.

[20] N.M. Nachtigal, S.C. Reddy, and L.N. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl.*, 13(3):778–795, 1992.

[21] J.M. Ortega and R.G. Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM Review*, 27(2):149–240, 1985.

[22] E.L. Poole and J.M. Ortega. Multicolor ICCG methods for vector computers. *SIAM J. Numer. Anal.*, 24:1394–1418, 1987.

[23] Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. Sci. Stat. Comp.*, 10(6):1200–1232, 1989.

[24] R. Schreiber and W. Tang. Vectorizing the conjugate gradient method. In *Proc. Symp. CYBER 205 Applications*, Ft. Collins, CO., 1982.

[25] H.C. Wang. *Parallelization of Iterative PDE Solvers on Shared-Memory Multiprocessors.* PhD thesis, University of Southern California, 1992.

[26] H.C. Wang and K. Hwang. Multicoloring for fast sparse matrix-vector multiplication in solving PDE problems. In *Proc. Int. Conf. Parallel Processing*, St. Charles, Illinois, August, 1993.

[27] M.J. Wolfe. Automatic vectorization, data dependence, and optimizations for parallel computers. In Hwang and DeGroot, editors, *Parallel Processing for Supercomputing and Artificial Intelligence*, chapter 11. McGraw-Hill, New York, 1989.