Multicoloring of Grid-Structured PDE Solvers on Shared-Memory Multiprocessors

Hwang-Cheng Wang, *Member, IEEE Computer Society*, and Kai Hwang, *Fellow, IEEE*

Abstract—**In order to execute a parallel PDE (partial differential equation) solver on a shared-memory multiprocessor, we have to avoid memory conflicts in accessing multidimensional data grids. A new multicoloring technique is proposed for speeding sparse matrix operations. 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 (1989) on an identical Alliant system, our results show a factor of 30 times higher performance in Mflops. Multicoloring transforms sparse matrices into ones with a *diagonal diagonal block* (DDB) structure, enabling parallel LU decomposition in solving PDE problems. The multicoloring technique can also 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, cache saturation, multiprocessor performance.**

I. Introduction

Parallel solution of *partial differential equations* (PDE) has attracted much attention in recent years. A number of studies were reported in [5], [7], [11], [13], [15], [19], [23], [24]. In this paper, we present a new multicoloring technique which allows the parallelization of *generalized conjugate gradient* (GCG) methods on a shared-memory multiprocessor system. These methods solve asymmetric systems of equations, often characterized by the multiplication of vectors by a matrix and its transpose. These matrix operations dominate the main computing cost of the PDE algorithms. The sparse matrix operations are highly desirable for parallel processing.

We concentrate on centralized, shared-memory multiprocessor systems because most vector multiprocessors fall into this category. The basic assumption is that only one copy of the data set, consisting of all nonzero elements of the characteristic matrix, is available in the shared memory. Distributed memory algorithms are beyond the scope of this study. No data replication is needed in a shared-memory multiprocessor. The main problem being attacked is the avoidance of access conflicts in the shared memory. Using a *multicoloring* approach, we can eliminate most memory conflicts and thus reduce the solution time significantly.

Most storage formats for sparse-matrix-vector multiplications form a duality of reduction-extension operations. Without coloring, the extension operation may cause the algorithm to diverge due to conflicting updates of the same memory location by more than one processor at the same time. To cope with the problem, a multicoloring technique for grid-structured PDE problems is developed. Coloring is formulated as a mapping from the coordinates of grid points to different colors. Sufficient and necessary conditions are derived for a mapping to generate a valid coloring scheme. The minimum number of colors is established for different discretization stencils.

Numerical experiments were conducted on a shared-memory Alliant FX/80 multiprocessor system in solving 2D and 3D PDE problems. The timing results demonstrate a significant improvement in performance. The results obtained for extension type of operation show an appreciable gain in speed over those reported in the past. With the proposed coloring scheme, the extension operations exhibit good scalability with respect to both 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 [19]. 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 the diagonals. Coloring has also been used with irregular grid structures [17].

Another use of coloring is to form long vectors in order to reduce the startup cost of pipelined processing on a vector processor. These methods were exemplified by the *continuous coloring* scheme proposed by Poole and Ortega in [20]. Related work can also be found in [1], [10], [22]. The main objective of these schemes is to transform a matrix into one possessing a DDB (diagonal diagonal blocks) structure [21]. Similar methods were used to vectorize the preconditioning phase of conjugate-gradient methods based on incomplete (block) factorization of the coefficient matrix [16].

We introduce a concept of *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 the continuous coloring scheme varies with the number of grid points along each coordinate direction. As a result, the scheme may need a number of iterations to determine a suitable coloring. Our method enumerates valid coloring schemes explicitly, thereby eliminating the tedious trial-and-error process.

Manuscript received June 11, 1993; revised Dec. 30, 1994.

H.-C. Wang is with the Department of Information Management and Engineering, Wufeng Institute of Technology, Taiwan, Republic of China.

K. Hwang is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA; e-mail: kaihwang@diana.usc.edu.

To order reprints of this article, e-mail: transactions@computer.org, and reference IEEECS Log Number D95014.

Special data structures are needed for the storage of sparse matrices to conserve the shared memory space. Several commonly used formats are specified below, followed by the corresponding codes for matrix-vector multiplications. These operations are required in GCG methods. Suppose A is a sparse $N \times N$ real matrix. Irrespective of the format used, the basic information to be stored includes the nonzero elements and its row and column indices in the matrix. Several storage schemes are compared and evaluated below:

- *Diagonal format.* For matrices having a banded structure with nonzero elements clustered along fixed diagonals, the nonzero elements can be stored in a V of dimension $N \times d$. A small vector I of dimension d stores the offset of each element relative to the main diagonal.
- *Rowwise format.* A vector V of size Z is used to store the nonzero elements of the matrix row by row. A second vector J of the same length as A stores the column index of each nonzero element. A third vector S of size $N + 1$ indicates where each row of the sparse matrix starts.
- *Columnwise format.* This is the counterpart of rowwise format. Similar to the rowwise format, a vector V holds the nonzero elements. Another vector I stores the row index of each nonzero elements, and a third vector S stores the the starting location of each column in A. This format is used in Harwell-Boeing sparse matrix collection [8].
- *ITPACK format.* A 2D array V 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 J of the same size indicates the column index of each nonzero element. This format has been used in the 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 required data structures are similar to the rowwise format except vector S stores the starting location of each column stripes in this scheme. An auxiliary vector X records the correct row index of each shuffled row. This format was proposed in [17] to improve vector processing efficiency.

Fig. 1 shows the use of the various formats for storing the following matrix:

$$A = \begin{pmatrix} 8 & -1 & 0 & 0 & 0 \\ 1 & 10 & 11 & 0 & 0 \\ 0 & 4 & 9 & 3 & 0 \\ 0 & 0 & 7 & 5 & 6 \\ 0 & 0 & 0 & -3 & 2 \end{pmatrix}. \quad (1)$$

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

The paper is organized as follows: The first section introduces the problem environment and characterizes several storage formats for sparse matrix-vector multiplications. In Section II, two iterative solvers are presented for solving nonself-adjoint PDEs. In Section III, matrix-vector multiplication operations are characterized as *reduction* and *extension* operations, depending on the storage scheme used. In Section IV, the coloring technique for a 2D grid is presented; four color assignment strategies and the resulting coloring are given.

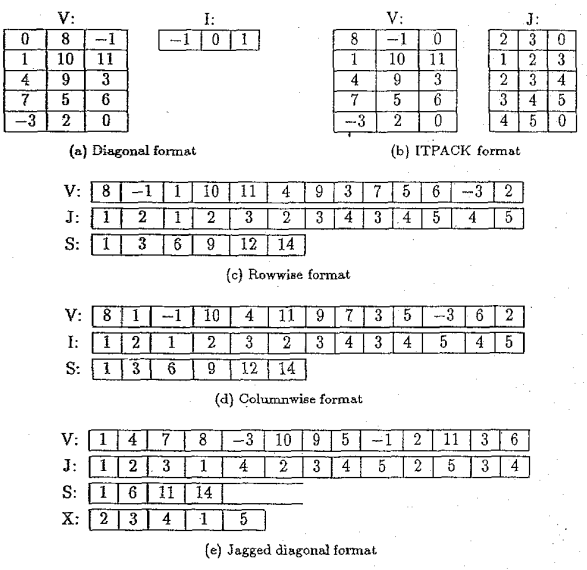


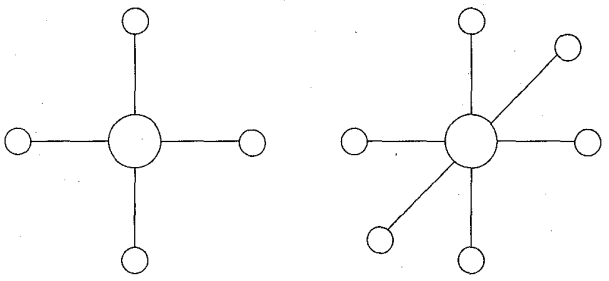
Fig. 1. Storage formats for the sparse matrix in (1).

In Section V, the coloring of a 3D grid is formulated as a linear mapping. Rigorous analysis leads to useful rules for coloring the grid points. Based on these rules, results obtained in Section IV are proved. In Section VI, we deal with matrices with a DDB structure and show that the proposed coloring scheme leads to DDB matrices. Section VII describes the numerical experiments performed on an Alliant FX/80 shared-memory system. Performance data obtained from numerical experiments are analyzed and compared with previous results. Finally, we comment on generalization of the multicoloring methods for parallel solution of other grid-structured problems.

II. Grid-Structured PDE Solvers

We limit the discussion to a system of linear equations arising from the discretization of a PDE problem over a regular region. Fig. 2 shows that for a 2D problem, the grid size is $m \times n$ with a 5-star discretization stencil; and for a 3D problem, a grid of size $m \times n \times p$ is used with a 7-star stencil. The discretization leads to 2D and 3D grids in Fig. 3, with the grid points numbered in a natural ordering and each grid point referred to by its coordinates. These problems are 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, any one of the storage formats discussed in the previous section can be used.

A difference between the model problems and real PDE problems is that the coefficients in the stencil are randomly gen-



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

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

erated positive numbers between 0 and 1. A ramification is that the corresponding matrices are made neither M-matrix[1] nor diagonally dominant. However, the model problems capture the essence of asymmetric PDE problems for which GCG methods are used, such as the *CG method applied to normal equations* (CGNR) and the *biconjugate-gradient* (BCG) methods.

These methods were shown to be suitable in solving different types of linear systems [18]. In particular, since CGNR solves the normal equation $A^TAu = A^Tf$ by a conjugate gradient method, the rate of convergence is determined by the singular values of matrix A. 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. These two methods are specified below:

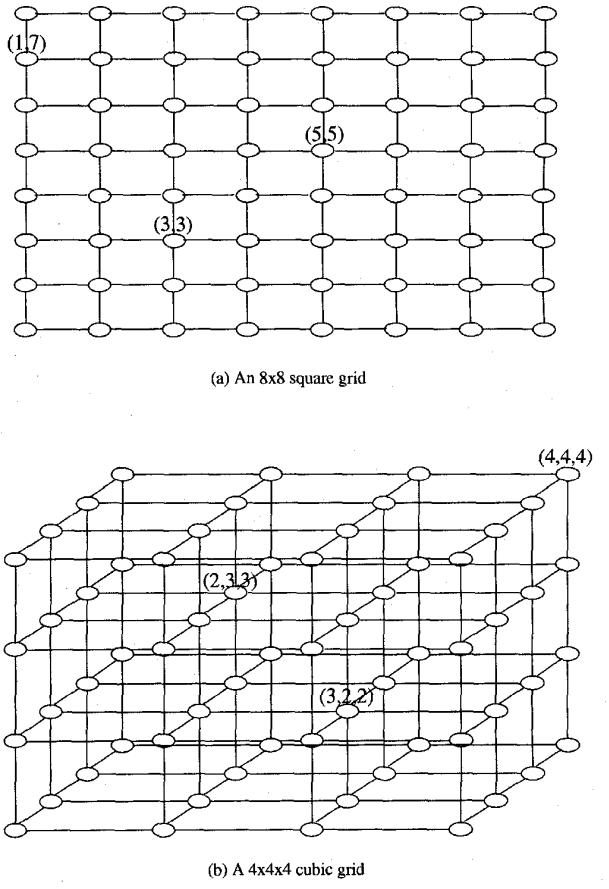
Algorithm 1 (CGNR method)

1. Initialization: Choose u_0 as an initial guess and compute $r_0 = f - Au_0$ and $p_0 = A^Tr_0$.
2. **For** $k = 0, 1, 2, \ldots$ until convergence **do**
   - 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 $p_{k+1}^* = r_{k+1}^* + \beta_k p_k^*$.

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

Fig. 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, respectively.

III. Reduction and Extension Operations

Both CGNR and BCG involve the multiplication of vectors by a matrix A and its transpose A^T. Performing both multiplications in the same algorithm is rather inefficient. 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, the operation A^Tv is an *extension* operation whereby the action of a scalar quantity can affect the values of several others. The concept is illustrated in Fig. 4.

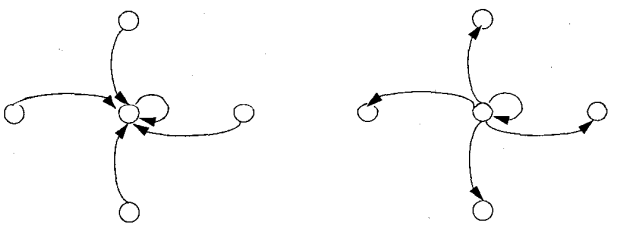


Fig. 4. Reduction and extension operations for a 2D grid.

The two matrix multiplication operations form a duality. The reduction and extension nature of the operations can be reversed, depending on the storage scheme used. For instance, if a columnwise storage scheme 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 a rowwise format is used.

The basic operation in the matrix-vector multiplication $u = Av$ is shown below, where u and v are column vectors of length N:

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

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

$$u_i = \sum_{j=1}^{Z_i} A_{ij}^T v_j = \sum_{j=1}^{Z_i} A_{ji} v_j . \tag{3}$$

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

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

The two types of multiplications are specified below in the rowwise storage format:

```
Operation u = Av:
 For i = 1 to N do
   For j = S(i) to S(i+1) - 1 do
    u(i) = u(i) + V(j) * v(J(j))
   Enddo
 Enddo
```

```
Operation u = Atv:
 For i = 1 to N do
   For j = S(i) to S(i+1) - 1 do
    u(J(j)) = u(J(j)) + V(j) * v(i)
   Enddo
 Enddo
```

Consider the operation $A^T v$ in the 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 $J(j)$ for a particular combination of i and j. Therefore, the code is not optimized for parallel execution. However, since the vector v is read-only, the element access order is irrelevant. Moreover, for the model problem each $J(j)$ corresponds to one of the neighbors of the grid point indexed by i. Hence, the values of $J(j)$ 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.

Different combinations of i and j may lead to the same value of $J(j)$. This is an example of the *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 [12]. The WAW hazard implies potential access conflicts in the shared memory. This mode of execution is termed *concurrent outer* (CO) mode. The hazard is likely to deteriorate the performance of the GCG algorithms. In Table I and Fig. 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 was obtained on a single processor running in scalar mode, and the parallel execution time is obtained on a 4-processor Alliant FX/80 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 the diagonal storage format. Therefore, it is not always possible to get rid of the hazard conditions by merely choosing a different storage scheme.

TABLE I
SEQUENTIAL AND PARALLEL EXECUTION TIME (IN msec) OF $U = A^T v$ USING THE 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

IV. MULTICOLORING OF A 2D GRID

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 an index src as the set of points whose values are affected by the operation. From the perspective of the outer loop in the code in a 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 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 $R_{i1} \cap R_{i2} = \phi$. This is Bernstein's third condition for parallel processing [3].

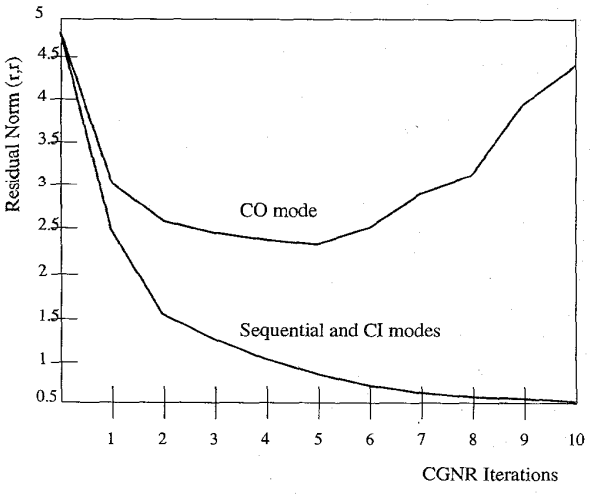


Fig. 5. Value of the residual norm (r, r) for the first 10 CGNR iterations in two execution modes.

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 to update P and Q concurrently. Graphically, a rhombic region surrounding point P can be drawn, which is referred to as the *turf* of point P, denoted $\mathcal{T}_p$. Equation (5) 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 grid points separated by a distance of 3 along each coordinate have the same color, satisfies (5). More compact 6-coloring and 5-coloring schemes are also allowed. With 6-coloring, points in the same color are separated by 3 in the horizontal direction and by 2 in the vertical direction. Fig. 6d shows that a 4-coloring scheme, in which points separated by a distance of 4 in one direction and 1 in the other are assigned the same color, gives rise to a write hazard.

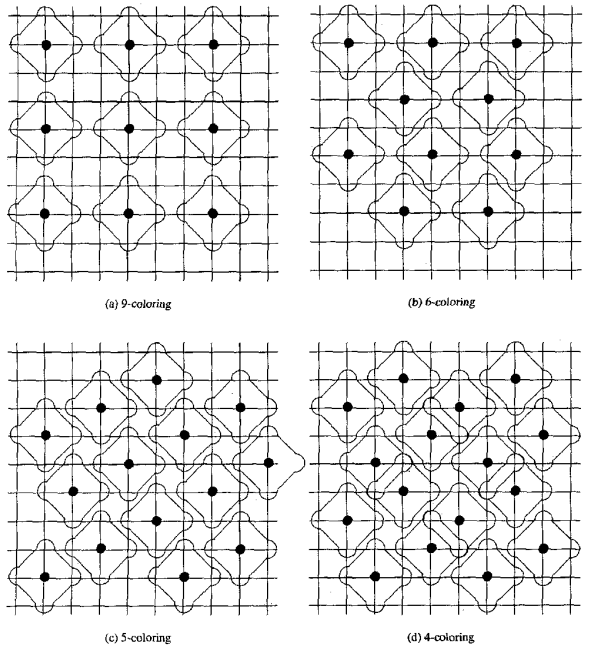


Fig. 6. Four coloring schemes with different numbers of colors. Each stencil shows the ranges of grid points of the same color. The coloring schemes in (a), (b), and (c) are valid, while the scheme in (d) is invalid.

From the diagrams in Fig. 6, it is easy to determine the set of points of the same color. It is also easy to label the color of a grid point from its coordinates. For instance, in the 9-coloring scheme, the color of a point at (r, s) is numbered by $3(s \bmod 3) + r \bmod 3$. In the case of 5-coloring, the same point is painted in color number $(3s + r) \bmod 5$. 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 $(s \bmod 4)$. The pattern selection process is repeated until all points on a line s are labeled. In practice, the assignment of colors can be carried out performed in parallel. Fig. 7 shows the color of each grid point on a 9×8 grid. The turfs associated with two points (circled) in the grid are indicated.

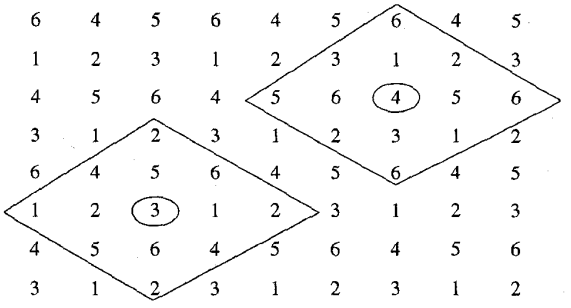


Fig. 7. Color assignment on a 9×8 grid using a 6-coloring scheme. Each rhombic area represents the turf of a circled point.

Using the 6-coloring, we obtain the timing results shown in the last column of Table I. It is useful to compare the results with those of the CI and CO modes. As the problem size increases, the performance of the 6-coloring scheme improves steadily relative to that of the concurrent outer mode. This is evident from the fact that the number of grid points of each color is larger and thus concurrent processing is more feasible with a higher degree of parallelism. CI mode shows a little improvement in speedup because of limited parallelism.

V. Multicoloring of a 3D Grid

For a 3D grid using 7-star discretization stencils, we expect seven colors to be sufficient for avoiding the concurrent write conflicts. The geometric rendition useful for 2D grids is difficult to depict for a 3D grid. In the following, we describe an algebraic approach to coloring and illustrate the idea with a seven-color scheme.

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) using the following formula:

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

These integers m_1, m_2, m_3 are the *weights* in the mapping. In general, once the number of colors is fixed, a coloring scheme is uniquely determined by the weights used.

Conceptually, a valid coloring can be determined by 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 divide grid points by different colors and color assignment is invariant with regard to coordinates, the number of distinct mappings is much smaller. In fact, using this coordinate-invariance property, we can reduce the number of distinct mappings to 56. In general, for a b-coloring scheme, $(b-1)b(b+1)/6$ distinct combinations are obtained after the invariance property is applied. For large b, exhaustive search is very time-consuming. Therefore, it is desirable to determine valid mappings systematically. To this end, we introduce the concept of *equivalent mappings*.

DEFINITION 1. *Two mappings are equivalent if two grid points assigned the same color c_1 by the first mapping are assigned the same color c_2 by the second 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 $M_2(r_j, s_j, t_j) = c_2$.

The equivalence relation can be used to partition the state space into equivalence classes. The concept is illustrated in Fig. 8. Equivalent mappings allow us to search for the representative state in each partition instead of inspecting all the individual states, thereby reducing the amount of search time required. Based on the above definition, we derive the following results:

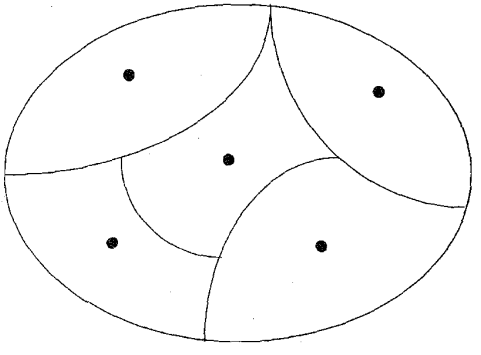


Fig. 8. Partition of the state space into equivalence classes. The dot in each class stands for the representative mapping in that class.

LEMMA 1. *Suppose a mapping M_1 is defined by the 3-tuple $\langle m_1, m_2, m_3 \rangle$ and another mapping M_2 by $\langle Sm_1, Sm_2, Sm_3 \rangle$ with a scaling factor S where $1 \leq S \leq 6$. 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,

$$(m_1r_1 + m_2s_1 + m_3t_1) \bmod 7 = c_1,$$

$$(m_1r_2 + m_2s_2 + m_3t_2) \bmod 7 = c_1,$$

$$S(m_1r_2 + m_2s_2 + m_3t_2) \bmod 7 = c_2.$$

From the above equations, we have

$$[(m_1r_1 + m_2s_1 + m_3t_1) - (m_1r_2 + m_2s_2 + m_3t_2)] \bmod 7 = 0.$$

Hence,

$$S[(m_1r_1 + m_2s_1 + m_3t_1) - (m_1r_2 + m_2s_2 + m_3t_2)] \bmod 7 = 0.$$

Therefore,

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

which establishes the equivalence relation. □

THEOREM 1. *Suppose a mapping M_1 is defined by the 3-tuple $\langle m_1, m_2, m_3 \rangle$ and another mapping M_2 by $\langle Sm_1 \bmod 7, Sm_2 \bmod 7, Sm_3 \bmod 7 \rangle$ where $1 \leq S \leq 6$. Then the two mappings M_1 and M_2 are equivalent.*

PROOF. First observe the relation

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

follows $Sm_1 = \lfloor S\, m_1 / 7 \rfloor \times 7 + (Sm_1) \bmod 7$. 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, by Lemma 1,

$$[(Sm_1 \bmod 7)r_2 + (Sm_2 \bmod 7)s_2 + (Sm_3 \bmod 7)t_2] \bmod 7$$
$$= (Sm_1r_2 + Sm_2s_2 + Sm_3t_2) \bmod 7$$
$$= c_2.$$ □

COROLLARY 1. *In 7-coloring, the mappings $\langle Sm_1, Sm_2, Sm_3 \rangle$ for $S = 1, 2, 3, 4, 5, 6$ are all 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 $\langle 1, 1, 2 \rangle$ is said to *cover* the other equivalent mappings. By the equivalence theorem, the number of unique mappings can be reduced 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 22 mappings lead to valid colorings which can prevent concurrent write conflicts. For example, Fig. 9 shows the values of the 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 the valid mappings in the minimum cover set by eliminating the invalid mappings from the set. For this purpose, we resort to a combination of geometric and algebraic approaches. Similar to the 2D case, 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:

$$\mathcal{T}_p = \{Q: |\triangle r| + |\triangle s| + |\triangle t| \leq 2\}, \quad (7)$$

where $\triangle$ stands for the distance along each direction between

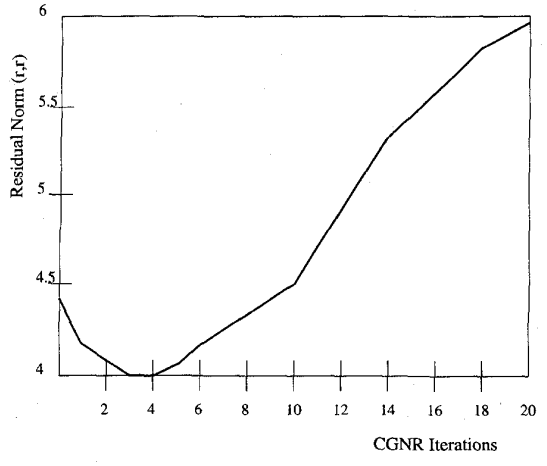


Fig. 9. Residual norm versus CGNR iterations in parallel execution with the use of $\langle 1, 1, 1 \rangle$ 7-coloring scheme.

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 of the same color satisfies the condition:

$$(m_1 \Delta r + m_2 \Delta s + m_3 \Delta t) \bmod 7 = 0. \tag{8}$$

Hence, the goal is to find values of m_1, m_2, and m_3 such that any point $Q \in \mathcal{T}_p$, $Q \neq P$ will have a different color than P. From (7), a point in $\mathcal{T}_p$ other than P satisfies one of the following conditions:

- One of $|\Delta r|$, $|\Delta s|$, $|\Delta t|$ is 2, the others are 0;
- One of $|\Delta r|$, $|\Delta s|$, $|\Delta t|$ is 1, the others are 0;
- Two of $|\Delta r|$, $|\Delta s|$, $|\Delta t|$ are 1, the third is 0.

In the first two cases, since $1 \leq m_1, m_2, m_3 \leq 6$, the condition in (8) is violated, hence Q and P will have different colors. In the last case, (8) is satisfied only if

$$(m_i \pm m_j) \bmod 7 = 0 \text{ for } i \neq j. \tag{9}$$

Because $1 \leq m_i \leq 6$, the condition (9) holds if

$$m_i = m_j \text{ or } m_i + m_j = 7 \text{ for } i \neq j. \tag{10}$$

A direct consequence of (10) is that any coloring scheme with two or more identical weights will lead 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 produce the valid 7-colorings.

Although the foregoing discussion was 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, respectively. Let $\langle m_1, \ldots, m_d \rangle$ be the coloring scheme where $d = 2$ or 3. We have the following set of rules on the selection of the weights:

1) $0 \leq m_i \leq b$ for all i.
2) $2\, m_i \neq b$ 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 these rules, it is straightforward to determine valid choices for the weights. Moreover, using these rules, we can show that neither 4-coloring for a 2D grid nor 6-coloring for a 3D grid is valid. For instance, if 6-coloring is applied to a 3D grid, the possible choices for any of the three weights are from 1 to 5. Value 3 is disallowed since it violates rule 2. The numbers 1 and 5 cannot appear together since rule 3 will be violated, neither can 2 and 4. Also, rule 4 prohibits any number from being chosen more than once. Thus, it is impossible to form any legitimate 3-tuple, and hence 6-coloring 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 in 2D PDEs with mixed differential terms, these rules need to be modified. Fig. 10 shows the range and turf of a grid point on a 2D grid when a 9-point stencil is used.

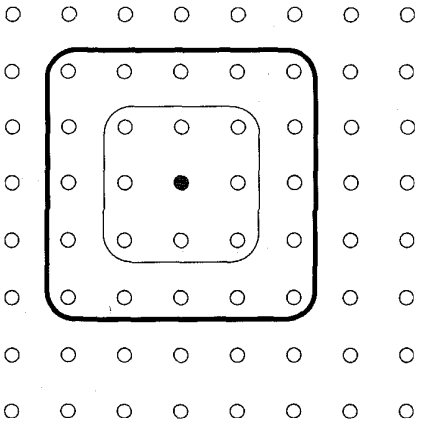


Fig. 10. The range and turf associated with a grid point in a 2D grid generated by a 9-point stencil.

VI. Multicoloring and the DDB Matrices

Because of the periodicity 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 the first (mn mod b) colors and $\lfloor mn / b \rfloor$ for the remaining colors.

Furthermore, matrices resulting from the use of multicoloring have a *diagonal diagonal block* (DDB) structure [21]. The grid points are numbered according to the colors. We number the grid points of color 1 first, followed by all grid points of color 2, and so on. Within each color, the grid points are numbered by natural ordering. This 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 (r, s) by $C(r, s)$ and its order in the coloring scheme by $order(r, s)$. Fig. 11a shows the color of each grid point in the 5-coloring of an 8×8 grid by the mapping $\langle 1, 3 \rangle$:

$$C(r, s) = (r + 3s) \bmod 5. \tag{11}$$

There are 13 grid points of colors 1, 2, 4, and 5 each, and 12 points of color 3. This reflects a uniform distribution property. Fig. 11b shows the ordering of the grid points corresponding to the coloring.

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

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

The DDB property essentially states that each diagonal block corresponding to a group G_c of grid points should be diagonal. The following theorem is used to test whether a matrix satisfies the DDB property.

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 also belongs to G_c, then there is a nonzero off-diagonal element in A_c in 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 is 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 [20]. □

For 2D and 3D grids, the neighbors of a grid point are all in different colors than it. Therefore, the DDB property is satisfied with the use of the multicoloring technique. In Fig. 12, we show the coefficient matrix corresponding to the grid in Fig. 11. Each plus sign "+" in the diagram represents a nonzero element in the matrix. Clearly, the matrix satisfies the DDB property. Besides the implied meaning of using more than two colors, multicoloring has been used to represent the class of general coloring schemes leading to matrices with DDB structure [21]. In light of this, our use of the term is consistent with the convention.

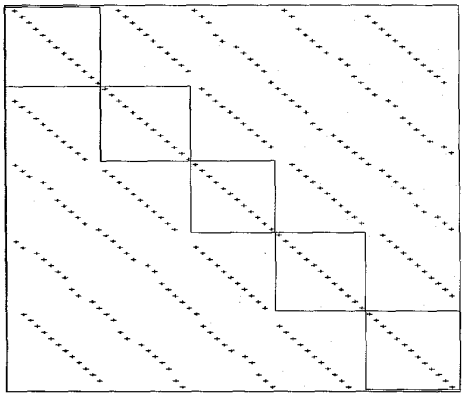


Fig. 12. The matrix resulting from the ordering shown in Fig. 11b.

The DDB structure has an important application in parallel processing of preconditioned conjugate gradient algorithms. In particular, it is very useful in parallel implementation of preconditioners based on incomplete LU decomposition of the coefficient matrix [6], [9], [16]. 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.

In summary, we have proved the DDB matrices can be obtained by multicoloring. This transformation is useful in parallel computation of the preconditioning steps. In conjugate gradient methods, preconditioning is an integral part of these algorithms. In fact, preconditioning can be used to significantly improve the convergence rate of the CG algorithms [4]. Thus, in addition to our original goal of resolving the concurrent write conflicts in extension type of operations, the multicoloring technique can also be used to improve the parallel performance of preconditioning operations.

VII. Numerical Results and Analysis

Numerical experiments have been conducted on an Alliant FX/80 multiprocessor with shared memory. The machine has a memory hierarchy consisting of cache and main memory. The cache serves a small-capacity, fast-access buffer storage. The system also provides compiler directives to facilitate parallel execution and vectorization [2].

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

The total number of nonzero elements in the coefficient matrix equals $5mn - 2(m + n)$ on an $m \times n$ 2D grid and equals $7mnp - 2(mn + mp + np)$ on an $m \times n \times p$ 3D grid. For each element, two arithmetic operations, one multiplication and one addition, are performed. Therefore, the number of floating-point operations equals $10mn - 4(m + n)$ and $14mnp - 4(mn + mp + np)$ for 2D and 3D model problems, respectively. This information together with the execution time allows us to estimate the Mflops rate.

Processing at grid points proceeds by colors. The colors define synchronization points for parallel processing. In other words, all grid points of the same color must have been processed before those of 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 must match match hardware parallelism which is a combination of pipelined vector processing within each processor and concurrent processing across multiple processors. Color assignment is performed only once at the beginning with the color information stored for subsequent runs of the PDE solvers.

The performance of matrix-vector operations depends on the problem size and the machine size. In general, we expect the performance to improve with an increasing number of grid points, since a higher degree of parallelism is expected. Likewise, when hardware parallelism is increased by the use of more processors, performance should improve accordingly. But certain hardware constraints may limit the speedup from grid size increase.

A. Measured Mflops Performance

In Fig. 13 we show the measured Mflops as a function of the number of processors for two grid sizes, 63×63 and $31 \times 31 \times 31$, respectively. Vector processing is enabled in all cases. In each figure, solid curves represent results using the diagonal storage format and dashed curves correspond to results using the rowwise format. Clearly, with the increase in hardware parallelism, the Mflops rate increases monotonically. Multicoloring has generated sufficient parallelism to utilize the multiprocessor resources.

Fig. 14 shows the Mflops rates obtained in extension operations using 8 processors for different grid sizes. Parallelism exploited increases initially with respect to the number of grid points in each dimension. When the grid size is small, there exists a mismatch between hardware and software parallelism. Hence the Mflops rate is moderate. As the grid size increases, the Mflops rate improves rapidly with both storage formats. However, when the grid size increases beyond some threshold value, the Mflops rate drops sharply. The effect is attributed to the fact that the storage space requirement of a large grid exceeds the cache capacity. This phenomenon is referred to as *cache saturation*.

With a 2D grid, the Mflops rate increases to a maximum value for a square grid of size 63×63 (Fig. 14a). This grid size reaches the limit of the cache capacity used in the 8-processor system. Beyond the cache saturation point, the dropping in speed performance is due to excessive cache miss penalties experienced in the Alliant FX/80 multiprocessor system. The same cache becomes saturated for a 3D grid of size $15 \times 15 \times 15$ as shown in Fig. 14b. The cache block size also affects the cache miss ratio in addition to the effect of total cache capacity.

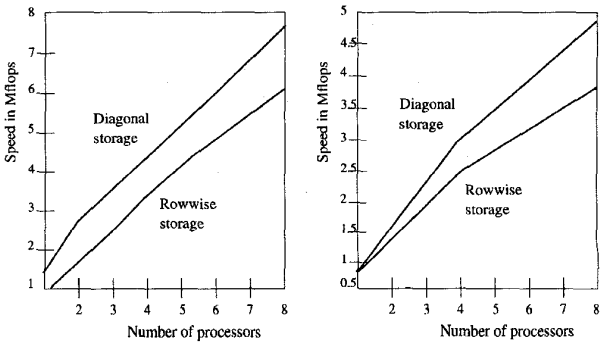


Fig. 13. Mflops achieved with up to eight processors in the Alliant FX/80.

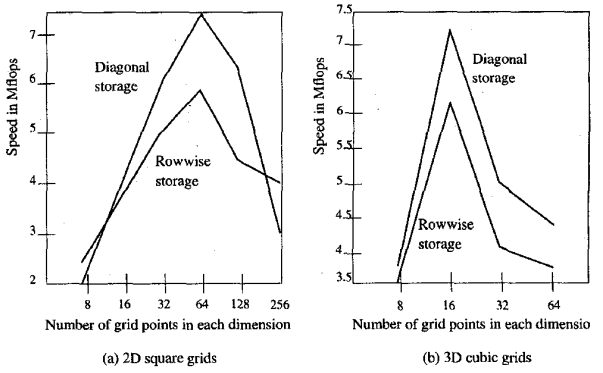


Fig. 14. The Mflops rate achieved with different grid sizes using an Alliant multiprocessor with eight processors.

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 parallel processing 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 the difficulty of cache saturation is to apply strip mining or tiling [25], a technique used to improve the reuse of a data item once they are are brought into the cache instead of reloading the same data from the main memory repeatedly.

Fig. 15 shows the variation of Mflops rates as a function of the grid size when 5-coloring is used with the rowwise storage format. There is a general trend of a declining Mflops rate with an increasing grid size. All of these grid sizes exceeded the cache limit. The fluctuation is caused by variations of cache misses when data items are fetched across cache block boundaries. The envelope of the fluctuating curve in Fig. 15 shows a steadily declining performance as the grid size becomes larger.

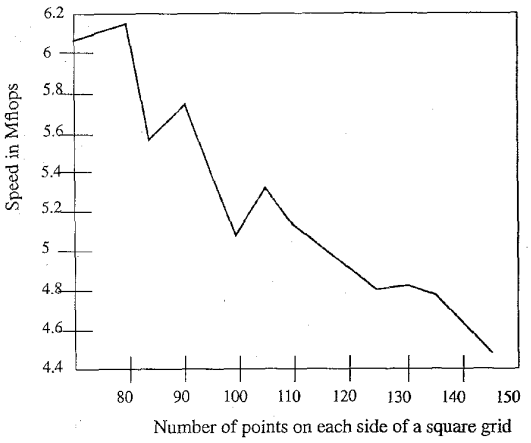


Fig. 15. The Mflops rate versus the grid size illustrating the effect of cache saturation on the multiprocessor performance.

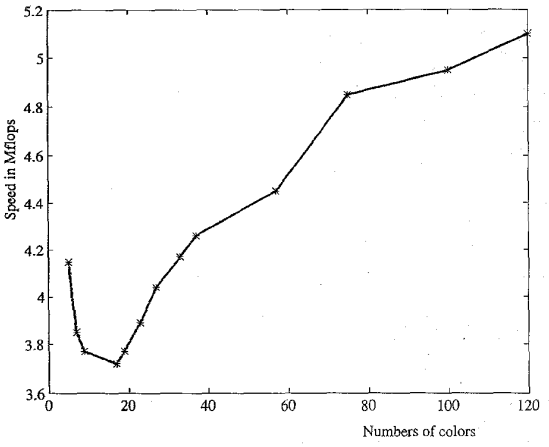


Fig. 16. The Mflops rate versus the number of colors assigned.

The idea of strip mining is to divide the long vectors into shorter strips to better fit vector registers or the cache memory. In fact, strip mining can be realized by using a larger number of colors. In Fig. 16, we show the Mflops rates obtained for a 2D grid with 255×255 points using various numbers of colors defined by the mapping $(r + 2s) \bmod b$ where b is the number of colors used.

The curve in Fig. 16 shows that the Mflops rate improves when the number of colors increases. With more colors, both the source and destination vectors are divided into more segments which are loaded to the same vector registers or to the same cache blocks This will reduce the detrimental effect of data thrashing.

If the number of colors is large, the number of grid points of each color becomes smaller and the degree of parallelism dwindles. The choice of a proper number of colors depends on the size of grids. The general rule is to strike a balance between maintaining a high degree of parallelism and ameliorating the damaging effect of data thrashing. The balanced choice of the number of colors is affected by the *grid size*, the *cache capacity*, the *cache block size*, *write-back* or *write-through caches* used, and the *level-2 cache size* being shared. Continued research is needed to optimize the grid sizes for different cache sizes. We conjecture that the performance would scale up if the cache size were unlimited.

B. Comparison With Saad's Results

We compare our results with those reported by Saad in [21] for Av operation using the columnwise storage scheme. Table II lists the Mflops rates from our experiments. Their results were obtained on a similar Alliant FX/80 with 8 processors. The same software environment was used in our experiments. Our results show an improvement factor of more than 30 over the Mflops rate obtained by Saad on an identical Alliant FX/80 multiprocessor. Checking the last column of Table II, we also observe the better scalability with respect to the increase in grid size before the cache is saturated.

TABLE II
COMPARISON OF MFLOPS RATES FOR THE Av MULTIPLICATION AS EXTENSION OPERATIONS

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

C. Reduction and Extension Operations

Finally, we compare the performance results of reduction and extension operations using the same storage format. Table III shows the Mflops rates for the operations Av and A^Tv respectively using the diagonal storage format. The results were obtained for a 3D grid on 8 processors with vector processing. In all cases, the reduction operation outperforms the extension operation by a factor of 10% to 20%. The difference may be attributed to the higher DOP in the reduction operation and the need for extra memory accesses to retrieve the color information in extension operations.

TABLE III
MFLOPS RATES FOR REDUCTION AND EXTENSION OPERATIONS

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

VIII. CONCLUSIONS

Shared-memory access conflicts may badly slow down grid-structured PDE solvers on a symmetric multiprocessor. What we have developed is a simple coloring approach that eliminates potential memory conflicts in solving PDE problems. The main contribution of this paper is to provide a systematic method for parallelizing sparse matrix-vector multiplications. Thus, the multicoloring technique offers a useful tool for partitioning grid-structured data elements for efficient parallel processing on a shared-memory multiprocessor.

We have characterized matrix-vector multiplications as reduction and extension operations depending on the sparse matrix storage scheme used. For extension operations, concurrent writes can diverge the numerical algorithm. The multicoloring technique can effectively avoid memory-access hazard conditions and improve the multiprocessor performance. The introduction of ranges and turfs reveals the memory hazard conditions and suggests methods for their avoidance.

By recasting multicoloring in an algebraic mapping setting, a systematic approach has been derived to assign colors to grid points in a straightforward manner. Using this method, we obtain improved performance far superior to those without coloring. Another advantage of the multicoloring scheme is that it imposes very little overhead compared to other parallelization techniques. The property ensures a high quality of parallel computation [12], [14] with the proposed coloring technique.

To be precise, two types of overhead are incurred: one with the determination of colors for individual grid points, and the other for the access of color information during execution. Both costs are negligible compared to the main computations involved in the PDE algorithm. The low overhead associated with our approach makes it attractive for implementation on real multiprocessors.

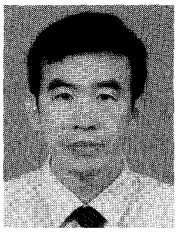
The proposed multicoloring scheme can transform a characteristic matrix into one with the desirable DDB structure, making it suitable for parallel preconditioning in CG-like PDE methods. By numbering the grid points according to their colors and coordinates rather than by coordinates alone (as in natural ordering), we have developed a method for obtaining DDB matrices. The proposed multicoloring can be generalized to parallelize any grid-structured sparse-matrix problems such as those used in large database manipulation and general scientific simulation modeling.

ACKNOWLEDGMENT

Preliminary results of the work were presented in the *International Conference on Parallel Processing*, August 1993.

REFERENCES

[1] L. Adams, "*m*-step preconditioned conjugate gradient methods," *SIAM J. Sci. Stat. Comp.*, vol. 6, pp. 452–463, 1985.
[2] Alliant Computer Systems Corp., *FX/Series Product Summary*, 1987.
[3] A.J. Bernstein, "Analysis of programs for parallel processing," *IEEE Trans. Elec. Computers*, pp. 746–757, Oct. 1966.
[4] J.H. Bramble, J.E. Pasciak, and A.H. Schatz, "The construction of preconditioners for elliptic problems by substructuring: I," *Math. Comp.*, vol. 47, no. 175, pp. 103–134, 1986.
[5] T.F. Chan and Y. Saad, "Multigrid algorithms on the hypercube multiprocessor," *IEEE Trans. Computers*, pp. 969–977, Nov. 1986.
[6] P. Concus, G.H. Golub, and G. Meurant, "Block preconditioning for the conjugate gradient method," *SIAM J. Sci. Stat. Comp.*, vol. 6, pp. 309–332, 1985, Also Report LBL-14856, Lawrence Berkeley Laboratory, 1982.
[7] C.C. Douglas and W.L. Miranker, "Constructive interference in parallel algorithms," *SIAM J. Numer. Anal.*, vol. 25, pp. 376–398, 1988.
[8] I.S. Duff, R.G. Grime, and J.G. Lewis, "Sparse matrix test problems," *ACM Trans. Math. Software*, vol. 15, pp. 1–14, 1989.
[9] T. Dupont, R.P. Kendall, and H.H. Rachford, Jr., "An approximate factorization procedure for solving self-adjoint difference equations," *SIAM J. Numer. Anal.*, vol. 5, pp. 559–573, 1968.
[10] 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, Univ. of Maryland, 1988.
[11] I. Garcia, J.J. Merelo, J.D. Bruguera, and E.L. Zapata, "Parallel quadrant interlocking factorization on hypercube computers," *Parallel Computing*, vol. 15, pp. 87–100, 1990.
[12] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, 1993.
[13] K. Hwang and H.C. Wang, "A multigrid Schwarz alternating method for parallel solution of elliptic PDE problems," *Proc. Int'l Conf. on Advances in Parallel Computing*, D.J. Evans et al., Ed., pp. 105–120, 1989.
[14] R.B. Lee, "Empirical results on the speedup, efficiency, redundancy, and quality of parallel computations," *Proc. Int'l Conf. on Parallel Processing*, Aug. 1980, pp. 91–96.
[15] 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.*, vol. 3, pp. 1–75, 1991.
[16] 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*, vol. 31, pp. 148–162, 1977.
[17] R. Melhem and K. Ramarao, "Multicolor reordering of sparse matrices resulting from irregular grids," *ACM Trans. Math. Software*, vol. 14, no. 2, pp. 117–138, 1988.
[18] N.M. Nachtigal, S.C. Reddy, and L.N. Trefethen, "How fast are nonsymmetric matrix iterations?" *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 3, pp. 778–795, 1992.
[19] J.M. Ortega and R.G. Voigt, "Solution of partial differential equations on vector and parallel computers," *SIAM Rev.*, vol. 27, no. 2, pp. 149–240, 1985.
[20] E.L. Poole and J.M. Ortega, "Multicolor ICCG methods for vector computers," *SIAM J. Numer. Anal.*, vol. 24, pp. 1,394–1,418, 1987.
[21] Y. Saad, "Krylov subspace methods on supercomputers," *SIAM J. Sci. Stat. Comp.*, vol. 10, no. 6, pp. 1,200–1,232, 1989.
[22] R. Schreiber and W. Tang, "Vectorizing the conjugate gradient method," *Proc. Symp. CYBER 205 Applications*, Ft. Collins, Colo., 1982.
[23] H.C. Wang, "Parallelization of iterative PDE solvers on shared-memory multiprocessors," PhD thesis, Department of Electrical Engineering–Systems, Univ. of Southern California, 1992.
[24] H.C. Wang and K. Hwang, "Multicoloring for fast sparse matrix-vector multiplication in solving PDE problems," *Proc. Int'l Conf. Parallel Processing*, St. Charles, Ill., Aug. 1993, vol. 3: *Algorithms and Applications*, pp. 215–222.
[25] M.J. Wolfe, "Automatic vectorization, data dependence, and optimizations for parallel computers," *Parallel Processing for Supercomputing and Artificial Intelligence*, K. Hwang and DeGroot, Eds., chap. 11., McGraw-Hill, New York, 1989.



Hwang-Cheng Wang earned a PhD degree in computer engineering from the University of Southern California in Los Angeles in 1992. He is an associate professor at Wufeng Institute of Technology, Taiwan, Republic of China. His research interests are in computer architecture, parallel and distributed processing, and scientific computing. Dr. Wang is a member of the IEEE Computer Society and ACM.

Kai Hwang is a professor of electrical engineering and computer science at the University of Southern California in Los Angeles. He has authored or coauthored 130 scientific papers and five books on computer architecture, digital arithmetic, and parallel processing. He has served on the ACM SIGARCH Board of Directors and the founding editor of the *Journal of Parallel and Distributed Computing*. His present research is focused on scalable, microprocessor-based multiprocessor with distributed shared memory applying heterogenous programming paradigms. He is an IEEE Fellow and an ACM distinguished lecturer. He received the Courvoissier Leadership Award and Outstanding Achievement Award for contributions to the academic community.