

# Parallel Implementation of the Bi-CGSTAB Method with Block Red-Black Gauss-Seidel Preconditioner Applied to the Hermite Collocation Discretization of Partial Differential Equations

Stephen H. Brill

*Department of Mathematics, Boise State University, Boise, Idaho, U.S.A.*

George F. Pinder

*Department of Civil and Environmental Engineering, University of Vermont, Burlington, Vermont, U.S.A.*

---

## Abstract

We describe herein the parallel implementation of the Bi-CGSTAB method with a block Red-Black Gauss-Seidel (RBGS) preconditioner applied to the systems of linear algebraic equations that arise from the Hermite collocation discretization of partial differential equations in two spatial dimensions. The method is implemented on the Cray T3E, a parallel processing supercomputer. Speedup results are discussed.

*Key words:* Hermite collocation, Bi-CGSTAB, preconditioner, speedup

---

## 1 Introduction

We are concerned with the rapid solution of matrix equations which arise from the Hermite collocation discretization of partial differential equations (PDEs) in two spatial dimensions. This discretization is attractive for several reasons, including

---

.

.

- the freedom in how one numbers equations and unknowns, thus being able to manipulate to best advantage the structure of the resulting matrix (this point is discussed in greater detail in Section 3 below);
- the freedom in how one locates the “collocation points,” which can accelerate the rate at which convergence to the solution occurs (see [4] and [5]); and
- the spline that represents the solution has a continuous first derivative. This last property can be helpful in modeling subsurface multiphase flow problems, like those in [12].

In general, the matrix arising from the Hermite collocation discretization does not enjoy many pleasant properties of matrices arising from other popular discretizations (e.g., finite differences or finite elements). In particular, collocation matrices are not diagonally dominant, not positive definite, and not symmetric (even in the case of symmetric differential operators).

To solve these matrix equations, the two most widely used approaches are the GMRES [20] and Bi-CGSTAB [22] methods. For this work, we choose the Bi-CGSTAB method for two reasons. First, the Bi-CGSTAB method lends itself more readily to parallel processing than does GMRES. Also, there is evidence ([2], [7], [21], [23]) that Bi-CGSTAB is superior for a large class of problems. We combine Bi-CGSTAB with a Red-Black numbering system and preconditioning matrix which facilitates use of the Bi-CGSTAB method in a parallel processing mode. It was shown in [2] that BiCGSTAB/RBGS is an effective serial (i.e., not parallel) method for solving such matrix equations.

In [1], a parallel method based on fast Fourier transforms (FFTs) for solving PDEs discretized by Hermite collocation is presented. However, the FFT method is limited by requiring the same uniform mesh in both coordinate dimensions (our technique allows nonuniform meshes and the meshes in the two coordinate dimensions may be different) and the PDEs amenable to the FFT method are less general than those amenable to ours.

A multisplitting method designed for parallel processing is implemented in [18] for solving Poisson’s equation discretized by Hermite collocation. Other efforts combining parallel processing with collocation include [8] and [9], which uses quadratic, rather than cubic, basis functions; [12], which uses an alternating direction solver; and [13], where the focus is on describing a method for coarse grain partitioning applied to PDE problems.

In addition, the PIM (Parallel Iterative Methods) [10] software package for solving linear systems deserves mention. PIM requires the user to select an iterative method (choices include Bi-CGSTAB and GMRES) and then to interface his/her own code with that of PIM. The user must supply code for matrix-vector multiplication, solving preconditioned linear systems, vector in-

ner products, and computation of vector norms. For the case of GMRES, using PIM would result in substantially less effort on the part of the user to produce working code. However, the only components of Bi-CGSTAB that the user would not have to write himself/herself are routines for vector addition (and subtraction), the multiplication of a vector by a scalar, and scalar arithmetic. Since these are easily written, using PIM in conjunction with our code is hardly worthwhile.

In general, Krylov methods (like GMRES and Bi-CGSTAB) are usually found to be superior to “splitting” methods, in terms of minimizing solving time. Thus, based on evaluation of the literature, we believe our method to be the fastest, most general one available for solving collocation matrix equations, even when implemented *serially*. This last point is relevant to the discussion in Section 5.4.

This paper is organized as follows. After a brief introduction to collocation, we introduce the Red-Black numbering scheme and our preconditioning matrix. After then discussing the parallel implementation of the Bi-CGSTAB with our preconditioner, we provide speedup results. A short section summarizing our work concludes the paper.

## 2 Collocation in two spatial dimensions

In this section we provide a brief overview of collocation in two spatial dimensions. For a thorough development, see [3].

We are concerned with the numerical solution of the PDE

$$\begin{aligned} \mathcal{L}[u](x, y) = & A(x, y) \frac{\partial^2 u}{\partial x^2} + B(x, y) \frac{\partial^2 u}{\partial x \partial y} + C(x, y) \frac{\partial^2 u}{\partial y^2} \\ & + D(x, y) \frac{\partial u}{\partial x} + E(x, y) \frac{\partial u}{\partial y} + F(x, y) u = H(x, y) \end{aligned} \quad (1)$$

on the domain  $\mathcal{D} = [a_x, b_x] \times [a_y, b_y]$  with Dirichlet or Neumann boundary conditions. Upon  $\mathcal{D}$  we impose a rectilinear mesh of rectangular finite elements. Let  $m_x$  and  $m_y$  be the number of finite elements in the  $x$ - and  $y$ -directions, respectively. (Throughout this work, we assume that  $m_y$  is even.)

The collocation discretization proceeds by introducing a piecewise Hermite bi-cubic interpolating polynomial  $\bar{u}$  into (1), resulting in

$$\mathcal{L}[\bar{u}](x, y) - H(x, y) = \epsilon(x, y),$$

where  $\epsilon(x, y)$  is an error function. We then enforce

$$\epsilon(x, y) = 0 \tag{2}$$

at four distinct collocation points within the interior of each rectangular finite element. Since there are  $m_x m_y$  finite elements, we are defining  $4m_x m_y$  equations by enforcing (2) at the collocation points. In this work, we let the collocation points coincide with the points of Gaussian quadrature, a choice that, given certain smoothness conditions, minimizes truncation error [19].

The degrees of freedom (*dofs*) of  $\bar{u}$  are approximations to  $u$ ,  $\frac{\partial u}{\partial x}$ ,  $\frac{\partial u}{\partial y}$ , and  $\frac{\partial^2 u}{\partial x \partial y}$  at the  $(m_x + 1)(m_y + 1)$  *nodes* (i.e., corners of the finite elements). After introducing boundary conditions, we are left with  $4m_x m_y$  unknown degrees of freedom. Since we have  $4m_x m_y$  equations from enforcing (2) at the collocation points, we can determine the unknown coefficients.

### 3 The Red-Black ordering scheme

Collocation provides freedom in how one orders the equations and unknowns. In the literature, the most popular numbering provides a block tridiagonal structure, as implemented in [11], [16], and [17]. In this work (and previously in [2] and [6]), we permute the block tridiagonal numbering to arrive at our Red-Black ordering, depicted in Figure 1 for the case  $m_x = 5$  and  $m_y = 4$ .

In Figure 1, we have 20 ( $= m_x m_y$ ) rectangular finite elements; at each corner of each finite element is a node (depicted by a filled-in circle). At each node are four *dofs*. However, only the interior nodes are depicted as having four *dofs*. This is because each node on the boundary has fewer than four *unknown dofs* associated with it due to the imposition of boundary conditions. In particular, the boundary nodes have two unknown *dofs* each with the exception of those nodes at the four corners of the domain, which have one unknown *dof* each. Each of these unknown *dofs* is depicted by an integer (from 0 to 79 ( $= 4m_x m_y - 1$ )) inside a small square adjacent to the node with which it is associated. In addition, in the interior of each of the 20 finite elements are four collocation points, each depicted by a circle with an integer (between 0 and 79 also) inside.

With reference to Figure 1, the Red-Black ordering proceeds by numbering (in ascending order and from left to right) the bottom row of collocation points, then skipping two rows of collocation points, then resuming the numbering with the next two rows of collocation points. This pattern of skipping and then numbering pairs of rows of collocation points continues until we reach the top of the mesh where, since  $m_y$  must be even, we have a single row of collocation points remaining, which we number. We then return to the bottom

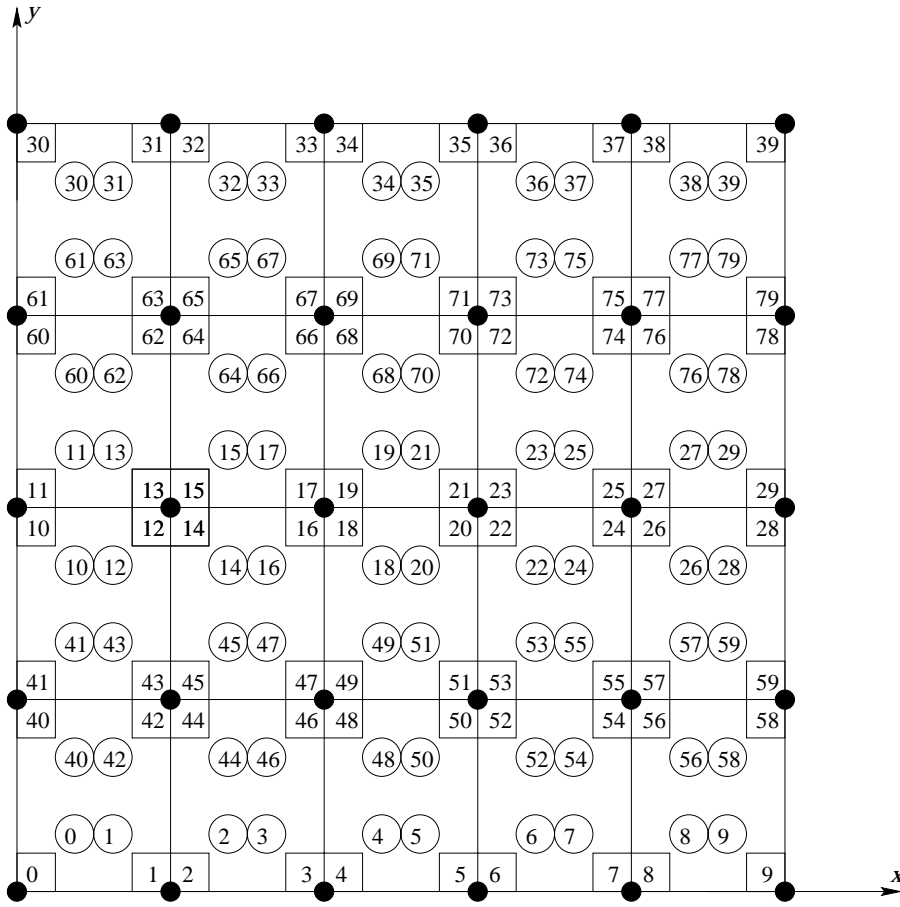


Fig. 1. Red-Black numbering of equations and unknowns for two-dimensional collocation.

of the mesh and work our way upward, numbering the previously skipped pairs of rows of collocation points as we go. The numbering of unknown *dofs* mirrors the numbering of the collocation points. The matrix structure corresponding to the Red-Black numbering is given in Figure 2 for the case  $m_x = 5$  and  $m_y = 4$ .

In general, the Red-Black numbering induces the matrix equation

$$M\mathbf{v} = \mathbf{b}, \quad (3)$$

whose block structure is

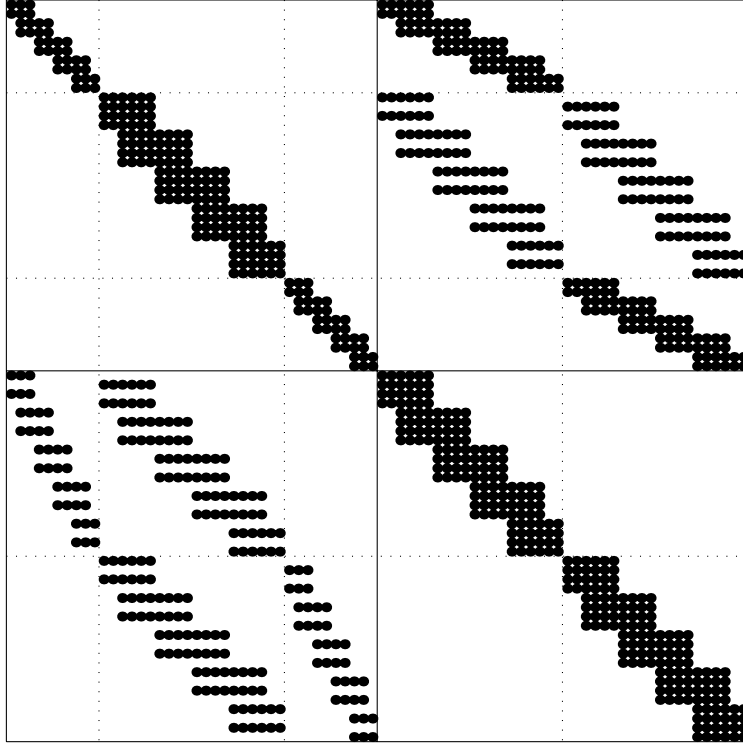


Fig. 2. Matrix structure arising from Red-Black numbering of equations and unknowns.

$$\begin{array}{c}
 \left[ \begin{array}{c|c}
 \begin{array}{cccc}
 A_F & & & \\
 & A_1 & & \\
 & & A_3 & \\
 & & \ddots & \\
 & & & A_{m_y-3} \\
 & & & & A_L
 \end{array} &
 \begin{array}{cc}
 B_F & \\
 C_0 & B_1 \\
 & C_2 & B_3 \\
 & & \ddots & \ddots \\
 & & & C_{m_y-4} & B_{m_y-3} \\
 & & & & C_L
 \end{array} \\
 \hline
 \begin{array}{cc}
 C_F & B_0 \\
 & C_1 & B_2 \\
 & & C_3 & \ddots \\
 & & & \ddots & B_{m_y-4} \\
 & & & & C_{m_y-3} & B_L
 \end{array} &
 \begin{array}{c}
 A_0 \\
 A_2 \\
 \ddots \\
 A_{m_y-4} \\
 A_{m_y-2}
 \end{array}
 \end{array} \right]
 \begin{array}{c}
 \mathbf{v}_F \\
 \mathbf{v}_1 \\
 \mathbf{v}_3 \\
 \vdots \\
 \mathbf{v}_{m_y-3} \\
 \mathbf{v}_L \\
 \hline
 \mathbf{v}_0 \\
 \mathbf{v}_2 \\
 \vdots \\
 \mathbf{v}_{m_y-4} \\
 \mathbf{v}_{m_y-2}
 \end{array}
 \end{array}$$

$$= \left[ \mathbf{b}_F^T, \mathbf{b}_1^T, \mathbf{b}_3^T, \dots, \mathbf{b}_{m_y-3}^T, \mathbf{b}_L^T \mid \mathbf{b}_0^T, \mathbf{b}_2^T, \dots, \mathbf{b}_{m_y-4}^T, \mathbf{b}_{m_y-2}^T \right]^T,$$

which we abbreviate

$$\left[ \begin{array}{c|c} R & U \\ \hline L & B \end{array} \right] \left[ \begin{array}{c} \mathbf{v}_R \\ \mathbf{v}_B \end{array} \right] = \left[ \begin{array}{c} \mathbf{b}_R \\ \mathbf{b}_B \end{array} \right]. \quad (4)$$

(The symbols  $R$  and  $B$  in (4) may, respectively, be considered to stand for “red” and “black”.) As discussed earlier, matrix  $M$  is not symmetric, not positive definite, and not diagonally dominant.

#### 4 Our preconditioning matrix

As with any Krylov method (like Bi-CGSTAB), the fastest convergence is obtained via the use of an appropriate “preconditioning” matrix  $Q$ . The matrix  $Q$  is chosen such that  $Q \approx M$  and that linear systems of the form  $Q\mathbf{y} = \mathbf{c}$  are easy to solve. For our purposes, we stipulate a third requirement:  $Q$  must have a structure amenable to parallel processing. With respect to (4), we choose our preconditioning matrix  $Q$  to be

$$Q = \left[ \begin{array}{c|c} R & \\ \hline L & B \end{array} \right]. \quad (5)$$

We note that this is the matrix corresponding to implementing a block Gauss-Seidel method to solve (3).

There are, of course, other possible choices for the preconditioning matrix  $Q$ . One is the incomplete block  $LU$  preconditioner which, as studied in [2], compares most unfavorably to  $Q$ . A block Jacobi preconditioner is another option, but would be expected to perform poorly because the matrix  $M$  is far from being block diagonally dominant.

A possible drawback of using our Red-Black numbering scheme is that the convergence of “splitting” methods (like Jacobi, Gauss-Seidel, SOR, etc.) is known to be adversely affected by such numberings [15]. However, this drawback does not apply directly to our case, since we are not using a splitting method. Rather, we are using the matrix corresponding to a splitting method (in this case, Gauss-Seidel) as a preconditioner within a Krylov method.

As is evident from Section 5.3 below, it is precisely the Red-Black numbering, combined with the block Gauss-Seidel preconditioning matrix  $Q$ , that allows us to distribute our data evenly among the various processors and to easily execute Bi-CGSTAB in parallel. While the block tridiagonal numbering of [11],

[16], and [17], discussed at the beginning of Section 3, permits the identical distribution of data among processors, the corresponding block Gauss-Seidel preconditioning matrix, lacking the Red-Black structure, does not permit simple parallelization. One could, of course, use a block Jacobi preconditioner in conjunction with the block tridiagonal ordering to achieve parallelization, but this suffers from the drawback discussed above.

## 5 Parallel implementation

In this section, we discuss the implementation of the Bi-CGSTAB method with our preconditioner on the Cray T3E, a parallel processing supercomputer. (Source code is available at [24].) After describing how each entry of  $M$ ,  $\mathbf{v}$ , and  $\mathbf{b}$  is assigned to its particular processor, we briefly discuss the issue of which parallel processing architecture should be chosen. We next detail the parallel implementation of those operations in the Bi-CGSTAB/RBGS algorithm which require interprocessor communication. (The operations in Bi-CGSTAB/RBGS that do not require interprocessor communication parallelize in a straightforward and obvious manner.) We then give speedup results.

In this work, we will assume that  $m_y$ , the number of finite elements in the  $y$ -direction, is a multiple of  $p$ , the number of processors.

### 5.1 The allocation of portions of $M$ , $\mathbf{v}$ and $\mathbf{b}$ to the various processors

We illustrate the ideas in this subsection using the particular case  $m_y = 16$  and  $p = 4$ . The general case will then be transparent.

Let the  $p = 4$  processors be denoted  $\hat{P}_1$ ,  $\bar{P}_2$ ,  $\dot{P}_3$ , and  $\check{P}_4$ . For the case where  $m_y = 16$ , we have  $\mathbf{v}$ ,  $\mathbf{b}$  and the blocks of  $M$  (i.e.,  $R$ ,  $U$ ,  $L$ , and  $B$ ) given by

$$\mathbf{v} = \left[ \hat{\mathbf{v}}_F^T, \hat{\mathbf{v}}_1^T, \bar{\mathbf{v}}_3^T, \bar{\mathbf{v}}_5^T, \dot{\mathbf{v}}_7^T, \dot{\mathbf{v}}_9^T, \check{\mathbf{v}}_{11}^T, \check{\mathbf{v}}_{13}^T, \hat{\mathbf{v}}_L \left| \hat{\mathbf{v}}_0^T, \hat{\mathbf{v}}_2^T, \bar{\mathbf{v}}_4^T, \bar{\mathbf{v}}_6^T, \dot{\mathbf{v}}_8^T, \dot{\mathbf{v}}_{10}^T, \check{\mathbf{v}}_{12}^T, \check{\mathbf{v}}_{14}^T \right. \right]^T,$$

$$\mathbf{b} = \left[ \hat{\mathbf{b}}_F^T, \hat{\mathbf{b}}_1^T, \bar{\mathbf{b}}_3^T, \bar{\mathbf{b}}_5^T, \dot{\mathbf{b}}_7^T, \dot{\mathbf{b}}_9^T, \check{\mathbf{b}}_{11}^T, \check{\mathbf{b}}_{13}^T, \hat{\mathbf{b}}_L \left| \hat{\mathbf{b}}_0^T, \hat{\mathbf{b}}_2^T, \bar{\mathbf{b}}_4^T, \bar{\mathbf{b}}_6^T, \dot{\mathbf{b}}_8^T, \dot{\mathbf{b}}_{10}^T, \check{\mathbf{b}}_{12}^T, \check{\mathbf{b}}_{14}^T \right. \right]^T,$$





### 5.3 Parallel implementation of Bi-CGSTAB/RBGS

In this section, we discuss the parallel implementation of the operations in the Bi-CGSTAB iteration that require interprocessor communication. The steps that do not require interprocessor communication are parallelized in an obvious fashion. The operations that require interprocessor communication are vector inner products; solving linear systems of the form  $Q\mathbf{y} = \mathbf{c}$ , where  $Q$  is defined in (5); and matrix-vector multiplications. We describe each of these below.

#### 5.3.1 Vector inner products

The inner (or dot) product of two vectors  $\mathbf{x}$  and  $\mathbf{y}$  is an operation that requires more interprocessor communication than is ideal for the “nearest neighbor” ring architecture. Let us assume  $\mathbf{x}$  and  $\mathbf{y}$  contain 12 elements each. The first step, which requires no interprocessor communication, is given schematically as

$$\begin{array}{l|l} \hat{P}_1 & \hat{x}_1\hat{y}_1 + \hat{x}_2\hat{y}_2 + \hat{x}_3\hat{y}_3 \rightarrow \hat{\sigma}_1 \\ \bar{P}_2 & \bar{x}_4\bar{y}_4 + \bar{x}_5\bar{y}_5 + \bar{x}_6\bar{y}_6 \rightarrow \bar{\sigma}_2 \\ \dot{P}_3 & \dot{x}_7\dot{y}_7 + \dot{x}_8\dot{y}_8 + \dot{x}_9\dot{y}_9 \rightarrow \dot{\sigma}_3 \\ \check{P}_4 & \check{x}_{10}\check{y}_{10} + \check{x}_{11}\check{y}_{11} + \check{x}_{12}\check{y}_{12} \rightarrow \check{\sigma}_4 \end{array}$$

At this point, each processor  $P_i$  knows the value of its corresponding  $\sigma_i$ . However, to complete the computation of the inner product, it is necessary for each processor to know the value of  $\sum_{i=1}^4 \sigma_i$ . The process by which this summation occurs, an example of a “reduction” operation, requires each processor to exchange information with processors other than its nearest neighbors.

However, in our code we utilize a “shared memory” routine available on the Cray T3E called “shmem\_double\_sum\_to\_all” which performs the necessary “reduction” operations very rapidly. Thus the interprocessor communication required by the computation of vector inner products is performed efficiently. (On systems without this “shared memory” feature, communication times would commensurately increase, making the speedup curve in Figure 3 flatter.)

#### 5.3.2 Solving linear systems of the form $Q\mathbf{y} = \mathbf{c}$

The goal is to solve the linear system

$$Q\mathbf{y} = \mathbf{c},$$





send  $\bar{\mathbf{y}}_3$  to processor  $\hat{P}_1$ . So, as part of the  $Q\mathbf{y} = \mathbf{c}$  operation, each processor must both send and receive a portion of a vector. Note that this interprocessor communication occurs only between nearest neighbors in the ring architecture.

In addition, since the matrices  $A_i$  in the subsystems (7) and (9) are block tridiagonal, efficient code may be written for the solution of each. For  $i = F$  or  $L$ , the blocks are  $2 \times 2$  matrices. For  $i = 0, 1, 2, \dots, m_y - 2$ , the blocks are  $4 \times 4$  matrices.

### 5.3.3 Matrix-vector multiplication

During the iteration process in the Bi-CGSTAB algorithm, we note that every step of the form

$$Q\mathbf{y} = \mathbf{c}$$

is immediately followed by a matrix-vector multiplication step of the form

$$\mathbf{v} = M\mathbf{y}.$$

Because of the manner in which  $Q$  was selected, we see that we do not have to perform the complete matrix-vector multiplication  $\mathbf{v} = M\mathbf{y}$ :

$$\begin{aligned} \mathbf{v} = M\mathbf{y} &= \left[ \begin{array}{c|c} R & U \\ \hline L & B \end{array} \right] \left[ \begin{array}{c} \mathbf{y}_R \\ \mathbf{y}_B \end{array} \right] \\ &= \left[ \begin{array}{c|c} R & \\ \hline L & B \end{array} \right] \left[ \begin{array}{c} \mathbf{y}_R \\ \mathbf{y}_B \end{array} \right] + \left[ \begin{array}{c|c} & U \\ \hline & \end{array} \right] \left[ \begin{array}{c} \mathbf{y}_R \\ \mathbf{y}_B \end{array} \right] \\ &= Q\mathbf{y} + \left[ \begin{array}{c} U\mathbf{y}_B \\ \hline \end{array} \right] \\ &= \mathbf{c} + \left[ \begin{array}{c} U\mathbf{y}_B \\ \hline \end{array} \right] \\ &= \left[ \begin{array}{c} \mathbf{c}_R + U\mathbf{y}_B \\ \hline \mathbf{c}_B \end{array} \right]. \end{aligned}$$



processor must both send and receive a portion of a vector and that this interprocessor communication occurs only between nearest neighbors in the ring architecture.

#### 5.4 Implementation on the Cray T3E

We implemented Bi-CGSTAB/RBGS on a Cray T3E, a supercomputer that can take advantage of the parallelism inherent in the method. The T3E has address space for 2048 processors configured on a three-dimensional torus architecture. Further details concerning the technical specifications of the T3E may be found in [25].

We illustrate below that we obtain reasonably good speedups for sufficiently large problem size. We note that the times reported below include only the “main loop” portion of the Bi-CGSTAB/RBGS algorithm which is, of course, where the bulk of the computations occur.

Let  $T_S$  = serial run time = time required to solve a problem using a single processor. Specifically, the algorithm used here is the one-processor version of the algorithm described above, the code for which does not include any features specific to the T3E or Cray computers in general. In addition, as discussed in Section 1, this is the fastest most general serial method known. Let  $T_P$  = parallel run time = time required to solve a problem using parallel processing. *Speedup* is defined as

$$S = \text{speedup} = \frac{T_S}{T_P}.$$

Ideally, one would like  $S = p$ , a phenomenon known as *linear speedup*. However, in practice, one observes only sub-linear speedup due to *parallel overhead* (i.e., manifestations of parallel algorithms that are absent in serial algorithms), the main sources of which are [14]:

- interprocessor communication,
- load imbalance, and
- extra computation.

We saw in our discussion above that interprocessor communication occurs in three separate operations of the Bi-CGSTAB/RBGS iteration: solving linear systems of the form  $Q\mathbf{y} = \mathbf{c}$ , matrix-vector multiplication, and in the reduction portion of vector inner product computation. Fortunately, the interprocessor communication is the only significant source contributing to sub-linear

speedup. The load is balanced as well as possible (i.e., processors do not sit idle waiting for messages from other processors before they can resume performing useful computations) and although extra computation does occur, these redundancies always occur *simultaneously* across all the processors.

When testing our code on the Cray T3E, we selected a particular PDE with Dirichlet boundary conditions and allowed the number of finite elements in the  $x$ -direction ( $m_x$ ) and the number of finite elements in the  $y$ -direction ( $m_y$ ) to (independently) assume the values 16, 32, 64, and 128. We ran the code using  $p = 1, 2, 4, 8, 16,$  and 32 processors. For each parameter set  $\{m_x, m_y, p\}$ , we ran the code five times and then took the mean of these five times as the representative time used to compile the results given in Table 1.

Table 1  
Speedup results

| $m_x$ | $m_y$ | $p = 1$ | $p = 2$ | $p = 4$ | $p = 8$ | $p = 16$ | $p = 32$ |
|-------|-------|---------|---------|---------|---------|----------|----------|
| 16    | 16    | 1.0000  | 1.2106  | 1.3818  |         |          |          |
| 16    | 32    | 1.0000  | 1.3881  | 1.8314  | 2.1219  |          |          |
| 16    | 64    | 1.0000  | 1.6372  | 2.2701  | 3.1482  | 3.7461   |          |
| 16    | 128   | 1.0000  | 1.7782  | 2.8277  | 4.3781  | 5.5691   | 6.4388   |
| 32    | 16    | 1.0000  | 1.4578  | 1.8974  |         |          |          |
| 32    | 32    | 1.0000  | 1.5980  | 2.3621  | 3.0986  |          |          |
| 32    | 64    | 1.0000  | 1.6408  | 2.7593  | 4.4253  | 5.5811   |          |
| 32    | 128   | 1.0000  | 1.8731  | 3.3643  | 5.6086  | 8.4241   | 11.3382  |
| 64    | 16    | 1.0000  | 1.5828  | 2.3070  |         |          |          |
| 64    | 32    | 1.0000  | 1.7641  | 2.9244  | 4.2942  |          |          |
| 64    | 64    | 1.0000  | 1.8707  | 3.4074  | 5.6325  | 8.2351   |          |
| 64    | 128   | 1.0000  | 1.9059  | 3.3788  | 6.5883  | 10.1703  | 16.1655  |
| 128   | 16    | 1.0000  | 1.7507  | 2.8698  |         |          |          |
| 128   | 32    | 1.0000  | 1.8780  | 3.4103  | 5.6460  |          |          |
| 128   | 64    | 1.0000  | 1.9383  | 3.6523  | 6.6972  | 11.1015  |          |
| 128   | 128   | 1.0000  | 1.9717  | 3.8835  | 7.3626  | 12.8075  | 22.0124  |

As can be seen in Table 1, we achieve only modest speedup for relatively small problem sizes. We achieve the best results for the largest problem, i.e., when  $m_x = m_y = 128$ , which corresponds to solving a linear system of 65536 equations in 65536 unknowns. For this example, i.e., when  $m_x = m_y = 128$ , we depict the results graphically in Figure 3, where we see that our speedup curve

stays remarkably close to the ideal linear speedup curve for values of  $p$  up to 8. For larger values of  $p$ , the speedup curve flattens out somewhat. This can be explained by the fact that in our implementation, the ring structure of the processors as described above need not have explicitly occurred on the T3E. That is, there was no guarantee that direct connections existed between processors that needed to communicate with each other. Therefore, communication may have been “bottlenecked” by one processor performing computations while another processor was waiting for the first to pass it a message. Or even more simply, messages may have passed through intermediary processors on their way to their final destination. Another possible explanation for the flattening of the speedup curve is that the amount of work each processor does relative to communication cost decreases as the number of processors increases.

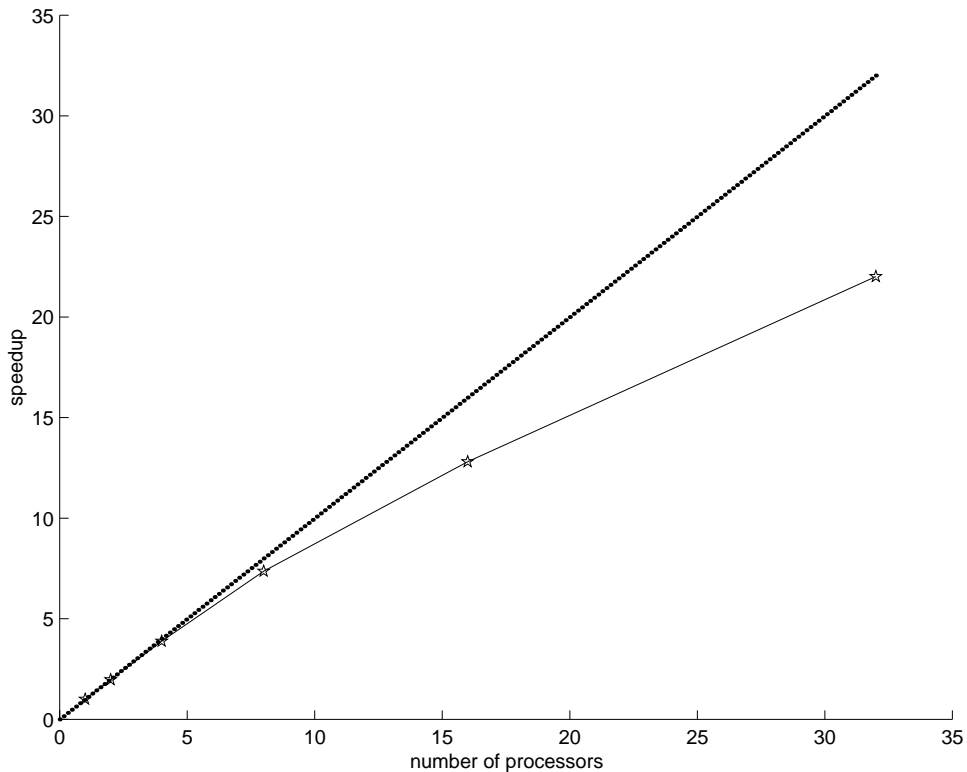


Fig. 3. Speedup curve for  $m_x = m_y = 128$ . The upper curve represents the unattainable ideal of linear speedup.

## 6 Summary

Given the Hermite collocation discretization of a PDE in two spatial dimensions, we utilize a Red-Black numbering scheme and corresponding preconditioning matrix. These permit the corresponding matrix equation to be rapidly solved in a parallel processing mode by the Bi-CGSTAB method. For sufficiently large problems, we achieve almost linear speedup if the number of pro-

processors is 8 or less. When a greater number of processors is used, the speedup curve flattens out somewhat. We believe that this could be ameliorated by ensuring that the ring architecture favored by our method be embedded upon the architecture of the parallel processing computer being used.

## Acknowledgments

The authors gratefully acknowledge the contribution of Dr. Ilene Carpenter of Cray Research/SGI, for providing access to the T3E supercomputer and for her magnanimous assistance during the numerical experiments. We also thank the referees for their insightful comments and suggestions.

## References

- [1] B. Bialecki, G. Fairweather, and K. R. Bennett, Fast Direct Solvers for Piecewise Hermite Bicubic Orthogonal Spline Collocation Equations, *SIAM J. Numer. Anal.* **29** (1992) 156–173.
- [2] S. H. Brill, Hermite Collocation Solution of Partial Differential Equations via Preconditioned Krylov Methods, *Numerical Methods for Partial Differential Equations* **17** (2001) 120–136.
- [3] S. H. Brill, *Documentation for Collocation Software*, RCGRD Publication #98-01, Research Center for Groundwater Remediation Design, University of Vermont, Burlington, Vermont, 1998. Available at the World Wide Web site [http://math.boisestate.edu/~brill/research/doc\\_coll.html](http://math.boisestate.edu/~brill/research/doc_coll.html).
- [4] S. H. Brill and G. F. Pinder, Analysis of a Block Red-Black Preconditioner Applied to the Hermite Collocation Discretization of a Model Parabolic Equation, *Numerical Methods for Partial Differential Equations*, in press.
- [5] S. H. Brill and G. F. Pinder, Eigenvalue Analysis of a Block Red-Black Gauss-Seidel Preconditioner Applied to the Hermite Collocation Discretization of Poisson's Equation, *Numerical Methods for Partial Differential Equations* **17** (2001) 204–228.
- [6] S. H. Brill and G. F. Pinder, A Parallel Preconditioned Bi-Conjugate Gradient Algorithm for Two-Dimensional Elliptic and Parabolic Equations Using Hermite Collocation, *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing* (1997).
- [7] P. Chin and P. A. Forsyth, A Comparison of GMRES and CGSTAB Accelerations for Incompressible Navier-Stokes Problems, *Journal of Computational and Applied Mathematics* **46** (1993) 415–426.

- [8] C. C. Christara, Quadratic Spline Collocation Methods for Elliptic Partial Differential Equations, *BIT* **34** (1998) 33–61.
- [9] C. C. Christara, E. N. Houstis, and J. R. Rice, A Parallel Spline Collocation-Capacitance Method for Elliptic Partial Differential Equations, *Supercomputing Conference Proceedings* (1988) 375–384.
- [10] R. D. da Cunha and T. Hopkins, *Parallel Iterative Methods*. Available at the World Wide Web site <http://www.mat.ufrgs.br/~rudnei/pim/pim-i.html>.
- [11] E. O. Frind and G. F. Pinder, A Collocation Finite Element Method for Potential Problems in Irregular Domains, *Int. J. Numer. Meth. Eng.* **14** (1979) 681–701.
- [12] J. F. Guarnaccia and G. F. Pinder, A Collocation Based Parallel Algorithm to Solve Immiscible Two Phase Flow in Porous Media, *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, (1992) 205–210.
- [13] C. E. Houstis, E. N. Houstis, and J. R. Rice, Partitioning PDE Computations: Methods and Performance Evaluation, *Parallel Computing* **5** (1987) 141–163.
- [14] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing* (Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994).
- [15] C.-C. J. Kuo and T. F. Chan, Two-Color Fourier Analysis of Iterative Algorithms for Elliptic Problems with Red/Black Ordering, *SIAM J. Sci. Stat. Comput.* **11** (1990) 767–793.
- [16] Y.-L. Lai, A. Hadjidimos, E. N. Houstis, and J. R. Rice, On the Iterative Solution of Hermite Collocation Equations, *SIAM J. Matrix Anal. Appl.* **16** (1995) 254–277.
- [17] T. S. Papatheodorou, Block AOR Iteration for Nonsymmetric Matrices, *Math. Comp.* **41** (1983), 511–525.
- [18] T. S. Papatheodorou and Y. G. Saridakis, Parallel Algorithms and Architectures for Multisplitting Iterative Methods, *Parallel Computing* **12** (1989) 171–182.
- [19] P. M. Prenter and R. D. Russell, Orthogonal Collocation for Elliptic Partial Differential Equations, *SIAM J. Numer. Anal.* **13** (1976) 923–939.
- [20] Y. Saad and M. H. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.* **7** (1986) 856–869.
- [21] W. Schmid, M. Paffrath, and R. H. W. Hoppe, Application of Iterative Methods for Solving Nonsymmetric Linear Systems in the Simulation of Semiconductor Processing, *Surveys on Mathematics for Industry* **5** (1995) 1–26.
- [22] H. A. van der Vorst, Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.* **13** (1992) 631–644.

[23] C. Vulk, Further Experiences with GMRESR, *Supercomputer* **10** (1993) 13–27.

[24] World Wide Web site:

[http://www.math.boisestate.edu/~brill/research/cray\\_code.html](http://www.math.boisestate.edu/~brill/research/cray_code.html).

[25] World Wide Web site:

<http://www.tera.com//products/systems/crayt3e/paper1.html>.