Computers Math. Applic. Vol. 24, No. 5/6, pp. 83–101, 1992
Printed in Great Britain.

0097-4943/92 $5.00 + 0.00

COMPARISON OF PERFORMANCE OF THREE PARALLEL VERSIONS OF THE BLOCK CYCLIC REDUCTION ALGORITHM FOR SOLVING LINEAR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

JUNG-SING JWO, S. LAKSHMIVARAHAN AND SUDARSHAN K. DHALL
Parallel Processing Institute
School of Electrical Engineering and Computer Science
University of Oklahoma, Norman, OK 73019, U.S.A.
JOHN M. LEWIS
National Severe Storms Laboratory, (NOAA)
Norman, OK 73069, U.S.A.

Abstract—This paper provides an experimental comparison of three versions of the block cyclic reduction (BCR) algorithm based on (a) polynomial factorization, (b) partial fraction expansion, and (c) rational approximations. Each of these versions is implemented using (a) the vector-oriented LU decomposition method, and (b) the scalar cyclic reduction method. It is found that BCR using the scalar cyclic reduction performs better than that using the vector-oriented LU decomposition. All our timings are based on the Alliant FX/8 at Argonne National Laboratory.

1. INTRODUCTION

Analysis of mathematical models of physical processes in Science and Engineering naturally leads to the problem of solving partial differential equations [1]. In this paper, our interest stems from a class of problems in meteorology [2–4], namely *data assimilation* in numerical weather prediction using the so called *adjoint* method [5]. In particular, we are interested in numerically solving a class of linear elliptic partial differential equation which constitutes an important component of the data assimilation process using the adjoint method [5].

With the upsurge in new observing systems for mesoscale meteorology such as the next generation radar network (NEXRAD) and the wind profile demonstration network, there has been an increased emphasis on the assimilation of these data into the numerical prediction models. One of the most promising strategies is the adjoint method [4,6]. This adjoint method is a strategy for finding the "optimal" initial state of the prediction model that minimizes a functional say, the mean square difference between the model forcast and the corresponding data collected during the model evolution. The gradient of this functional with respect to the initial state is found by an iterative procedure that involves (a) the forward integration of the model equations over the time period where the observations are available, and (b) backward integration of the adjoint equations (of complexity similar to the forward model). Given this gradient, a new approximation to the optimal state is obtained by using one of many optimum seeking methods. This process is repeated until a satisfactory estimate of the optimal initial state is obtained. As the models are made more complex, it is reasonable to expect that the number of iterations will increase.

This is the first of a series of reports devoted to exploring parallelism in Data Assimilation problems. This research was supported in part by a grant from the "Center for the Analysis and Prediction of Storms" (CAPS) which is an NSF Science and Technology center recently established at the University of Oklahoma.

Our thanks are due to Dr. Swarztrauber (NCAR) who very liberally shared his ideas for calculating the rational approximations given in the appendix with us. We are also thankful to Dr. J. Cody (Argonne National Laboratory) for helping us to understand the intricacies of Chebyshev approximation, and Dr. E. Gallopoulos (CSRD, University of Illinois) for sharing his experience with this class of problems. Finally, we wish to record an appreciation for the Argonne National Laboratory for granting permission to use their Alliant machine.

This is particularly true for the mesoscale models that include water substance conversion and associated latent heat release.

At the Center for the Analysis and Prediction of Storms (CAPS) at the University of Oklahoma, a major effort is underway to implement the adjoint method of data assimilation on a variety of meteorological models. We are currently working on two models: the so called barotropic model that is still used to forcast the large scale winds associated with hurricane tracking, and a non-hydrostatic dry convection model that is used to forcast winds in the vicinity of smaller scale fronts such as dry lines. In both of these models, solution of elliptic equations of the Poisson's type is required at each incremental time step, and is traditionally a bottleneck for even the forward models. With the added complexity of data assimilations and the associated backward integration which also involves solution of elliptic equations, it is imperative that efficient solution of elliptic type partial differential equations be found to make the data assimilation problem tractable in the operational environment.

Over the past two decades enormous progress has been made in solving discrete versions of the linear elliptic partial differential equations using *direct* methods [7–10]. For definiteness, consider the standard Poisson's equation in two dimension, say over a rectangle. The discrete version of this equation resulting from a finite difference scheme (using a five point stencil) on an $n \times m$ grid takes the form

$$\mathbf{A}\,\mathbf{x} = \mathbf{y}, \tag{1.1}$$

where the non-zero elements of the $m \times m$ block tridiagonal matrix $\mathbf{A}$ are themselves $n \times n$ matrices [1]. A class of direct techniques that is particularly amenable for parallel and vector computing is based on an extension of sort of the now classical Gaussian elimination, and is known as the *odd-even reduction* or *cyclic reduction*. This technique was first introduced in 1965 by Hockney [11] in collaboration with Golub. There are at least two ways in which this method can be applied to solve (1.1). One possibility is to first *decouple* (1.1) into a collection of n independent $m \times m$ (scalar) linear tridiagonal systems by applying the *discrete Fourier transform* to (1.1) and then solve the resulting scalar tridiagonal systems using the cyclic reduction in parallel [12,13]. This was the original approach by Hockney and has come to be known as the fast Poisson solver [11,14]. In the second approach, apply the method of cyclic reduction rather directly to the block tridiagonal system, as was first done by Buzbee, Golub and Nielson [7]. This is called the *block cyclic reduction (BCR)* method. Buneman [15] was the first to expose the instability of this method and presented a stable version of this algorithm. Since then, this method has been extended in a number of different directions—to grids with arbitrary number of points in [16–19], to separable elliptic partial differential equations by Swarztrauber [20], to problems with different boundary conditions and boundary shapes [7], to mention a few. The relation between the cyclic reduction and the LU decomposition is examined in [21], and the effect of diagonal dominance in [14,21]. For a succinct summary of these and other related results, refer to [22–29].

Despite these theoretical advances, the practical applicability the block cyclic reduction was severely limited by the *serial bottleneck* arising from the subproblem of solving linear systems with a (Chebyshev type) matrix polynomial. While these polynomials can readily be factored, the real slow down is due to the fact that the linear system involving each of the linear factors of the above polynomial must be solved in a sequence. For later reference, we call this approach BCR with *polynomial factorization*. To remedy this bottleneck, recently two approaches were proposed. One approach is to express the inverse of the product form of the polynomial matrix as a sum of the inverses of linear factors using the *partial fraction* expansion. This approach called the *partial fraction* method was independently developed by Sweet [30], and Gallopoulos and Saad [31]. In fact, Gallopoulos and Saad [31] demonstrate the superiority of the BCR with partial fraction compared to BCR with polynomial factorization, by presenting an experimental comparison of the performance of these techniques on the Alliant multivector processor.

As an alternative, Swarztrauber [20] showed that the inverse of the polynomial matrix that arise in the BCR method can be replaced by rational approximation of considerably low degree. As an example, he showed that the inverse of the (Chebyshev) polynomial of degree 128 (with respect to a 256×256 grid) can be replaced by a rational approximation of degree 10 with error of the order

of 10^{-15}. While the use of this approximation holds great promise to speed up the BCR method, we know not of any experimental comparison of the BCR with rational approximation with the two other methods. In this paper, we provide a comprehensive comparison of the performance of BCR with rational approximation, partial fraction and polynomial factorization on the Alliant multivector processor. In particular, we present a trade-off between the speed-up and accuracy resulting from variation of the degree of the approximating rational polynomial.

In Section 2, we present the Buneman's algorithm followed by the discussion of (a) the polynomial factorization, (b) partial fraction expansion and (c) the rational approximation. Various comments relating to the implementation and the comparison of the performance are presented in Section 3. An example of the computation of the rational approximation is given in the Appendix A.

2. THE BLOCK CYCLIC REDUCTION (BCR) ALGORITHM

Let $\psi = \psi(x, y)$ be the solution of the linear elliptic partial differential equation of the form

$$a(x)\frac{\partial^2\psi}{\partial x^2} + b(x)\frac{\partial\psi}{\partial x} + c(x)\psi + \frac{\partial^2\psi}{\partial y^2} = f(x, y), \tag{2.1}$$

where $a(\cdot)$, $b(\cdot)$, $c(\cdot)$ and $f(\cdot,\cdot)$ are given. By applying the standard finite difference approximation to this equation with *Dirichlet* boundary condition over a rectangular area (using the standard five-point stencil), we obtain a linear system of the form

$$\mathbf{A}\,\mathbf{x} = \mathbf{y}, \tag{2.2}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{I} & 0 & \cdots & \cdots & \cdots & 0 \\ \mathbf{I} & \mathbf{B} & \mathbf{I} & \ddots & & & \vdots \\ 0 & \mathbf{I} & \mathbf{B} & \mathbf{I} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \ddots & \mathbf{B} & \mathbf{I} \\ 0 & \cdots & \cdots & \cdots & 0 & \mathbf{I} & \mathbf{B} \end{bmatrix} \tag{2.3}$$

is an $m \times m$ block tridiagonal matrix, $\mathbf{B}$ is an $n \times n$ scalar tridiagonal matrix, $\mathbf{I}$ is a unit matrix of size n, $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)^T$ and $\mathbf{y} = \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m)^T$, where $\mathbf{x}_i$ and $\mathbf{y}_i$ are $n \times 1$ column vectors, for $1 \le i \le m$.

For simplicity, let $m = 2^{k+1} - 1$, for some integer $k \ge 0$. We first present a stable version of the BCR known as the Buneman's algorithm for solving $\mathbf{A}\,\mathbf{x} = \mathbf{y}$. For a derivation of this algorithm, refer to [7,14,26].

BCR ALGORITHM: BUNEMAN'S VERSION.

(a) Initialization Phase:

$q^{(0)} = 2^{k+1} - 1, \quad \mathbf{B}^{(0)} = \mathbf{B}, \quad \mathbf{x}_0 = \mathbf{x}_{m+1} = 0.$

For $r = 1$ **to** k **step** 1 **do**

$\quad \mathbf{u}_0^{(r)} = \mathbf{u}_{m+1}^{(r)} = 0$

$\quad \mathbf{v}_0^{(r)} = \mathbf{v}_{m+1}^{(r)} = 0$

End

For $j = 1$ **to** m **step** 1 **do**

$\quad \mathbf{v}_j^{(0)} = \mathbf{y}_j$

End

(b) Reduction Phase:

$$
\begin{aligned}
&q^{(1)} = \left[\frac{q^{(0)}-1}{2}\right]; \quad \mathbf{B}^{(1)} = 2\mathbf{I} - [\mathbf{B}^{(0)}]^2 \\
&\textbf{For } j = 2 \textbf{ to } 2 \times q^{(1)} \textbf{ step } 2 \textbf{ do} \\
&\quad \textbf{Solve } \mathbf{B}\mathbf{u}_j^{(1)} = \mathbf{y}_j \\
&\quad \mathbf{v}_j^{(1)} = \mathbf{y}_{j+1} + \mathbf{y}_{j-1} - 2\mathbf{u}_j^{(1)} \\
&\textbf{End} \\
&\textbf{For } r = 1 \textbf{ to } k-1 \textbf{ step } 1 \textbf{ do} \\
&\quad q^{(r+1)} = \left[\frac{q^{(r)}-1}{2}\right]; \quad \mathbf{B}^{r+1} = 2\mathbf{I} - [\mathbf{B}^{(r)}]^2 \\
&\quad \textbf{For } j = 2^{r+1} \textbf{ to } 2^{r+1} \times q^{(r+1)} \textbf{ step } 2^{r+1} \textbf{ do} \\
&\quad\quad \textbf{Solve } \mathbf{B}^{(r)} f = \mathbf{u}_{j+2^r}^{(r)} + \mathbf{u}_{j-2^r}^{(r)} - \mathbf{v}_j^{(r)} \\
&\quad\quad \mathbf{u}_j^{(r+1)} = \mathbf{u}_j^{(r)} - f \\
&\quad\quad \mathbf{v}_j^{(r+1)} = \mathbf{v}_{j+2^r}^{(r)} + \mathbf{v}_{j-2^r}^{(r)} - 2\mathbf{u}_j^{(r+1)} \\
&\quad \textbf{End} \\
&\textbf{End}
\end{aligned}
\tag{2.4}
\tag{2.5}
$$

(c) Solve the k^{th} Case:

$$
\begin{aligned}
&\textbf{Solve } \mathbf{B}^{(k)} f = \mathbf{v}_{2^k}{}^{(k)} \\
&\mathbf{x}_{2^k} = f + \mathbf{u}_{2^k}{}^{(k)}
\end{aligned}
$$

(d) Back Substitution Phase:

$$
\begin{aligned}
&\textbf{For } r = k \textbf{ to } 1 \textbf{ step } -1 \textbf{ do} \\
&\quad \textbf{For } j = 2^{r-1} \textbf{ to } 2^{r-1} \times q^{(r-1)} \textbf{ step } 2^r \textbf{ do} \\
&\quad\quad \textbf{Solve } \mathbf{B}^{(r-1)} f = \mathbf{v}_j^{(r-1)} - (\mathbf{x}_{j+2^{r-1}} + \mathbf{x}_{j-2^{r-1}}) \\
&\quad\quad \mathbf{x}_j = f + \mathbf{u}_j^{(r-1)} \\
&\quad \textbf{End} \\
&\textbf{End}
\end{aligned}
\tag{2.6}
$$

Since the mathematical properties of this algorithm relating to operation count, stability, etc., are well understood [7,13,32,33], in the following, we confine our attention to the computational aspects of the algorithm. The key features of this algorithm are:

(a) It requires solution of the linear system of the form

$$\mathbf{B}^{(r)}\mathbf{z} = \mathbf{d} \tag{2.7}$$

both in the reduction phase and in the back substitution phase (refer to (2.4) to (2.6)).

(b) $\mathbf{B}^{(r)}$ is an $n \times n$ matrix polynomial of degree 2^r. Thus, the *degree* of the polynomial *increases* by a factor of two after each step of the reduction phase.

(c) The number of linear systems of the type (2.7) to be solved in a given step also changes from step to step. Referring to (2.4), in the first step of the reduction phase (namely, when $r = 0$) there are 2^k systems to be solved, and when $r = i$ for $1 \leq i \leq k-2$, from (2.5), it is clear that there are 2^{k-i} systems to be solved. Finally, when $r = k-1$, there is only one system of the type $\mathbf{B}^{(k)}\mathbf{z} = \mathbf{d}$ to be solved. In other words, the *number* of linear

systems to be solved *decreases* by a factor of two after each step of the reduction phase. During the back substitution phase, on the other hand, (refer to (2.6)), the number of linear systems of the type (2.7) increases by a factor of two. Another interesting feature that is computationally important is that the systems at a given level differ only in the right-hand side; refer to (2.4) through (2.6).

From the computational point of view, there are at least two compelling reasons for *not* calculating $\mathbf{B}^{(r)}$ explicitly. First, the tridiagonal structure of $\mathbf{B}$ is destroyed when the powers of $\mathbf{B}$ are computed. Thus, the bandwidth of $\mathbf{B}^{(r)}$ increases with r. Second, the *cost* (measured in terms of elapsed time) of computing $\mathbf{B}^{(r)}$ involves matrix multiplication and will outweigh the gains resulting from the use of BCR. In the following subsections, we describe three strategies for solving $\mathbf{B}^{(r)}\mathbf{z} = \mathbf{d}$ without actually computing $\mathbf{B}^{(r)}$.

2.1. Polynomial Factorization

Recall that $\mathbf{B}^{(r)}$ is a polynomial of degree 2^r. It can be shown [7,26] that $\mathbf{B}^{(r)}$ can be factored as follows:

$$\mathbf{B}^{(r)} = -\prod_{j=1}^{2^r} \mathbf{G}_j^{(r)}, \tag{2.8}$$

where

$$\mathbf{G}_j^{(r)} = \mathbf{B} + 2\mathbf{I}\cos\theta_j^{(r)}, \tag{2.9}$$

and

$$\theta_j^{(r)} = \frac{2j-1}{2^{r+1}}\pi, \qquad \text{for } 1 \le j \le 2^r. \tag{2.10}$$

Given this factorization, we can solve (2.7) using the following algorithm.

POLYNOMIAL FACTORIZATION ALGORITHM.

$$\begin{aligned}
&\mathbf{z}_0 = -\mathbf{d}\\
&\texttt{For } j = 1 \texttt{ to } 2^r \texttt{ step 1 do}\\
&\qquad \texttt{Solve } \mathbf{G}_j^{(r)}\mathbf{z}_j = \mathbf{z}_{j-1}\\
&\texttt{End}\\
&\mathbf{z} = \mathbf{z}_{2^r}.
\end{aligned} \tag{2.11}$$

The advantages of this approach are that

(a) the factors $\mathbf{G}_j^{(r)}$ of $\mathbf{B}^{(r)}$ are known *a priori* and are considered as input to the algorithm, and
(b) the linear factors $\mathbf{G}_j^{(r)}$ are tridiagonal matrices.

In other words, there is no additional cost involving factorization of $\mathbf{B}^{(r)}$ and the tridiagonal structure is preserved. The major disadvantage, however, is that the collection of 2^r linear tridiagonal systems must be solved in a sequence. In the literature, this is often referred to as the *serial bottleneck*. There are at least two known methods for breaking this bottleneck which are described below.

2.2. Partial Fraction Expansion

One approach to alleviate the serial bottleneck is to express the inverse of $\mathbf{B}^{(r)}$ as the sum of the inverses of linear factors using the partial fraction expansion [30,31]. It can be shown [30,31] that

$$[\mathbf{B}^{(r)}]^{-1} = \sum_{j=1}^{2^r} \frac{\alpha_j}{\mathbf{B} + 2\mathbf{I}\cos\theta_j^{(r)}}, \tag{2.12}$$

where

$$\alpha_j = \frac{(-1)^{j+1}}{2^r} \sin\frac{2j-1}{2^{r+1}}\pi. \tag{2.13}$$

This immediately leads to the following algorithm.

PARTIAL FRACTION ALGORITHM.

$$\begin{aligned} &\textbf{For } j = 1 \textbf{ to } 2^r \textbf{ step } 1 \textbf{ do } \textit{in parallel} \\ &\qquad \textbf{Solve } [\mathbf{B} + 2\mathbf{I}\cos\theta_j^{(r)}]\,\mathbf{z}_j = \mathbf{d} \\ &\textbf{End} \end{aligned} \tag{2.14}$$

$$\textbf{Compute } \mathbf{z} = \sum_{j=1}^{2^r} \alpha_j \mathbf{z}_j \textbf{ using the Associative Fan-in Algorithm [26].} \tag{2.15}$$

The advantages of this method are

(a) the tridiagonal structure is preserved, and
(b) there is *parallelism* both at the level of finding $\mathbf{z}_j$ using (2.14), and at the level of computing $\mathbf{z}$ using (2.15).

This algorithm can be mapped [26] onto a variety of parallel architectures to exploit the parallelism in (2.14) and (2.15).

2.3. Rational Approximation

The idea of replacing $[\mathbf{B}^{(r)}]^{-1}$ by a suitable rational approximation to speed up the BCR algorithm was first introduced by Swarztrauber [34]. For completeness, we begin by presenting the salient features of his approach. The (scalar) polynomial $p_{2^r}(x)$ corresponding to $\mathbf{B}^{(r)}$ may be *recursively* defined by

$$p_{2^r}(x) = 2 - [p_{2^{r-1}}(x)]^2, \tag{2.16}$$

where $p_1(x) = x$. When (2.1) corresponds to the standard Poisson's equation (that is, when $a(x) \equiv 1$, $b(x) \equiv 0$ and $c(x) \equiv 0$ in (2.1)), it can be shown that $\mathbf{B}$ is of the form

$$\mathbf{B} = \begin{bmatrix} -4 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 1 & -4 & 1 & \ddots & & & \vdots \\ 0 & 1 & -4 & 1 & \ddots & & \vdots \\ \vdots & \ddots & 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \ddots & -4 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & -4 \end{bmatrix}. \tag{2.17}$$

Since the eigenvalues [7] of this matrix are given by

$$\lambda_j = -4 + 2\cos\left(\frac{j\pi}{n+1}\right), \tag{2.18}$$

for $1 \le j \le n$, we seek a *good* rational approximation to $p_{2^r}(x)$ for $x \in [-6, -2]$. A particularly useful representation of $p_{2^r}(x)$ is obtained by changing variables. Let

$$x = -2\cosh\phi. \tag{2.19}$$

Substituting (2.19) in (2.16), it can be shown that

$$p_n(x) = -2\cosh\left[n\cosh\left(\frac{-x}{2}\right)\right], \tag{2.20}$$

for $|x| \ge 2$. Now define

$$q_n(x) = -\frac{1}{2}\,p_n\left(-2 - \frac{x}{n^2}\right). \tag{2.21}$$

Using the fact that

$$\cosh^{-1}(x) = \log\left[x + (x^2 - 1)^{1/2}\right], \tag{2.22}$$

it can be shown [34] that

$$\lim_{n\to\infty} q_n(x) = \cosh\sqrt{x}. \tag{2.23}$$

In other words, $q_n^{-1}(x)$ for large n *behaves* like sech $\sqrt{x}$. Thus, we can derive approximations to $q_n^{-1}(x)$ by suitably approximating sech $\sqrt{x}$. Instead, Swarztrauber has demonstrated that rational approximations to $p_n^{-1}(x)$ can be obtained rather directly [34].

Let

$$R_{s,t}(x) = \delta\,\frac{x^s + a_1\,x^{s-1} + \cdots + a_{s-1}\,x + a_s}{x^t + b_1\,x^{t-1} + \cdots + b_{t-1}\,x + b_t} \tag{2.24}$$

be the approximating rational function to $p_n^{-1}(x)$. In [34], it has been shown that

$$\max_{-6\le x\le -2} |p_{210}^{-1}(x) - R_{10,10}(x)| \le 10^{-15}. \tag{2.25}$$

Thus, even for moderately large n, $p_n^{-1}(x)$ can be approximated to a very high degree of accuracy by a comparatively very low degree rational function.

From the computational point of view, it is better to consider $R_{s,t}(x)$ in the factored form. Let

$$R_{s,t}(x) = \delta\,\frac{\prod_{i=1}^{t}(x - \alpha_i)}{\prod_{i=1}^{t}(x - \beta_i)}. \tag{2.26}$$

Then, the solution of (2.7) may be written in the form

$$\begin{aligned}\mathbf{z} &= [\mathbf{B}^{(r)}]^{-1}\,\mathbf{d}\\ &= \delta\prod_{i=1}^{t}\frac{\mathbf{B} - \alpha_i\,\mathbf{I}}{\mathbf{B} - \beta_i\,\mathbf{I}}\,\mathbf{d}.\end{aligned} \tag{2.27}$$

Let $\mathbf{z}_0 = \delta\,\mathbf{d}$ and define

$$\mathbf{z}_i = \frac{\mathbf{B} - \alpha_i\,\mathbf{I}}{\mathbf{B} - \beta_i\,\mathbf{I}}\,\mathbf{z}_{i-1}. \tag{2.28}$$

If $h = \mathbf{z}_i - \mathbf{z}_{i-1}$, then (2.28) is equivalent to

$$(\mathbf{B} - \beta_i\,\mathbf{I})\,h = (\beta_i - \alpha_i)\,\mathbf{z}_{i-1}. \tag{2.29}$$

Thus,

$$\mathbf{z}_i = h + \mathbf{z}_{i-1} \tag{2.30}$$

can be obtained by solving (2.29) and $\mathbf{z} = \mathbf{z}_t$. A method for computing $R_{s,t}(x)$ in the form (2.26) is given in Appendix A.

RATIONAL APPROXIMATION ALGORITHM.

$\mathbf{z}_0 = \delta\,\mathbf{d}$
For $i = 1$ **to** t **step** 1 **do**
  Solve $(\mathbf{B} - \beta_i\,\mathbf{I})\,h = (\beta_i - \alpha_i)\,\mathbf{z}_{i-1}$
  $\mathbf{z}_i = h + \mathbf{z}_{i-1}$
End
$\mathbf{z} = \mathbf{z}_t$

A number of comments are in order. In converting $R_{s,t}(x)$ from (2.24) to the product form in (2.27), it is likely that some of the α_i's and/or β_j's are complex. To avoid solving linear systems with complex matrices, it is suggested that the linear factors corresponding to complex conjugate roots be combined to obtain quadratic terms. Thus, if $\alpha = \alpha_1 + i\alpha_2$, and $\overline{\alpha} = \alpha_1 - i\alpha_2$, then

$$(\mathbf{B} - \alpha)\,(\mathbf{B} - \overline{\alpha}) = \mathbf{B}^2 - 2\alpha_1\,\mathbf{B} + (\alpha_1^2 + \alpha_2^2)\,\mathbf{I} \tag{2.31}$$

is a pentadiagonal matrix. This involves one matrix multiplication and matrix additions, but instead of solving two complex tridiagonal systems, there is only one pentadiagonal system to be solved.

Table 3.1. BCR with polynomial factorization—Vector oriented LU decomposition.

Dimension	No. of Processors	Time (sec.)	Error = ($\|\|Ax - y\|\|_\infty$)
31	Serial	7.6709747E-02	1.023181539494544E-012
	1	5.5840015E-02	
	2	5.2150249E-02	
	4	5.0030231E-02	
	8	4.8510075E-02	
63	Serial	0.3509808	6.139089236967266E-012
	1	0.2297001	
	2	0.2172489	
	4	0.2091217	
	8	0.2094498	
127	Serial	1.609932	3.637978807091713E-011
	1	0.9815369	
	2	0.9045563	
	4	0.8764496	
	8	0.8602219	
255	Serial	7.363403	3.419700078666210E-010
	1	4.386017	
	2	3.890839	
	4	3.714813	
	8	3.672821	

Table 3.2. BCR with polynomial factorization—Scalar cyclic reduction method.

Dimension	No. of Processors	Time (sec.)	Error = ($\|\|Ax - y\|\|_\infty$)
31	Serial	9.1949940E-02	7.389644451905042E-013
	1	3.9510250E-02	
	2	2.9850006E-02	
	4	2.2550106E-02	
	8	2.0579815E-02	
63	Serial	0.4368305	4.547473508864641E-012
	1	0.1408901	
	2	9.3919754E-02	
	4	6.8649292E-02	
	8	6.0409546E-02	
127	Serial	2.082527	2.910383045673370E-011
	1	0.5535660	
	2	0.3343735	
	4	0.2357635	
	8	0.2055359	
255	Serial	9.927917	2.728484105318785E-010
	1	2.499542	
	2	1.424835	
	4	0.9917297	
	8	0.8098755	

Table 3.3. BCR with partial fraction expansion—Vector oriented LU decomposition.

Dimension	No. of Processors	Time (sec.)	Error = ($\|\|Ax - y\|\|_\infty$)
31	Serial	0.1188798	1.080024958355352E-012
	1	7.5910091E-02	
	2	4.2530060E-02	
	4	2.7940273E-02	
	8	2.3409843E-02	
63	Serial	0.5652103	5.002220859751105E-012
	1	0.3189106	
	2	0.1663799	
	4	0.1021690	
	8	7.6589584E-02	
127	Serial	2.667580	5.366018740460277E-011
	1	1.370216	
	2	0.6843643	
	4	0.3879700	
	8	0.2577515	
255	Serial	12.51776	4.329194780439138E-010
	1	6.082245	
	2	3.174835	
	4	1.815979	
	8	1.220642	

Table 3.4. BCR with partial fraction expansion—Scalar cyclic reduction method.

Dimension	No. of Processors	Time (sec.)	Error = ($\|\|Ax - y\|\|_\infty$)
31	Serial	0.1120601	1.477928890381008E-012
	1	4.9199581E-02	
	2	2.9150009E-02	
	4	1.8290043E-02	
	8	1.5850067E-02	
63	Serial	0.5447693	6.366462912410498E-012
	1	0.1808586	
	2	9.6559525E-02	
	4	5.6119919E-02	
	8	3.7309647E-02	
127	Serial	2.616165	7.730704965069890E-011
	1	0.7491913	
	2	0.3830795	
	4	0.2245178	
	8	0.1436462	
255	Serial	12.64664	4.220055416226387E-010
	1	3.516144	
	2	1.839844	
	4	1.075256	
	8	0.717316	

Table 3.5. BCR with rational approximation—First degree vector oriented LU decomposition.

Dimension	No. of Processors	Time (sec.)	Relative Error	Error = $(\|\|Ax - y\|\|_\infty)$
31	Serial	4.6500206E-02	3.6538050E-02	8.37737484699664
	1	2.4350166E-02		
	2	2.0259857E-02		
	4	1.8400192E-02		
	8	1.6609669E-02		
63	Serial	0.1977196	5.5996761E-02	35.3071230900935
	1	7.8090668E-02		
	2	6.1441422E-02		
	4	5.2928925E-02		
	8	4.8700333E-02		
127	Serial	0.8389359	6.6736691E-02	181.121938609601
	1	0.2987518		
	2	0.2152405		
	4	0.1664276		
	8	0.1463776		
255	Serial	3.595306	1.0985650E-01	857.322034442546
	1	1.269897		
	2	0.832855		
	4	0.635071		
	8	0.548767		

Table 3.6. BCR with rational approximation—First degree scalar cyclic reduction method.

Dimension	No. of Processors	Time (sec.)	Relative Error	Error = $(\|\|Ax - y\|\|_\infty)$
31	Serial	5.1330090E-02	1.653923496734399E-002	8.37737484699608
	1	2.3880005E-02		
	2	1.5260220E-02		
	4	1.1170387E-02		
	8	8.5000992E-03		
63	Serial	0.2198696	5.599366776255554E-002	35.3071230900964
	1	8.0810547E-02		
	2	4.4570923E-02		
	4	2.7618408E-02		
	8	1.9380569E-02		
127	Serial	0.9303513	6.673814210144235E-002	181.121938609600
	1	0.2931061		
	2	0.1688080		
	4	0.1037292		
	8	8.0039978E-02		
255	Serial	4.025299	1.113111420839860E-001	857.322034442663
	1	1.277588		
	2	0.736267		
	4	0.471771		
	8	0.389496		

Table 3.7. BCR with rational approximation—Second degree oriented LU decomposition.

Dimension	No. of Processors	Time (sec.)	Relative Error	Error = $(\|\|Ax - y\|\|_\infty)$
31	Serial	9.8279953E-02	1.0731510E-03	0.210292313195168
	1	6.0539722E-02		
	2	4.4500351E-02		
	4	3.9180279E-02		
	8	3.7549973E-02		
63	Serial	0.4012814	1.4821142E-03	0.598620732175959
	1	0.1796894		
	2	0.1291294		
	4	0.1048794		
	8	9.2941284E-02		
127	Serial	1.652122	1.4770838E-03	2.59783823573980
	1	0.571373		
	2	0.381493		
	4	0.285019		
	8	0.254684		
255	Serial	5.016022	3.2048693E-03	15.2010356384671
	1	1.664612		
	2	1.084686		
	4	0.835602		
	8	0.708038		

Table 3.8. BCR with rational approximation—Third degree oriented LU decomposition.

Dimension	No. of Processors	Time (sec.)	Relative Error	Error = $(\|\|Ax - y\|\|_\infty)$
31	Serial	8.7059975E-02	5.4993325E-05	8.246075572628797E-003
	1	6.1620235E-02		
	2	4.7489643E-02		
	4	4.5899868E-02		
	8	4.1369915E-02		
63	Serial	0.3489304	6.2682673E-05	2.591201017867206E-002
	1	0.1863308		
	2	0.1499004		
	4	0.1194096		
	8	0.1108398		
127	Serial	1.405479	1.0562982E-04	6.801820681812387E-002
	1	0.6178818		
	2	0.4205322		
	4	0.3401260		
	8	0.3036194		
255	Serial	5.771973	3.1735256E-04	626.923060752088
	1	2.097961		
	2	1.423553		
	4	1.098816		
	8	0.964356		

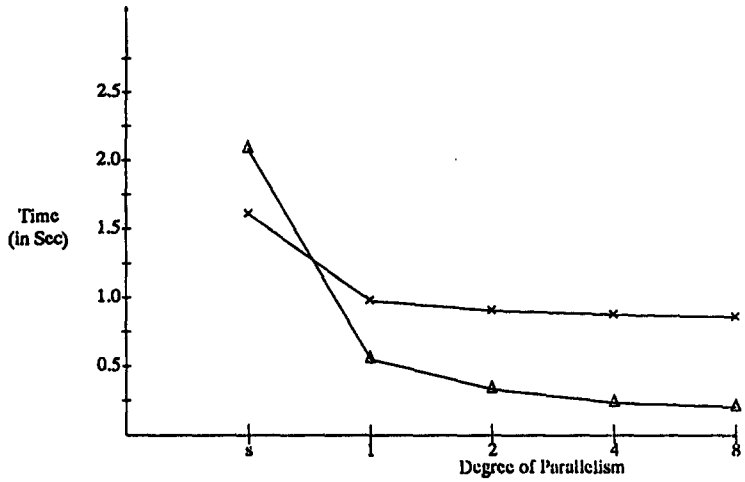

Figure 3.1. BCR with polynomial factorization; ×—LU decomposition, Δ—Scalar CR, $n = 127$.

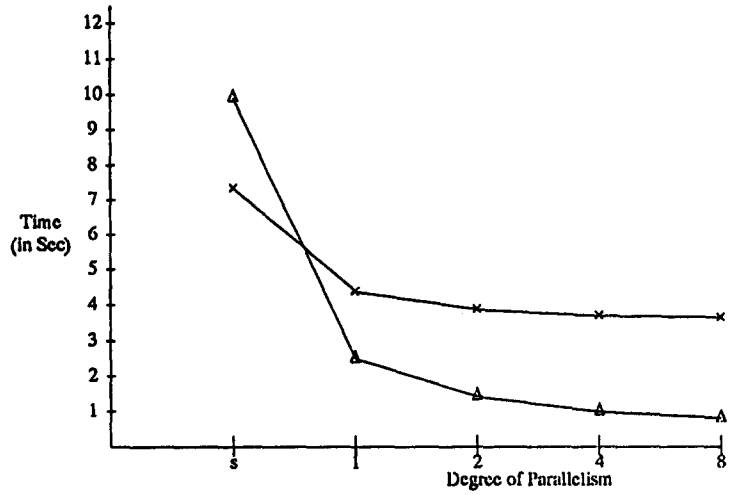

Figure 3.2. BCR with polynomial factorization; ×—LU decomposition, Δ—Scalar CR, $n = 255$.

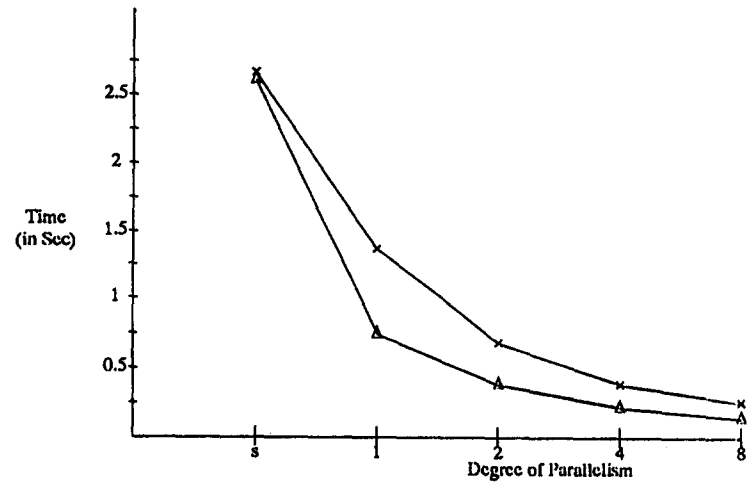

Figure 3.3. BCR with partial fraction expansion; ×—LU decomposition, Δ—Scalar CR, $n = 127$.

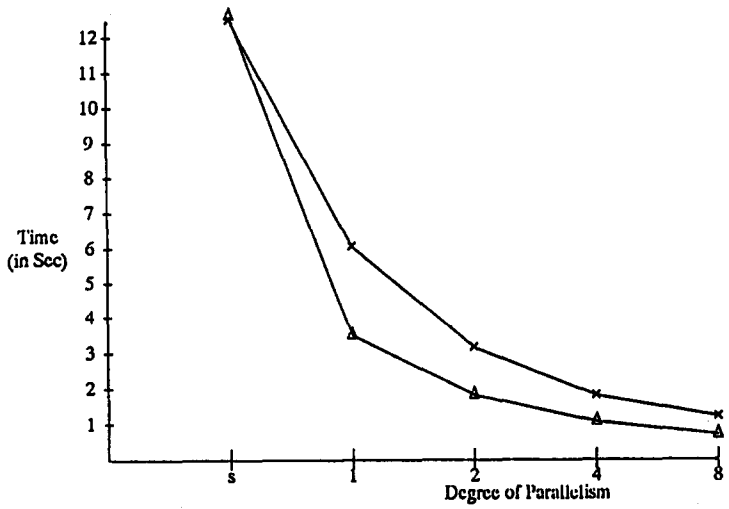

Figure 3.4. BCR with partial fraction expansion; ×—LU decomposition, Δ—Scalar CR, $n = 255$.

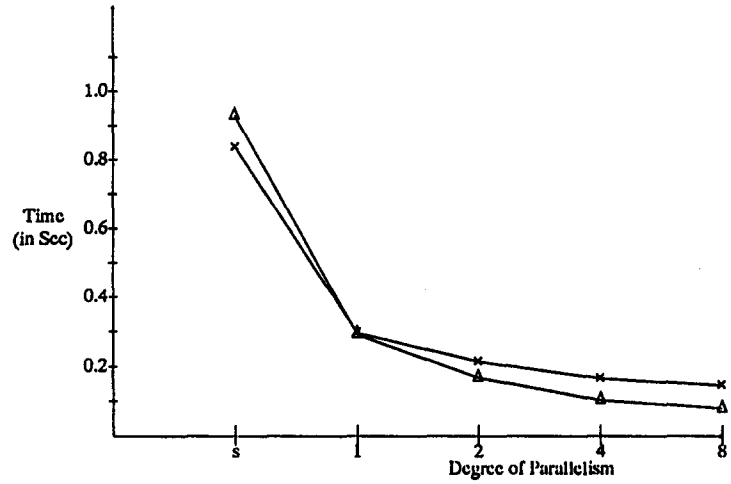

Figure 3.5. BCR with rational approximation—First degree; ×—LU decomposition, Δ—Scalar CR, $n = 127$.

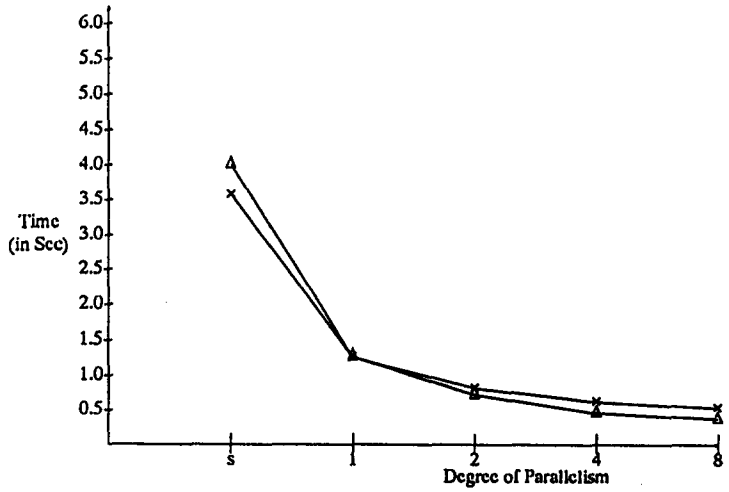

Figure 3.6. BCR with rational approximation—First degree; ×—LU decomposition, Δ—Scalar CR, $n = 255$.

3. EXPERIMENTAL COMPARISONS

For purposes of experimental comparisons, we chose, without loss of generality, the standard Poisson's equation, (that is, in (2.1), $a(x) \equiv 1$, $b(x) \equiv 0$ and $c(x) \equiv 0$). In this case, $\mathbf{B}$ is an $n \times n$ matrix of the form given in (2.17).

A distinguishing feature of the BCR algorithm is that there are one or more linear tridiagonal systems to be solved during each step of the reduction/back-substitution phase, and that each of these linear systems have the same matrix but different right-hand sides. As an example, from (2.4), it is clear that the first step of the reduction phase involves systems of the type

$$\mathbf{B}\,\mathbf{u}_j^{(1)} = \mathbf{y}_j, \tag{3.1}$$

for $j = 2, 4, 6, \ldots, 2^{k+1} - 2$. These systems can be combined to obtain

$$\mathbf{B}\,\mathbf{X} = \mathbf{Y}, \tag{3.2}$$

where $\mathbf{X}$ is a matrix formed by the vectors $\mathbf{u}_j^{(1)}$, i.e., $\mathbf{X} = [\mathbf{u}_2^{(1)}, \mathbf{u}_4^{(1)}, \mathbf{u}_6^{(1)}, \ldots]$ and $\mathbf{Y}$ is a matrix of the type $\mathbf{Y} = [\mathbf{y}_2, \mathbf{y}_4, \mathbf{y}_6, \ldots]$. This latter system can be solved by *two* different methods:

(a) the vector oriented LU-decomposition [35,36], and
(b) scalar version of the cyclic reduction algorithm [14,26]. (Also refer to Section 2.)

For completeness, the vector-oriented LU-decomposition algorithm is presented in Appendix B.

Each of the *three* algorithms—BCR with polynomial factorization, partial fraction expansion and rational approximation—were run in *two modes*—one using the vector oriented LU decomposition and the second using the scalar cyclic reduction. In the case of rational approximations, we also vary the degree of the approximating rational function. All the experiments were conducted on the Alliant FX/8 (at the University of Oklahoma and at the Argonne National Laboratory) in *double precision,* using block tridiagonal matrices with $n = m = 2^{k+1} - 1$, for $k = 4, 5, 6$ and 7. For a given value of n, the algorithms were run in serial mode, and p vector processors for $p = 1, 2, 4$ and 8. The timings and the error (measured in terms of the L_1-norm of the vector $\mathbf{A}\,\mathbf{x} - \mathbf{y}$, where $\mathbf{x}$ is the computed solution) are tabulated for various combinations of the algorithms.

From Tables 3.1 and 3.2, it is clear that the BCR with polynomial factorization using the scalar cyclic reduction is *faster* while maintaining the same level of accuracy. Similar observations follow from Tables 3.3 and 3.4 for BCR with partial fraction expansion.

The performance of the BCR with the first degree rational approximation is contained in Tables 3.5 and 3.6, and those with second and third degree are given in Tables 3.7 and 3.8, respectively. In this case, in addition to the absolute error, the relative error in the solution is also given. From Tables 3.5, 3.7 and 3.8, it is clear that the accuracy in terms of the relative error *improves* with the degree of the rational approximation at the cost of *increased* elapsed time. Comparing Tables 3.5 and 3.6, we again see that the BCR with the scalar cyclic reduction algorithm performs better for larger number of processors. For example, when $n = 255$ using $p = 8$ processors, it takes 0.54877 seconds for vector oriented LU decomposition (Table 3.5) compared to 0.3895 seconds for the scalar cyclic reduction (Table 3.6).

Figures 3.1 through 3.6 compare the reduction in the elapsed time with the increase in parallelism. From Figures 3.1 to 3.4, it is clear that with the increasing parallelism, the scalar cyclic reduction based implementation of BCR with polynomial factorization and partial fraction expansion performs better than those based on LU decomposition. Also, when BCR is used with the first degree rational approximation, from Figures 3.5 and 3.6 it follows that the scalar cyclic reduction based algorithm performs better.

REFERENCES

1. D.A. Anderson, J.C. Tannehill and R.H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corporation, New York, (1984).
2. M. DeMaria, Tropical cyclone track prediction with a baratropic spectral model, *Monthly Weather Review* 115, 2346–2357 (October 1987).

3. J. Sun and D.W. Flicker, Recovery of three dimensional wind and temperature fields from single-Doppler radar data, (Preprints) School of Meteorology, University of Oklahoma, Norman, Oklahoma, (November 1989).
4. O. Talagrand and P. Courtier, Variational assimilation of meteorological and observations with the adjoint vorticity equation—I: Theory, *Quarterly Journal of Royal Meteorological Society* **113**, 1311–1328 (1987).
5. W.C. Thacker, Large least squares problems and the need for automating the generation of adjoint code, In *Computational Solution of Non-Linear Systems of Equations*, Lecture Notes in Applied Mathematics, American Mathematical Society, Providence, RI, (1989).
6. W.C. Thacker and R.B. Long, Fitting dynamics to data, *Journal of Geophysical Research* **93**, 1277–1240 (1988).
7. B.L. Buzbee, G.H. Golub and C.W. Nielson, On direct methods for solving Poisson's equations, *SIAM Journal on Numerical Analysis* **7** (4), 627–656 (December 1970).
8. S.L. Johnsson, Fast PDE solvers on fine and medium grain architectures, YALEU/DCS/TR-583, Yale University, (April 1987).
9. M. Machura and R.A. Sweet, A survey of software for partial differential equations, *ACM Transactions on Mathematical Software* **6** (4), 461–488 (December 1980).
10. P.N. Swarztrauber and R.A. Sweet, Algorithm 541. Efficient Fortran subprograms for the solution of separable elliptic partial differential equations, *ACM Transactions on Mathematical Software* **5** (3), 352–364 (September 1979).
11. R.W. Hockney, A fast direct solution of Poisson's equation using Fourier analysis, *Journal of ACM* **12**, 95–113 (1965).
12. B.L. Buzbee, A fast Poisson solver amenable to parallel computation, *IEEE Transactions on Computers* **C-22** (8), 793–796 (August 1973).
13. C. Temperton, On the FACR(l) algorithm for the discrete Poisson equation, *Journal of Computational Physics* **34**, 312–329 (1980).
14. R.W. Hockney and C. Jesshope, *Parallel Computers: Architecture, Programming and Algorithms*, Adam Hilger, Bristol, (1981).
15. O. Buneman, A compact non-iterative Poisson solver, Report 294, Stanford University, Institute for Plasma Research, Stanford, CA, (1969).
16. U. Schumann and R.A. Sweet, A direct method for the solution of Poisson's equation with Neumann boundary conditions on a staggered grid of arbitrary size, *Journal of Computational Physics* **20**, 171–182 (1976).
17. R.A. Sweet, Direct methods for the solution of Poisson's equation on a staggered grid, *Journal of Computational Physics* **12**, 422–428 (1973).
18. R.A. Sweet, A generalized cyclic reduction algorithm, *SIAM Journal on Numerical Analysis* **11** (3), 506–520 (June 1974).
19. R.A. Sweet, A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension, *SIAM Journal on Numerical Analysis* **14** (4), 706–720 (September 1977).
20. P.N. Swarztrauber, A direct method for the discrete solution of separable elliptic equations, *SIAM Journal on Numerical Analysis* **11** (6), 1136–1150 (December 1974).
21. D. Heller, Some aspects of cyclic reduction algorithm for block tridiagonal systems, *SIAM Journal on Numerical Analysis* **13**, 484–496 (1976).
22. B.L. Buzbee and F.W. Dorr, The direct solution of the biharmonic equation on rectangular regions and the Poisson equation, *SIAM Journal on Numerical Analysis* **11** (4), 753–763 (September 1974).
23. E. Detyna, Point cyclic reductions for elliptic boundary-value problems—I. The constant-coefficient case, *Journal of Computational Physics* **33**, 204–216 (1979).
24. F.W. Dorr, The direct solution of the discrete Poisson equation in $O(N^2)$ operations, *SIAM Review* **17** (3), 412–414 (July 1975).
25. K. Dowers, S. Lakshmivarahan, S.K. Dhall and J.C. Diaz, Block tridiagonal system on Alliant FX/8, In *Parallel Processing for Scientific Computing*, (Edited by G. Rodrigue), pp. 56–60, SIAM Publications, Philadelphia, PA, (1989).
26. S. Lakshmivarahan and S.K. Dhall, *Analysis and Design of Parallel Algorithms*, (Chapter 5), McGraw-Hill, New York, (1990).
27. N.K. Madsen and G.H. Rodrigue, Odd-even reduction for pentadiagonal matrices, In *Parallel Computers—Parallel Mathematics*, (Edited by M. Feilmeier), pp. 103–106, International Association for Mathematics and Computers in Simulation, New York, NY, (1977).
28. P.N. Swarztrauber, The direct solution of the discrete Poisson equation on the surface of a sphere, *Journal of Computational Physics* **15**, 46–54 (1974).
29. P.N. Swarztrauber, The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle, *SIAM Review* **19** (3), 490–501 (July 1977).
30. R.A. Sweet, A parallel and vector variant of the cyclic reduction algorithm, *SIAM Journal on Scientific and Statistical Computing* **9**, 762–765 (1988).
31. E. Gallopoulos and Y. Saad, A parallel block cyclic reduction algorithm for the fast solution of elliptic equations, *Parallel Computing* **10**, 143–159 (1989).
32. I. Fujishiro, Y. Ikebe, A. Harashima and M. Watanabe, A note on cyclic reduction Poisson solvers with application to bioconvective phenomena problems, *Computers and Fluids* **17** (3), 419–435 (1989).

33. P.N. Swarztrauber and R.A. Sweet, Vector and parallel methods for direct solution of Poisson's equations, *Journal of Computational and Applied Mathematics* **27**, 241–263 (1989).
34. P.N. Swarztrauber, Approximate cyclic reduction for solving Poisson's equations, *SIAM Journal on Scientific and Statistical Computing* **8**, 199–209 (1987).
35. G.H. Golub and C.F. Van Loan, *Matrix Computations*, 2nd edn., The Johns Hopkins University Press, (1989).
36. J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, (1988).
37. W.J. Cody, W. Fraser and J.F. Hart, Rational Chebyshev approximation using linear equations, *Numerische Mathematik* **12**, 242–251 (1968).
38. C.B. Moler and G.W. Stewart, An algorithm for generalized matrix eigenvalue problems, *SIAM Journal on Numerical Analysis* **10**, 241–256 (1973).

APPENDIX A
RATIONAL APPROXIMATION TO THE INVERSE OF CHEBYSHEV POLYNOMIAL OF THE FIRST KIND

This appendix describes a method due to Swarztrauber for computing the rational approximation of small degree to the rational function $p_{2^r}^{-1}(x)$. The polynomial $p_{2^r}(x)$ is defined recursively as

$$p_{2^{r+1}}(x) = 2 - (p_{2^r}(x))^2, \tag{A1}$$

where $p_1(x) = x$. We seek a uniformly "good" (in the min-max sense) approximation to $p_{2^r}(x)$ for $x \in [-6, -2]$. As a first step, we state some crucial properties of this class of polynomials:

1. By setting $x = -2\cosh\phi$, for $|x| \geq 2$, we obtain
$$p_n(x) = -2\cosh\left[n\cosh\left(\frac{-x}{2}\right)\right]. \tag{A2}$$
2. $-p_n(x)$ is called the Chebyshev polynomial of the first kind.
3. Values of $p_n^{-1}(x)$ for $n = 127$ are given in Table A1. It can be verified that $p_n^{-1}(x)$ has a very large derivative in the interval of interest.

Normally, one would invoke to standard theory of Chebyshev approximation to compute the rational approximation. Indeed, application of the Remes-type algorithm [37] to obtain such an approximation leads to non-convergence, either due to a sequence of critical points that are not monotonic, or the value of the error is too large indicating the presence of poles. To get around this difficulty, Swarztrauber developed a method based on the generalized eigenvalue problem [38] which we now describe.

Let

$$-6 < w_1 < w_2 < w_3 < w_4 < -2 \tag{A3}$$

be a set of preselected points, and let

$$z_i = p_{2^k}(w_i), \tag{A4}$$

where $m = 2^{k+1} - 1$. We illustrate the basic idea of this approach using a first degree rational approximation to z_i. Extension to higher degree rational approximation is straightforward. For $1 \leq i \leq 4$, let

$$z_i - \frac{a + b\,w_i}{c + d\,w_i} = (-1)^{i-1}\,\eta. \tag{A5}$$

Multiplying both sides by $(c + d\,w_i)$, we can express (A5) succinctly in the matrix form, as

$$\mathbf{A}\,\mathbf{x} = \eta\,\mathbf{B}\,\mathbf{x}, \tag{A6}$$

where

$$\mathbf{x} = (a, b, c, d)$$

$$\mathbf{A} = \begin{bmatrix} -1 & -w_1 & z_1 & z_1 w_1 \\ -1 & -w_2 & z_2 & z_2 w_2 \\ -1 & -w_3 & z_3 & z_3 w_3 \\ -1 & -w_4 & z_4 & z_4 w_4 \end{bmatrix}, \tag{A7}$$

and

$$\mathbf{B} = \begin{bmatrix} -0 & -0 & 1 & w_1 \\ -0 & -0 & -1 & -w_2 \\ -0 & -0 & 1 & w_3 \\ -0 & -0 & -1 & -w_4 \end{bmatrix}. \tag{A8}$$

The method then computes the eigenvector $\mathbf{x}$ corresponding to the minimum eigenvalue η using an algorithm due to Moler and Stewart [38] available in the EISPACK Library. Using the computed values of the coefficients a, b, c, and d, the algorithm then computes a new set of w_i's at which the absolute value of the error is maximum and the process is repeated. This iterative refinement process is quite akin to the Remes-type approach.

Rewriting (refer to 2.26)

$$R_{11}(x) = \frac{a + x}{c + d\,x} = \delta\,\frac{x - \alpha}{x - \beta},$$

values of α, β and δ for various values of n are given in Table A2.

Table A1. Comparison of the exact and approximate values of $p_{127}^{-1}(x)$. Maximum error = 0.01509716.

x	Exact value of $p_n^{-1}(x)$	Approximate value of $p_n^{-1}(x)$
−6.00	−0.59562592E − 97	0.15097157E − 01
−5.75	−0.18690324E − 94	0.15096300E − 01
−5.50	−0.78513577E − 92	0.15095322E − 01
−5.25	−0.45655171E − 89	0.15094193E − 01
−5.00	−0.38243857E − 86	0.15092876E − 01
−4.75	−0.48432148E − 83	0.15091320E − 01
−4.50	−0.98397579E − 80	0.15089451E − 01
−4.25	−0.34544559E − 76	0.15087168E − 01
−4.00	−0.23049286E − 72	0.15084314E − 01
−3.75	−0.33131709E − 68	0.15080645E − 01
−3.50	−0.12171732E − 63	0.15075753E − 01
−3.25	−0.14586099E − 58	0.15068904E − 01
−3.00	−0.82630731E − 53	0.15058631E − 01
−2.75	−0.41043027E − 46	0.15041511E − 01
−2.50	−0.58774619E − 38	0.15007272E − 01
−2.25	−0.50324108E − 27	0.14904587E − 01
−2.00	−0.50000000E + 00	−0.49862999E + 00

Table A2. Coefficients of the approximating first degree rational function.

n	α	β	δ
1	−9.6344	−1.5034	0.03272
3	−4.3877	−1.8917	0.02311
7	−2.7586	−1.9748	0.01706
15	−2.2043	−1.9938	0.01558
31	−2.0521	−1.9985	0.01522
63	−2.0131	−1.9996	0.01528
127	−2.0033	−1.9999	0.01511
255	−2.0008	−1.9999	0.01510
512	−2.0002	−1.9999	0.01509
1024	−2.0000	−1.9999	0.01509

APPENDIX B

VECTOR ORIENTED LU DECOMPOSITION

Let **A** be an $n \times n$ matrix and consider the linear system

$$\mathbf{A}\,\mathbf{x} = \mathbf{d}. \tag{B1}$$

The standard *Gaussian* elimination consists in decomposing $\mathbf{A} = \mathbf{L}\,\mathbf{U}$ where **L** and **U** are lower and upper triangular matrices, and replacing (B1) with

$$\mathbf{L}\,\mathbf{y} = \mathbf{d} \tag{B2}$$

and

$$\mathbf{U}\,\mathbf{x} = \mathbf{y}. \tag{B3}$$

Obtaining (B3) from (B1) is known as the forward elimination, and solving (B3) is called the backward substitution. However, in situations where the factor $\mathbf{L}$ is *not* explicitly needed, we can directly obtain (B3) from (B1) by using a sequence of transformations called *Gaussian* transformations. Let $\mathbf{A}^{(1)} = \mathbf{A}$ and $\mathbf{d}^{(1)} = \mathbf{d}$. Define the $n \times n$ matrix

$$\mathbf{M}_1 = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ -m_{21} & 1 & 0 & \dots & 0 \\ -m_{31} & 0 & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ -m_{n1} & 0 & \dots & \dots & 1 \end{bmatrix}, \tag{B4}$$

where

$$m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \qquad \text{for } i \geq 2. \tag{B5}$$

If $m_1 = (0, m_{21}, m_{31}, \dots, m_{n1})^{\mathsf{T}}$ and e_i is the i^{th} unit (column) vector, then it can be verified that

$$\mathbf{M}_1 = \mathbf{I}_n - m_1 e_1^{\mathsf{T}}, \tag{B6}$$

where $\mathbf{I}_n$ is the unit matrix of order n. Multiplying both sides of (B1) by $\mathbf{M}_1$, we obtain

$$\mathbf{A}^{(2)} = \mathbf{d}^{(2)}, \tag{B7}$$

where

$$\mathbf{A}^{(2)} = \mathbf{M}_1\,\mathbf{A}^{(1)} = [a_{ij}^{(2)}] \tag{B8}$$

is such that $a_{i1}^{(2)} = 0$ for $i \geq 2$, and

$$\mathbf{d}^{(2)} = \mathbf{M}_1\,\mathbf{d}^{(1)}. \tag{B9}$$

Similarly, define

$$\begin{aligned} \mathbf{A}^{(k+1)} &= \mathbf{M}_k\,\mathbf{A}^{(k)}, \text{ and} \\ \mathbf{d}^{(k+1)} &= \mathbf{M}_k\,\mathbf{d}^{(k)} \end{aligned} \tag{B10}$$

where

$$\mathbf{M}_k = \mathbf{I}_n - m_k\,e_k^{\mathsf{T}} \tag{B11}$$

$$m_k = (0, 0, \dots, 0, m_{k+1,k}, \dots, m_{nk})^{\mathsf{T}}, \tag{B12}$$

and

$$m_{jk} = \frac{a_{jk}^{(k)}}{a_{kk}^{(k)}}, \qquad \text{for } j \geq k+1.$$

It can be verified that all the elements of the first k columns of $\mathbf{A}^{(k+1)}$ that are below the principal diagonal are zero. Continuing in this way, we obtain

$$\mathbf{A}^{(n)}\,\mathbf{x} = \mathbf{d}^{(n)}, \tag{B13}$$

where $\mathbf{A}^{(n)}$ is an upper triangular matrix, and this system can be solved by back substitution.

A number of observations relating to the computational process are in order:

1. Computation of $\mathbf{A}^{(k)}$. Combining (B6), (B10) and (B11), it follows that

$$\mathbf{A}^{(k+1)} = \mathbf{M}_k\,\mathbf{A}^{(k)} \tag{B14}$$

$$= \mathbf{A}^{(k)} - m_k\,(e_k^{\mathsf{T}}\,\mathbf{A}^{(k)}). \tag{B15}$$

Notice that

$$e_k^{\mathsf{T}}\,\mathbf{A}^{(k)} = (0, 0, \dots, 0, a_{kk}^{(k)}, \dots, a_{kn}^{(k)})$$

is the k^{th} row of $\mathbf{A}^{(k)}$. Thus, the i^{th} row of $\mathbf{A}^{(k+1)}$ is obtained by subtracting m_{ik} times the k^{th} row of $\mathbf{A}^{(k)}$ from the i^{th} row of $\mathbf{A}^{(k)}$. As $m_{ik} = 0$, for $i = 1$ to k, only the rows $k+1$ through n of $\mathbf{A}^{(k)}$ are altered to obtain $\mathbf{A}^{(k+1)}$. Thus,

$$a_{ik}^{(k+1)} = 0, \qquad \text{for } i = k+1, k+2, \dots, n,$$

and

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}\,a_{kj}^{(k)}, \tag{B16}$$

for $i = k+1, k+2, \dots, m$, and $j = k+1, k+2, \dots, n$.

The above development naturally leads to the following algorithm [35,36].

ROW-ORIENTED LU FACTORIZATION.

```
FOR k = 1 TO n - 1
    FOR i = k + 1 TO n
        m_ik = a_ik / a_kk
        FOR j = k + 1 TO n
            a_ij = a_ij - m_ik a_kj
        END
    END
END
```

On vector machines, the j^{th} loop can be executed in the pipelined vector mode, and on multivector machines, such as the Alliant FX/8, the i^{th} loop can be executed in a concurrent mode with the j^{th} loop in the vector mode.

2. Computation of $\mathbf{d}^{(k)}$. By appending $\mathbf{d}^{(1)} = \mathbf{d}$ as the $(n+1)^{\text{th}}$ column of $\mathbf{A}$ and extending the j^{th} loop to $n+1$, we can readily integrate the computation of $\mathbf{d}^{(k)}$ along with $\mathbf{A}^{(k)}$.

3. Multiple Right-hand Side. Let there be r systems

$$\mathbf{A}\,\mathbf{x}_i = \mathbf{d}_i,$$

for $i = 1, 2, \ldots, r$ to be solved. We can simultaneously solve these r systems using the row-oriented L U decomposition algorithm by simply appending $\mathbf{d}_i$ to $\mathbf{A}$ as the $(n+i)^{\text{th}}$ column for $1 \le i \le r$, and extending the j^{th} loop to $n+r$. Since this process increases the length of the (row) vectors, it often leads to better vector efficiency.

REMARK. When the matrix $\mathbf{A}$ is a tridiagonal matrix (as in the BCR algorithm described in the text), by carefully avoiding computations involving the zero element, we can further reduce the overall time.