CONF-8706183--4

CSRD Rpt. No. 659

DOE/ER/25001--47

DE88 003595

A PARALLEL BLOCK CYCLIC REDUCTION ALGORITHM FOR THE FAST SOLUTION OF ELLIPTIC EQUATIONS

E. Gallopoulos and Youcef Saad

April 1987

Center for Supercomputing Research and Development
University of Illinois
305 Talbot – 104 South Wright Street
Urbana, IL 61801-2932
Phone: (217) 333-6223

This work was supported in part by the National Science Foundation under Grants No. US NSF DCR84-10110 and NSF DCR85-09970, the U. S. Department of Energy under Grant No. US DOE-DE-FG02-85ER25001, the U.S. Air Force Office of Scientific Research under Grant No. AFOSR-85-0211, and the IBM Donation.

[illegible]

ERRATA (**A parallel block cyclic reduction ..., CSRD report no. 659**)

(1) p. 5, l. 14: change $n/2$ to $(n+1)/2$.

(2) p. 6, l. 13: (in Lemma 1) change *does not exceed* to *is less than.*

(3) p. 6, l. –15: change "40% for small k and 20%" to "20% for small k and 10%".

(4) p. 6, l. –7: change $n+1$ to $\left\lfloor \frac{n+1}{2} \right\rfloor$.

(5) p. 8, l. 20: change $n+1-k \geq n/2$ to

$$\left\lfloor \frac{n+1}{2} \right\rfloor - k \geq \left\lfloor \frac{n+1}{4} \right\rfloor$$

A parallel block cyclic reduction algorithm for the fast solution of elliptic equations

E. Gallopoulos
Y. Saad

Center for Supercomputing Research and Development
University of Illinois at Urbana Champaign
Urbana, Illinois 61801
U.S.A.

Abstract. This paper presents an adaptation of the Block Cyclic Reduction (BCR) algorithm for a multi-vector processor. The main bottleneck of BCR lies in the solution of linear systems whose coefficient matrix is the product of tridiagonal matrices. This bottleneck is handled by expressing the rational function corresponding to the inverse of this product as a sum of elementary fractions. As a result the solution of this system leads to parallel solutions of tridiagonal systems. Numerical experiments performed on an Alliant FX/8 are reported.

1. Introduction

The numerical solution of linear elliptic partial differential equations is a problem of major importance in many fields of science and engineering. Aside from the traditional iterative methods, there exist several techniques that can solve many of the separable elliptic problems much faster [Hock70], [BuGN70], [Swar84]. The most widely used of these methods, which usually come under the general name *Rapid Elliptic Solvers* (RES), are based on either using the Fast Fourier Transform (FFT) to decouple the block tridiagonal systems into multiple scalar ones, or on using general *Block Cyclic Reduction* (BCR) as described in [Swee77], or finally, a combination of the above approaches as is the case in the FACR algorithm [Swar77], [Temp80]. For example, the solution of Poisson's equation on a rectangle discretized with an $n \times n$ grid, entails an asymptotic operation count of $O(n^3 \log n)$ and $O(n^2 \log^2 n)$ for the (iterative) SOR and ADI methods respectively, but only $O(n^2 \log n)$ for the cyclic reduction and FFT based methods.

With the advent of vector and parallel architectures, researchers have concentrated their efforts on making the best use of the computational resources in order to increase the performance of these solvers [SaCK76], [OrVo85]. The FFT based methods are very suitable for such a task because of the amenability of the FFT to parallel computation

This work was supported by the National Science Foundation under Grants No. US NSF DCR84-10110 and US NSF DCR85-09970, the US Department of Energy under Grant No. DOE DE-FG02-85ER25001, by the US Air Force under Contract AFSOR-85-0211, and the IBM donation.

and because of the immediate decoupling of the equations into multiple scalar tridiagonal systems. Thus, the development of highly efficient algorithms for the parallel/vector solution of multiple tridiagonal systems as well as for the FFT were instrumental in the acceptance of this approach. On the other hand, and despite some important advantages, the BCR methods didn't fare as well and little can be found in the literature concerning the implementation of BCR based elliptic problem solvers on the new generation supercomputers. The main reason for this is that as the BCR algorithm progresses, it appears that one must solve a steadily decreasing number of tridiagonal systems, the coefficient matrix of each being a matrix polynomial. The difficulty in the standard implementation is that one must solve in sequence a tridiagonal system per polynomial factor.

In this paper we describe a new method which is used to overcome this sequential bottleneck and in turn introduce parallelism at each step of the computation. Our approach can be used with success on most of the new supercomputers. An important advantage of the method is that it allows the efficient parallelization/vectorization of NCAR's FISHPACK (see [SwSw75]) which is entirely based on BCR for two–dimensional problems.

The structure of the paper is as follows: Section 2 describes BCR and the computational bottleneck for a parallel system, sections 3 and 5 describe the new algorithm, and section 4 discusses implementation issues. In section 6 we present some numerical experiments and in section 7 we propose a few tentative concluding remarks.

2. Background: the main bottleneck in Buneman's algorithm.

Consider a block tridiagonal system of the form

$$\begin{pmatrix} A & -I & & & \\ -I & A & -I & & \\ & \cdot & \cdot & \cdot & \\ & & -I & A & -I \\ & & & -I & A \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ v_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ f_n \end{pmatrix} \tag{2.1}$$

where each block is an $m \times m$ tridiagonal matrix and the sub–vectors v_i and f_i are all of size m. Such systems frequently arise when discretizing an elliptic partial differential equation of the form

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

with Dirichlet boundary conditions over, for example, a rectangular region.

For simplicity we assume at first that the block-size n of the system (2.1) is of the form $n = 2^{\mu}-1$. For a detailed description of the algorithm in the general case the reader is referred to [Swee77]. At the the $r-th$ step of BCR we have a system of the form:

$$\begin{pmatrix} A^{(r)} & -I & & & \\ -I & A^{(r)} & -I & & \\ & \cdot & \cdot & \cdot & \\ & & -I & A^{(r)} & -I \\ & & & -I & A^{(r)} \end{pmatrix} \begin{pmatrix} v_h \\ v_{2h} \\ \cdot \\ \cdot \\ v_{n,h} \end{pmatrix} = \begin{pmatrix} f_h \\ f_{2h} \\ \cdot \\ \cdot \\ f_{n,h} \end{pmatrix} \tag{2.2}$$

where $h = 2^r$, and each component of the right hand-side is put in the form

$$f_{jh} = A^{(r)} p_{jh}^{(r)} + q_{jh}^{(r)}. \tag{2.3}$$

Initially ($r = 0$) the vectors $p_i^{(0)}$ are zero while $q_i^{(0)} = f_j$. Equations whose block index j is odd are eliminated by multiplying each equation numbered $2jh$ by $A^{(r)}$ and adding to it equations $(2j-1)h$ and $(2j+1)h$. This yields a system of half the size involving only equations whose indices are even multiples of h. Specifically, the general equation becomes

$$-v_{(2j-2)h} + ((A^{(r)})^2-2I)v_{2jh} - v_{(2j+2)h} =$$
$$(A^{(r)})^2 p_{2jh}^{(r)} + A^{(r)}(q_{2jh}^{(r)} + p_{(2j-1)h}^{(r)} + p_{(2j+1)h}^{(r)}) + q_{2j-1)h}^{(r)} + q_{(2j+1)h}^{(r)}.$$

The above equation is then rewritten such that the new right-hand-sides will have a form similar to (2.3) in step $r+1$. This is achieved by defining

$$A^{(r+1)} = (A^{(r)})^2 - 2I \tag{2.4}$$

$$p_{2jh}^{(r+1)} = p_{2jh}^{(r)} + (A^{(r)})^{-1}(q_{2jh}^{(r)} + p_{(2j-1)h}^{(r)} + p_{(2j-1)h}^{(r)}) \tag{2.5}$$

$$q_{2jh}^{(r+1)} = 2p_{2jh}^{(r+1)} + q_{(2j-1)h}^{(r)} + q_{(2j-1)h}^{(r)}\ . \tag{2.6}$$

After $\mu-1$ steps of these transformations system (2.2) reduces to a system with a single block which can be solved directly. Then the back–substitution phase will consist of computing v_{jh}, for the odd values of j, using the fact that for even values of j, the v_{jh} have been computed in the previous step. Using (2.3), we see that we must compute, for j odd and $jh \leq n$:

$$v_{jh}=p_{jh}^{(r)}+(A^{(r)})^{-1}(q_{jh}^{(r)} + v_{(j-1)h} + v_{(j+1)h}) \cdot \tag{2.7}$$

The above backward substitution steps are performed for $h=2^r$ decreasing from 2^μ to $2^0=1$. In summary, Buneman's variant of BCR takes the following form:

Algorithm 1

```
1. Initialize: $p_i^{(0)}=0$, $q_j^{(0)}=f_j$, $j=1,...,n$ and $h=1$,

2. Forward solution:
 a. Form the matrix $Y_r$ with columns $q_{2jh}^{(r)} + p_{(2j-1)h}^{(r)} + p_{(2j-1)h}^{(r)}$, $j = 1,...,n_r/2$
 b. Solve the (multi)- linear system $A^{(r)}X_r = Y_r$
 c. Update the vectors $p$ and $q$ according to (2.5) and (2.6).
 d. If $h<n$ then $h = 2h$ ; go to a.

3. Solve for $u$, $A^{(r)}u = q_1^{(r)}$ and set $v_h=p_h + u$.

4. Backward substitution: while $h \geq 1$ do
 a. $h = h/2$
 b. Form the matrix $Y_r$ with column vectors $q_{jh}^{(r)} + v_{(j-1)h} + v_{(j+1)h}$ , $j=1,3,5...,n/h$.
 c. Solve for $U_r$, $A^{(r)}U_r=Y_r$
 d. Update the solution vectors $v_{jh}$, $j=1, 3$ , .. according to (2.7).
```

The most time–consuming part of the above algorithm lies in the solution of the linear system in the forward and backward phases, i.e., in 2.b and 4.c. Clearly, the matrix $A^{(r)}$ is never formed explicitly. In fact the way in which 2.a and 4.c are usually performed is by first observing that the matrix $A^{(r)}$ is a known polynomial in A, specifically [Swee77]

$$A^{(r)} = 2T_{2^r}(\frac{A}{2}) \equiv \prod_{i=1}^{2^r}(A-\lambda_i I)$$

where T_k denotes the Chebyshev polynomial of the first kind of degree k and therefore

$$\lambda_i = 2\cos\frac{(2i-1)\pi}{2^{r+1}}$$

When implemented on a serial computer, the linear systems corresponding to each of the factors and each of the right–hand–sides are solved in succession. Moreover, Gaussian elimination is used to solve these tridiagonal systems. On vector and parallel machines, these facts result in less efficient utilization of the available computational resources. An obvious but partial remedy is to still use Gaussian elimination but solve for all the right–hand–sides simultaneously. This will achieve good performance at the beginning of the forward sweep and at the end of the backward sweep since there are many such right–hand–sides. However, looking at the forward solution step only, the number of right–hand–sides decreases exponentially ($(n-1)/2$ then $(n-1)/4, \ldots, 1$), while the number of factors increases in the reverse order. This means that this high performance can only be enjoyed at the very first few steps. A similar situation occurs in the backward solution phase where the number of right–hand sides starts at one and then doubles at every step to reach the maximum number $n/2$ at the end of step 4.

Another way of introducing parallelism is to use (scalar) cyclic reduction for the tridiagonal systems. However, the cost of cyclic reduction in terms of operation count is high and its use may lead to disappointing speed–ups. This was in particular pointed out in [Saad87] in the context of Alternating Direction Implicit (ADI) methods.

3. The new algorithm .

In this section, we continue with the assumption that we are solving a problem with Dirichlet conditions on the boundary and with a rectangular grid of dimension $n \times m$ with $n=2^{\mu}-1$. The basic problem to be solved at each step of BCR is of the form

$$\prod_{i=1}^{i=k} (A-\lambda_i I)X = Y \qquad (3.1)$$

in which X and Y are matrices of dimension $m \times \nu$. The usual way of solving (3.1) is by means of the following algorithm:

Algorithm 2

```
1. Set $X_0 = Y$
2. Do i=1,...,k
   a. Set $Y_i = X_{i-1}$
   b. Solve $(A-\lambda_i I)X_i = Y_{i-1}$
3. $X = X_k$
```

As was pointed out earlier when ν is large enough, the amount of vectorization and parallelization that is available in each of the inner loops of the above algorithm with the standard Gaussian elimination may be sufficient to deliver reasonable performance. The vector length with this simple approach is ν so that when ν is small, the delivered speed may become inadequate.

To introduce the alternative we propose, we start by observing that we are seeking the matrix $[g_k(A)]^{-1}Y \equiv s_k(A)Y$ where s_k is the rational function

$$s_k(t) \equiv \frac{1}{g_k(t)} \equiv \frac{1}{\prod_{i=1}^{k}(t-\lambda_i)} \tag{3.2}$$

We will need the following well-known elementary result:

Lemma 1
Every rational function

$$f(t) = \frac{u_l(t)}{g_k(t)} = \frac{u_l(t)}{\prod_{i=1}^{k}(t-\lambda_i)},$$

where the degree l of u_l does not exceed k and where the poles λ_i are all distinct, can be expanded in terms of elementary fractions as follows

$$f(t) = \sum_{i=1}^{k} \frac{\alpha_i}{t-\lambda_i}$$

where the scalars α_i are given by

$$\alpha_i = \frac{u_l(\lambda_i)}{g_k'(\lambda_i)}$$

In the context of BCR the polynomials involved are Chebyshev polynomials of the first and second kind, so that the partial fraction coefficients can be easily calculated (cf. section 5). For example, in the case we are describing, f is given by (3.2) with $g_k(t)=2T_k(t/2)$ and a little calculation shows that

$$\alpha_i = \frac{(-1)^{i-1}}{2^r} \cdot \sin\frac{2i-1}{2^{r+1}}\pi$$

As a result we can decompose the solution X as

$$X=\sum_{i=1}^{k}\alpha_i(A-\lambda_i I)^{-1}Y$$

where α_i are given by (3.5). This leads to the following rival for Algorithm 2:

Algorithm 3

```
1. Do for i=1,2,...,k
      Solve $(A-\lambda_i I)X_i=Y$
2. Compute $X=\sum_{i=1}^{k}\alpha_i X_i$
```

The clear advantage of Algorithm 3 over Algorithm 2 is that the tridiagonal systems involving the matrix $(A-\lambda_i I)$ are now decoupled and can be solved in parallel. In essence we have replaced a problem involving sequential solutions of tridiagonal linear systems by one which involves the simultaneous solution of independent tridiagonal linear systems and the linear combination of the partial results.

Two important questions might be asked at this point. First there is the danger of instability: the algorithm is not acceptable if it produces wrong answers. A simple analysis based on the exact expressions of the α_i's shows that the process is extremely stable. This fact is also confirmed by the experiments. The second question concerns cost. The summation in step 2 of the algorithm involves an additional $2k\times m\times\nu=2\times m\times n$ operations to be added to the usual $5\times m\times n+5\times k\times m$ associated with the traditional algorithm 2. Thus the additional operation count is not negligible: approximately 40% for small k and 20% for large k. The above are approximate operation counts obtained by dropping the less significant terms. Note however that this additional work is perfectly amenable to parallelization and high performance can be achieved.

4. Implementation issues

The most difficult issue that one faces when implementing the approach outlined in Section 3 is the fact that in the course of BCR, the two parameters k, the degree of the polynomial, and ν the number of right-hand sides, vary widely from one outer step to the other. In fact the relation $k\times(\nu+1)=n+1$ holds for the case when n is of the form $2^{\mu}-1$ which means that the two numbers k and $\nu+1$ will change exponentially and in inverse proportion of each other. As a result one must employ different strategies for the various cases: k large but ν small, ν large and k small, etc.

First we discuss one way of implementing the algorithm for a multi-vector processor architecture. The model architecture is that of an Alliant FX/8 which consists of 8 Computational Elements (CE's) each being a vector processor. From the above discussion we must distinguish two cases.

Case 1: ν is large. Here one can vectorize the operations with respect to the ν right–hand sides. Moreover if k is large enough (larger than the number of CE's) it is important to run concurrently the outer loop of step 1 in Algorithm 3. However, this becomes ineffective as k goes below the number of CE's since then fewer CE's would be active. In this particular case it is preferable to run the outer loop sequentially and perform the vector operations within the subroutine in a vector concurrent mode.

Case 2: k is large. The situation is the opposite of the previous one: vectorization of the operations should now be performed along the variables corresponding to the outer loop of Algorithm 3. Again the outer loop of step 1 in Algorithm 3 should be performed concurrently unless ν the number of right–hand sides is smaller than the number of CE's in which case it should be executed sequentially, with the vector loops run in a vector–concurrent mode.

Next we should say a few words on how Algorithm 3 can be implemented on a vector computer. Neither of the two approaches outlined above for the two different cases is likely to be effective because the vector lengths will decrease very quickly. An approach that seems best suited for a vector machine is to exploit the observation that at every step we have in fact a large number, namely $k \times \nu$, of tridiagonal systems to solve simultaneously. This leads to a process which is vectorizable with vector length equal to $k\nu = n+1-k \geq n/2$ which will be satisfactory for reasonable size problems. These $k\nu$ simultaneous tridiagonal systems are obtained by simply reproducing the tridiagonal matrix $(A - \lambda_i I)$ involved in the (multi) linear system $(A - \lambda_i I)x = b_i$, $i = 1,2,\ldots,\nu$ for each different right–hand–side. Step 1 of Algorithm 3, is thereofre expanded into the simultaneous solution of exactly $k \times \nu$ independent tridiagonal systems each with a single right–hand–side. Although good performance can be achieved when n is large, this approach has a drawback which needs to be evaluated more carefully. Indeed, in order to achieve computations with vector lengths of $k\nu$, we must actually reproduce the computed data corresponding to forward solution in the tridiagonal solve ν times. For example, in the first step we can compute in vector mode all the divisions $1/(a_{11} - \lambda_i)$, $i = 1,2,\ldots,k$ then create a copy of the result for each of the ν right hand sides. Although this involves no arithmetic, the price to pay may be far from negligible.

5. The general case

Sweet generalized the Buneman variant of BCR for general n and showed how it can be applied for any boundary conditions [Swee77]. We describe here the coefficients of the expansions necessary for the application of our method for general n as well as Dirichlet or Neumann boundary conditions. We postpone the description of the detailed, generalized algorithms to a forthcoming paper. As before, the key to the success of the method is the ability to express the matrix blocks at each step of BCR as a rational or polynomial matrix form. In fact, whereas before all the diagonal blocks in (2.2) involved the matrix polynomial $A^{(r)}$, in the general case the last diagonal block in (2.2) will be of the form $B^{(r)^{-1}} C^{(r)}$ which can be shown to be a rational function in A. In particular $B^{(r)}$ and $C^{(r)}$ are matrix polynomials in A of degree k_r and l_r respectively, with $k_r = l_r + 2^r$. Depending on the boundary conditions and dimensions, these will be Chebyshev polynomials of the first or the second kind. Specifically, following the notation in [Swee77], the systems that must be solved at each step of the Buneman variant of BCR have one of the following five forms:

$$2\, T_{2^r}\, (\frac{1}{2}A)\, X = Y \tag{4.1}$$

$$U_{k_r}\, (\frac{1}{2}A)\, X = U_{l_r}\, (\frac{1}{2}A)\, Y \tag{4.2}$$

$$U_{k_r}\, (\frac{1}{2}A)\, X = 2\, U_{l_r}\, (\frac{1}{2}A)\, T_{2^r}(\frac{1}{2}A)\, Y \tag{4.3}$$

$$T_{k_r}\, (\frac{1}{2}A)\, X = T_{l_r}\, (\frac{1}{2}A)\, Y \tag{4.4}$$

and

$$T_{k_r}\, (\frac{1}{2}A)\, X = 2\, T_{l_r}\, (\frac{1}{2}A)\, T_{2^r}(\frac{1}{2}A)\, Y \tag{4.5}$$

where T_k and U_k denote, respectively, the Chebyshev polynomials of the first kind

$$T_k(\cos\theta) = \cos(k\theta),$$

and of the second kind

$$U_k(\cos\theta) = \frac{\sin((k+1)\theta)}{\sin\theta}.$$

Note in particular that (4.1) arises when $n = 2^\mu - 1$ described in the previous section, (4.2) and (4.3) when $n \neq 2^\mu - 1$ and with Dirichlet boundary conditions. (4.1), (4.4) and (4.5) arise in the corresponding circumstances but for Neumann boundary conditions.

In product representation

$$T_k(t) = \frac{1}{2} \cdot \prod_{i=1}^{k} (2t - \rho_k(i))$$

and

$$U_k(t) = \prod_{i=1}^{k} (2t - \sigma_k(i))$$

where

$$\rho_k(i) = 2\cos\frac{(2i-1)}{2k}\pi$$

and

$$\sigma_k(i) = 2\cos\frac{i}{k+1}\pi$$

From Lemma 1 it follows that we need to calculate:

i) for (4.1)

$$\alpha_i = \left[\frac{d}{dt}\left(2T_{2^r}(t/2)\right)\right]^{-1}$$

at $t = \rho_{2^r}(i)$ $(i = 1, \ldots, 2^r)$;

ii) for (4.2)

$$\beta_i = U_{l_r}(t/2)\,.\left[\frac{d}{dt}U_{k_r}(t/2)\right]^{-1}$$

at $t = \sigma_{k_r}(i)$ $(i = 1, \ldots, k_r)$;

iii) for (4.3)

$$\gamma_i = 2\,U_{l_r}(t/2)\,.\,T_{2^r}(t/2)\,.\left[\frac{d}{dt}\left(U_{k_r}(t/2)\right)\right]^{-1}$$

at $t = \sigma_{k_r}(i)$ $(i = 1, \ldots, k_r)$;

iv) for (4.4)

$$\delta_i = T_{l_r}(t/2)\,.\left[\frac{d}{dt}T_{k_r}(t/2)\right]^{-1}$$

at $t = \rho_{k_r}(i)$ $(i = 1, \ldots, k_r)$;

v) for (4.5)

$$\varsigma_i = 2\,T_{l_r}(t/2)\,.\,T_{2^r}(t/2)\,.\left[\frac{d}{dt}\left(T_{k_r}(t/2)\right)\right]^{-1}$$

at $t = \rho_{2^r}(i)$ $(i = 1, \ldots, k_r)$.

The lemma below follows after some algebraic manipulations and by using the relation $k_r = 2^r + l_r$.

Lemma 2

$$\alpha_i = \frac{(-1)^{i-1}}{2^r} \,.\, \sin\frac{(2i-1)\pi}{2^{r+1}}$$

$$\beta_i = \frac{2}{k_r+1} \,.\, \sin\frac{2^r i\pi}{k_r+1} \,.\, \sin\frac{i\pi}{k_r+1}$$

$$\gamma_i = \frac{2}{k_r+1} \,.\, \sin\frac{2^{r+1} i\pi}{k_r+1} \,.\, \sin\frac{i\pi}{k_r+1}$$

$$\delta_i = \frac{2}{k_r} \,.\, \sin\frac{2^{r-1}(2i-1)\pi}{k_r} \,.\, \sin\frac{(2i-1)\pi}{2k_r}$$

and

$$\zeta_i = \frac{2}{k_r} \,.\, \sin\frac{2^r(2i-1)\pi}{k_r} \,.\, \sin\frac{(2i-1)\pi}{2k_r}$$

6. Numerical Experiments

We have applied our method to Poisson's equation

$$u_{xx} + u_{yy} = f(x,y),$$

on the unit square with Dirichlet boundary conditions. The right hand side f and the Dirichlet boundary conditions are defined so that the true solution is

$$u(x,y) = \sin\left[\frac{\pi.(x-y+2)^5}{(\,1+(x-y+2)^4\,)}\right]$$

This is identical with problem 11 of [RiHD81].

The resolution was chosen so that the number of points in each direction is $n = 2^\mu - 1$, with μ ranging from 6 to 9. The resulting matrix has block dimension $n \times n$ with each block being of size $n \times n$.

Table 1 shows the timings and discretization errors for the original and modified versions of the NCAR package running on the CSRD Alliant FX/8 system using version 3 of the Concentrix operating system. All of the experiments were performed in double precision arithmetic. We note that the experiments were not done in single-user mode. Hence in order to reduce the effect of other user jobs on the timings, the main program solved the problem 4 times and the total runtime was then averaged. Supporting our previous comments regarding the error accumulation of the new algorithms, we found that the maximum difference between the true and the computed solutions was the same for both methods.

We should point out that the times reported are the total times and do not reflect the gains made in the steps which have been optimized namely the steps corresponding to solving the linear systems. If these were to be timed separately these gains could be substantially better especially for the smaller grid–size problems. For example it is estimated that for the 255×255 grid, the time not attributed to the solution of linear systems takes up to 0.5 sec.

Observe that as the grid–sizes increase the gains in total time appreciate steadily. This is to be expected from the very sequential nature of the original FISHPACK. For small grids the total gain in the linear systems is somewhat masked by the time spent in the parts of the algorithm that are not related to solving linear systems. Moreover, the benefits of vectorization and parallelization become visible only after a minimal vector length is available. Typically, on the Alliant FX/8, vector processing on each CE becomes more effective than scalar processing after the vector length exceeds 4.

Finally, the times on one CE do not show a speed–up when the grid–size is below n=255. Our algorithm has been optimized for the case where the number of CE's is 8 and does attempt to reduce the overhead when the number of processors is much smaller. Here it would have been better to use the alternative proposed for vector processors as discussed in Section 4.

Dimension	CE's	Original times (sec)	New Times (sec)
63	1	0.41	0.67
63	4	0.32	0.22
63	8	0.31	0.16
127	1	1.93	2.03
127	4	1.49	0.64
127	8	1.43	0.44
255	1	8.86	5.60
255	4	6.82	1.70
255	8	6.52	1.28
511	8	29.70	5.55

Table 1: Runtimes of the original and modified FISHPACK on Alliant FX/8

7. Concluding remarks

Our preliminary results indicate that the method described in this paper is a promising alternative to FFT–based RES. Its reliance on a simple partial fraction expansion, makes it possible to apply the same idea to problems with Neumann or periodic boundary conditions, to problems involving other coordinate systems, as well as to the more general separable elliptic problems. Because of the simplicity of the technique, great care must be taken in choosing the parallel tridiagonal solvers to be used. The proper choice depends on many factors, particularly the exact set of computational resources available and the proper balance between parallelism and vectorization at each step of the computation. Moreover, it is important to note that using a square grid, as in the experiments, is a "worst case" situation. For any other rectangular grid, the improvement in speed with the new method will be even better. This is so, because of the strategy to vectorize and parallelize across systems, instead of seeking the vectorization within the tridiagonal system solver (e.g. using scalar cyclic reduction). A systematic description of the method for each case of boundary conditions, values of $n \neq 2^{\mu}-1$, its stability properties, as well as comparisons with other methods will be given in a forthcoming paper.

Acknowledgement. We would like to thank Kyle Gallivan for his helpful comments on the manuscript.

Bibliography

[BuGN70]
B. Buzbee, G. Golub, and C. Nielson, "On direct methods for solving Poisson's equation" *SIAM J. Numer. Anal*, vol. 7, pp. 627–656, (December 1970).

[Hock70]
R. Hockney, "The Potential Calculation and Some Applications", in *Methods Comput. Phys.*, v. 9, pp. 135–211, Academic Press, (1970).

[OrVo85]
J. Ortega and R. Voigt, "Partial differential equations on vector and parallel computers", *SIAM Review*, pp. 213–240, (June 1985).

[RiHD81]
J. Rice, E. Houstis and W. Dyksen, "A population of linear, second order elliptic partial differential equations on rectangular domains. Part 1", *Math. Comp.*, v. 36, pp. 479–484, (1981).

[Saad87]
Y. Saad, "On the design of parallel numerical methods in message passing and shared–memory environments", Proc. International Seminar on Scientific Supercomputer, Paris, (2–6 February 1987).

[SaCK76]
A. H. Sameh, S. C. Chen, D. J. Kuck, "Parallel Poisson and biharmonic solvers", *Computing*, vol. 17, pp. 219–230 (1976).

[Swar77]
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*, v. 19, pp. 490–501, (July 1977).

[Swar84]
P. N. Swarztrauber, "Fast Poisson solvers", in *Studies in Numerical Analysis*, G. H. Golub ed., pp. 319–369, Mathematical Association of America, (1984).

[SwSw75]
P. Swarztrauber and R. Sweet, "Efficient Fortran subprograms for the solution of elliptic partial differential equations", NCAR Technical Note IA–109, Boulder, (July 1975).

[Swee77]
R. A. Sweet, "A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension", *SIAM J. Numer. Anal.*, vol. 14, pp. 707–720, (September 1977).

[Temp80]
C. Temperton, "On the FACR(l) algorithm for the discrete Poisson equation", *J. of Comp. Physics*, v. 34, pp. 314–329, (1980).

BIBLIOGRAPHIC DATA SHEET	1. Report No. CSRD-659	2.	3. Recipient's Accession No.

4. Title and Subtitle
PARALLEL BLOCK CYCLIC REDUCTION ALGORITHM FOR THE FAST SOLUTION OF ELLIPTIC EQUATIONS

5. Report Date
April 1987

6.

7. Author(s)
E. Gallopoulos and Y. Saad

8. Performing Organization Rept. No. CSRD-659

9. Performing Organization Name and Address
University of Illinois at Urbana-Champaign
Center for Supercomputing Research and Development
Urbana, IL 61801-2932

10. Project/Task/Work Unit No.

11. Contract/Grant No.NSF DCR84-10110 and NSF DCR85-09970; US DOE DE-FG02-85ER25001; AFOSR-85-0211; IBM Corp.

12. Sponsoring Organization Name and Address
National Science Foundation, Washington, DC
US Department of Energy, Washington, DC
US Air Force Office of Scientific Research, Washington, DC
IBM Corporation, Armonk, NY

13. Type of Report & Period Covered
Technical Report

14.

15. Supplementary Notes

16. Abstracts
This paper presents an adaptation of the Block Cyclic Reduction (BCR) algorithm for a multi-vector processor. The main bottleneck of BCR lies in the solution of linear systems whose coefficient matrix is the product of tridiagonal matrices. This bottleneck is handled by expressing the rational function corresponding to the inverse of this product as a sum of elementary fractions. As a result the solution of this system leads to parallel solutions of tridiagonal systems. Numerical experiments performed on an Alliant FX/8 are reported.

17. Key Words and Document Analysis. 17a. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 14
	20. Security Class (This Page UNCLASSIFIED	22. Price