Parallel Computing 11 (1989) 37–53
North-Holland

On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy *

A.T. CHRONOPOULOS

Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.

C.W. GEAR

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, U.S.A.

Received August 1987
Revised January 1988

Abstract. Multiprocessor architectures combining vector and parallel processing capabilities on a two-level shared memory structure have been implemented. This memory hierarchy structure requires that numerical algorithms possess good data locality in order to achieve high performance rates. The s-step Conjugate Gradient method (s-CG) is a generalization of the standard CG method with improved data locality and parallel properties. Here we show how to implement efficiently the Incomplete Cholesky and Polynomial Preconditioning with s-CG on multiprocessors with memory hierarchy.

Keywords. Preconditioned conjugate gradient method, implementation features, model problem, parallelisation of preconditioning, ALLIANT FX/8.

1. Introduction

Memory contention on shared memory machines constitutes a severe bottleneck for achieving the maximum performance. The algorithm should not only lend itself to vectorization and parallelization but it must provide good data locality; that is, the organization of the algorithm should be such that the data can be kept as long as possible in fast registers or local memories and have many arithmetic operations performed on them. A good first measure of the data locality of a set of vector operations is the size of

$$\text{Ratio} = (\text{Memory References})/(\text{Floating Point Operations}). \tag{1.1}$$

The data locality of the computation is good if this ratio is much less than one.

Some dense linear algebra algorithms for efficient use of local memories have been studied in [6,9]. Here we are concerned with algorithms for large sparse problems. We consider the use of preconditioned conjugate gradient methods in approximating the solution of a sparse $N \times N$ symmetric and positive definite (SPD) linear system of equations

$$Ax = f. \tag{1.2}$$

* The research was partially supported by the U.S. Department of Energy under grant DOE DEFG02-87ER25026 while the first author was at the University of Illinois.

The multiprocessor implementation of CG has two difficulties in achieving high speedups. Firstly, the inner products in each iteration require synchronization of all the processors in a parallel system. Secondly, temporary vectors are moved very frequently (even within one iteration) between the global memory and the local memories of the processors. Van Rosendale [19] studied the data dependencies of inner products in CG and used recursive formulas to precompute the inner products of an iteration from parameters computed in past iterations. Barkai et al. [2] and Saad [14] present a modification of the CG algorithm which does one sweep of the global memory per iteration.

In [4] we obtained on s-step form of the Hestenes and Stiefel CG [8] which does one memory sweep per s CG iterations and allows simultaneous execution of the $2s$ required inner products. In the s-CG iteration s new directions are formed simultaneously from $\{r_i, ar_i, \dots, A^{s-1}r_i\}$ and the preceding s directions. All s directions are chosen to be A-orthogonal to the preceding s directions. The approximation to the solution is then advanced by minimizing an error functional simultaneously in all s directions. This intuitively means that the progress towards the solution in one iteration of the s-step method equals the progress made over s consecutive steps of the one-step method. This is proven to be true. If the matrix is banded the s-step method can be organized so that only one global memory sweep is required to complete one step of s-CG (or s successive steps of CG). This means that one of the reasons that this method is faster than the standard CG is the efficient use of the fast local memories in a memory hierarchy system.

We should point out that the s-step CG is different from two iterative methods with which it may seem to overlap in the goals achieved. These methods are the block CG [13] and the Lanczos algorithm for solving linear systems [15].

The block CG is used to solve $AX = B$ with dimension of $X = N \times m$. This, for example, is the case when CG is used to solve $Ax = b$ for m right-hand sides. The s-step CG is applied to solve the linear system with a single right-hand side.

In the Lanczos method on orthonormal basis $V_m = [v_1, \dots, v_m]$ is build for the Krylov space $\{r_0, Ar_1, \dots\}$ starting from the residual vector $r_0 = b - Az$. At the same time the symmetric tridiagonal reduction matrix T_m of the matrix A is formed. After convergence is reached the approximate solution is obtained by inverting the tridiagonal reduction matrix. The size of the matrix is approximately equal to the total number of steps in CG using the same stopping criterion. Details can be found in [15]. The Lanczos algorithm forms serially the vectors v_j, $j = 1, \dots, m$ using a matrix multiply with the preceding vector and two inner products per iteration. Thus it has the same shortcomings for parallel processing as the standard CG method.

Our main goal in this paper is show that the preconditioned s-CG method can be efficiently implemented so that only one global memory sweep (per s CG iterations) is required. We use the polynomial and the incomplete Cholesky preconditionings. The preconditioned s-CG method computes

$$\left\{Kr_i, (KA)Kr_i, \dots, (KA)^{s-1}Kr_i\right\} \tag{1.3}$$

with $K \approx A^{-1}$ at each iteration. For the polynomial preconditioner [10,14], we show that this computation can be organized in a way similar to the plain s-CG because K is simply a polynomial expression A. However, for the Incomplete Cholesky (ICCG) [17,18] this is not the case even for the vectorizable incomplete Cholesky preconditioner (VICP). The fact that in Ku there is a forward step which must be completed before the backward step can start prevents computing (1.3) with only one main memory sweep. This motivated us to extend (VICP) to a completely parallel preconditioner. Although we have considered three distinct preconditioners (VICP, PICP, PP) in the theoretical implementation study we only include numerical experiments with VICP and PICP. We implemented PICP only with CG to show that it is an effective preconditioning.

It should be noted that the preconditioned s-step CG is different from the m-step preconditioned CG [1]. In the latter, the preconditioner has the form $K=(I+G+\cdots+G^{m-1})P^{-1}$ with $G=P^{-1}Q$ and $A=P-$ is a splitting of A.

2. A model problem

Large, sparse, and structured linear systems arise frequently in the numerical integration of partial differential equations (PDEs). Thus we borrow our model problems from this area. Let us consider the second-order elliptic PDE in two dimensions in a rectangular domain Ω in $\mathbf{R}^2$ with homogeneous Dirichlet boundary conditions:

$$-(au_x)_x-(bu_y)_y+(eu)_x+(hu)_y+cu=g \tag{2.1}$$

where $u=H$ on $\partial\Omega$, and $a(x, y)$, $b(x, y)$, $c(x, y)$, $e(x, y)$, $f(x, y)$ and $g(x, y)$ are sufficiently smooth functions defined on Ω, and a, $b>0$, $c\geqslant 0$ on Ω. If we discretize (2.1) using the five-point centered difference scheme on a uniform $n\times n$ grid with $h=1/(n+1)$, we obtain a linear system of equations

$$Ax=f$$

of order $N=n^2$. If $e(x, y)\equiv h(x, y)\equiv 0$, then (2.1) is self-adjoint and A is symmetric and weakly diagonally dominant [16]. If we use the natural ordering of the grid points, we get a block tridiagonal matrix of the form

$$A=[C_{k-1}, T_k, C_k],\quad 1\leqslant k\leqslant n,$$

where T_k, C_k are matrices of order n; and $C_0=C_n=0$. The blocks have the form

$$C_k=\operatorname{diag}[c_1^k,\dots,c_n^k],\qquad T_k=[b_{i-1}^k, a_i^k, b_i^k],\quad 1\leqslant i\leqslant n,$$

with $b_i^k<0$, $c_i^k<0$, $b_0^k=b_n^k=0$, and $a_i^k>0$.

Suppose the three-dimensional problem were considered with 7-point line discretization and with natural ordering applied to the planes and to the discretization points in each plane. The matrix A would then be symmetric, weakly diagonally dominant, block tridiagonal of order n^3 and have the form

$$A=[D_k, \overline{T}_k, D_k],\quad 1\leqslant k\leqslant n,$$

where, $D_k=\operatorname{diag}[d_1^k,\dots,d_{n^2}^k]$ with $d_j^k<0$ and the blocks $\overline{T}_k$ have the form of the matrix for the 2-D case (just discussed).

3. The preconditioned conjugate gradient method

If K is a symmetric and positive definite matrix, then applying the conjugate gradient method to the transformed system $[K^{1/2}AK^{1/2}]K^{-1/2}x=K^{1/2}f$ gives rise to the following algorithm. The matrix K is called the preconditioning matrix and is selected as an approximation to the inverse of A.

Algorithm 3.1. The preconditioned conjugate gradient method (PCG).

```
Choose $x_0$
$p_0 = Kr_0 = K(f - Ax_0)$
Compute $(r_0, Kr_0)$
**For** $i = 0$ **Until** Convergence **Do**
1. Compute and Store $Ap_i$
2. Compute $(p_i, Ap_i)$
3. $a_i = (r_i, Kr_i)/(p_i, Ap_i)$
4. $x_{i+1} = x_i + a_i p_i$
5. $r_{i+1} = r_i - a_i Ap_i$
6. Compute $(r_{i+1}, Kr_{i+1})$
7. $b_i = (r_{i+1}, Kr_{i+1})/(r_i, Kr_i)$
8. $p_{i+1} = Kr_{i+1} + b_i p$
**EndFor**.
```

Storage is required for the entire vectors x, Kr, p, Ap and maybe the matrix A (or K). Note that steps 2 and 3 must be completed before steps 4 and 5, and steps 6 and 7 must be completed before step 8. This forces double access of vectors r, p, Ap from the main memory at each CG step.

At the ith step the residual error $E(z) = (h - x, A(h - x))$, where $h = A^{-1}f$, is minimized over the translated i-dimensional Krylov subspace $x_0 + \{Kr_0, \ldots, KA^{i-1}r_0\}$. Thus convergence occurs (in infinite arithmetic) in at most N iterations. The purpose of preconditioning is to decrease the work needed to solve the system. This is accomplished if K is a good approximation to the inverse of A because the preconditioned system has smaller condition number than the original system. We note that the number of iterations required for convergence does not exceed the degree of the minimal polynomial of r_0 and CG tends to treat tight clusters as single eigenvalues. Hence, a good preconditioner should also group the eigenvalues of A into tight clusters. The preconditioned system requires fewer steps to convergence. This is also good for stability because fewer direction vectors must be generated.

The sequential and parallel complexity of a single step of CG for the 2-D and 3-D model problem is shown in Table 1 (n_d = the number of nonzero diagonals of A). For the matrix K we assume that polynomial or incomplete Cholesky preconditioning is used. For the purposes of Table 1, the parallel system is assumed to have at least O(N) processors. For uniprocessors or multiprocessors with a small number of processors (e.g. four or eight processors) the matrix-vector products dominate the computation whereas on parallel systems (e.g. Hypercube) the inner products dominate because they require global communication (synchronization of all processors) of the system. Performing an inner product on a parallel system can be thought of as a binary tree height reduction with the nodes of the tree being the processors of the system.

For shared memory systems with few processors, processor synchronization is fast but accessing data from the main memory may be slow. Thus in this case the data locality of the three parts of the computation determine the actual time complexity of the algorithm. The data

Table 1
Serial and parallel complexity of CG parts

Operation	Sequential	Parallel
Vector updates	$O(N)$	$O(1)$
Inner products	$O(N)$	$O(\log_2 N)$
Matrix-vector products	$O(n_d N)$	$O(\log_2 n_d)$

locality of Algorithm 3.1 is poor. For example the inner products and the vector updates provide a ratio (as defined (1.1)) of one and 3/2 respectively.

The preconditioning cost should be such that the total runtime for PCG is less than that of CG. On a scalar processor this can be calculated in terms of the total number of floating-point operations (ops) performed. PCG requires fewer iterations to converge than CG but each iteration involves the additional ops of a matrix vector multiplication by the matrix K. On a vector or parallel system the different parts of CG (inner products, vector updates, multiplications by A) may be running at different speeds and thus it is possible that the choice of a fast preconditioner should not be based on introducing the fewest number of ops.

4. The s-step preconditioned conjugate gradient method (s-CG)

The s-step Conjugate Gradient [4] is a generalization of the Hestenes and Stiefel CG [8] with improved data locality and parallel properties. The s-step method can be organized so that only one sweep through the data is required to complete one step of s-CG (equivalent to s successive steps of CG). This means that the method makes more efficient use of slower memory in a memory hierarchy system than the standard CG method. Also, the method can be organized so that the $2s$ inner products required for one s-step iteration are executed simultaneously. This reduces the need for frequent global communication in a parallel system and enhances the performance of the method by pipelining the $2s$ inner products.

One way to obtain an s-step conjugate gradient method is to use the s linearly independent directions $\{r_i, \ldots, A^{s-1}r_i\}$ to lift the iteration s dimensions out of the ith step Krylov subspace $\{r_0, \ldots, A^{is}r_0\}$. These directions must be made A-conjugate to the preceding s directions, which we will call $\{p_{i-1}^1, \ldots, p_{i-1}^s\}$. Finally, the error functional $E(x)$ must be minimized simultaneously in all s new directions to obtain the new residual r_{i+1}. This method is outlined in the following algorithm.

Algorithm 4.1. The s-step Conjugate Gradient Method (s-CG).

```
$x_0$, $p_0^1 = r_0 = f - Ax_0, \ldots, p_0^s = A^{s-1}r_0$
For $i = 0$ Until Convergence Do
    $x_{i+1} = x_i + a_i^1 p_i^1 + \cdots + a_i^s p_i^s$
    Select $a_i^j$ to minimize $E(x)$ over $L_i^s = \{x_i + \Sigma_{j=1}^s a_i^j p_i^j\}$
    Compute $r_{i+1} = f - Ax_{i+1}, A^1 r_{i+1}, \ldots, A^{s-1} r_{i+1}$
    Select $\{b_i^{(j,l)}\}$ to force A-conjugacy of $\{p_{i+1}^1, \ldots, p_{i+1}^s\}$, $\{p_i^1, \ldots, p_i^s\}$ where
        $p_{i+1}^1 = r_{i+1} + b_i^{(1,1)} p_i^1 + \cdots + b_i^{(1,s)} p_i^s$
        $p_{i+1}^2 = Ar_{i+1} + b_i^{(2,1)} p_i^1 + \cdots + b_i^{(2,s)} p_i^s$
        ...
        $p_{i+1}^s = A^{s-1} r_{i+1} + b_i^{(s,1)} p_i^1 + \cdots + b_i^{(s,s)} p_i^s$
EndFor
```

The parameters $\{b_{i-1}^{(j,l)}\}$ and a_i^j are determined by solving $s+1$ linear systems of equations of order s. In order to describe these systems we need to introduce some notation.

Remark 4.1. Let $W_i = \{(p_i^j, Ap_i^l)\}$, $1 \leqslant j, l \leqslant s$. W_i is symmetric. It is nonsingular if and only if $p_i^1, \ldots, p_i^s$ are linearly independent.

Remark 4.2. For $j = 1, \ldots, s$ let $\{b_{i-1}^{(j,l)}\}$, $1 \leqslant l \leqslant s$ be the parameters used in updating the direction vector p_i^j. We use the following s-dimensional vectors to denote them (for simplicity we drop the index i from the vectors):

$$\boldsymbol{b}^1 \left[b_{i-1}^{(1,1)}, \ldots, b_{i-1}^{(1,s)}\right]^{\mathrm{T}}, \ldots, \boldsymbol{b}^s = \left[b_{i-1}^{(s,1)}, \ldots, b_{i-1}^{(s,s)}\right]^{\mathrm{T}}.$$

For p_i^j to be A-conjugate to $\{p_{i-1}^1, \ldots, p_{i-1}^s\}$ it is necessary and sufficient that

$$W_{i-1}b^1 + c^1 = 0, \ldots, W_{i-1}b^s + c^s = 0 \tag{4.1a}$$

where the vectors c^j, $1 \leqslant j \leqslant s$ are

$$c^1 = \left[(r_i, A_{i-1}^1), \ldots, (r_i, Ap_{i-1}^s)\right]^{\mathrm{T}}$$

$$\ldots$$

$$c^s = \left[(A^{s-1}r_i, Ap_{i-1}^1), \ldots, (A^{s-1}r_i, Ap_{i-1}^s)\right]^{\mathrm{T}}.$$

Remark 4.3. Let $a = [a_i^1, \ldots, a_i^s]^{\mathrm{T}}$ denote the steplengths used in updating the solution vector at the ith iteration of the method. It is uniquely determined by solving

$$W_i a = m \equiv \left[(r_i, p_i^1), \ldots, (r_i, p_i^s)\right]^{\mathrm{T}}. \tag{4.1b}$$

Remark 4.4. Let R_i and P_i be the s-dimensional spaces $\{r_i, Ar_i, \ldots, A^{s-1}r_i\}$ and $\{p_i^1, \ldots, p_i^s\}$ respectively.

The following theorem guarantees the convergence of the s-CG method in at most N/s steps.

Theorem 4.1. *Let m be the degree of the minimal polynomial of r_0, and assume $m > (i+1)s$. Then the direction spaces P_i and the residuals R_i generated by the s-CG process for $i = 0, 1\ldots$ satisfy the following relations*:

(1) *P_i is A-conjugate to P_j for $j < i$,*
(2) *R_i is A-conjugate to R_j for $j < i - 1$,*
(3) *P_j, R_j, $j = 0, \ldots, i$ form bases for the Krylov subspace $V_i = \{r_0, Ar_0, \ldots, A^{(i+1)s-1}r_0\}$,*
(4) *r_i is orthogonal to V_{i-1}.*

Proof. Induction on i is used [4] or [5]. □

Proposition 4.1. *The following recurrence formula holds*:

$$\begin{aligned}(A^{(s+k)}r_i, r_{i-1}) = -\left(\frac{1}{a_{i-1}^s}\right)\Big\{&(A^k r_i, r_i) + a_{i-1}^{(s-k)}(A^s r_i, r_{i-1}) \\ &+ a_{i-1}^{(s-k+1)}(A^{(s+1)}r_i, r_{i-1}) + \cdots \\ &+ a_{i-1}^{(s-1)}(A^{(s+k-1)}r_i, r_{i-1})\Big\}\end{aligned}$$

for $k = 1, \ldots, s-1$ and $(A^s r_i, r_{i-1}) = -(r_i, r_i)/a_{i-1}^s$.

Proof. By induction and use of Theorem 4.1 [4,5]. □

The following corollary of Theorem 4.1 combined with Proposition 4.1 reduce the computation of the vectors c^j to the first s moments of r_i.

Corollary 4.1. *The right-hand side vectors $c^1, \ldots, c^s$ for the linear systems* (4.1) *become*

$$c^1 = \left[0, \ldots, 0, (r_i, A^s r_{i-1})\right]^{\mathrm{T}}$$

$$c^2 = \left[0, \ldots, 0, (Ar_i, A^{s-1}r_{i-1}), (Ar_i, A^s r_{i-1})\right]^{\mathrm{T}}$$

$$\ldots$$

$$c^s = \left[(A^{s-1}r_i, A^s r_{i-1}), \ldots, (A^{s-1}r_i, A^s r_{i-1})\right]^{\mathrm{T}}.$$

The following corollary reduces the computation of W_i to the first $2s$ moments of r_i and scalar work.

Corollary 4.2. *The matrix of inner products $W_i = (p_i^l, ap_i^j)$, $1 \leqslant l, j \leqslant s$ can be formed from the moments of r_i and the s-dimensional vectors $\boldsymbol{b}_{i-1}^1, \ldots, \boldsymbol{b}_{i-1}^s$ and $\boldsymbol{c}_i^1, \ldots, \boldsymbol{c}_i^s$.*

Proof.

$$\left(p_i^l, Ap_i^j\right) = \left(A^l r_i, A^j r_i\right) + \boldsymbol{b}_{i-1}^{l\mathrm{T}} \boldsymbol{c}_i^j. \qquad \square$$

The following corollary reduces the vector m_i to the first s moment of r_i.

Corollary 4.3. *The vector m_i can be derived from the moments.*

Proof.

$$m_i = \left[\left(r_i, p_i^1\right), \ldots, \left(r_i, p_i^s\right)\right]^{\mathrm{T}} = \left[\left(r_1, r_i\right), \ldots, \left(r_i, A^{s-1} r_i\right)\right]^{\mathrm{T}}. \qquad \square$$

We now reformulate the s-CG algorithm taking into account the results of Theorem 4.1. We will use

$$P = \left[p^1, \ldots, p^s\right], \qquad Q = \left[q^1, \ldots, q^s\right]$$

to denote the direction planes in the odd and even iterates respectively.

If K is an SPD matrix, then applying the s-CG method to $[K^{1/2} A K^{1/2}] K^{-1/2} x = K^{1/2} f$ gives rise to the following algorithm. Here $\mu^0, \ldots, \mu^{2s-1}$ denote the moments of the vector $r_i = f - Ax_i$ with respect to the matrix AK and the inner product $(\cdot, K\cdot)$. For example μ_0, μ_1, μ_2 are (r_i, Kr_i), (AKr_i, Kr_i), $((AK)^2 r_i, Kr_i)$, respectively.

Algorithm 4.2. The s-step preconditioned conjugate gradient method (s-CG).

```
Select $x_0$
Set $P = 0$
Compute $Q = [Kr_0 = K(f - Ax_0), (KA)Kr_0, \ldots, (KA)^{s-1}Kr_0]$
Compute $\mu^0, \ldots, \mu^{2s-1}$
**For** $i = 0$ **Until** Convergence **Do**
    Call Scalar Work
    **If** ($i$ even) **then**
1.  $Q = Q + P[\boldsymbol{b}^1, \ldots, \boldsymbol{b}^s]$
2.  $x_{i+1} = x_i + Q\boldsymbol{a}$
3.  $P = [Kr_{i+1} = K(f - Ax_{i+1}), (KA)Kr_{i+1}, \ldots, (KA)^{s-1}Kr_{i+1}]$
4.  Compute $\mu^0, \ldots, \mu^{2s-1}$
    **Else**
1.  $P = P + Q[\boldsymbol{b}^1, \ldots, \boldsymbol{b}^s]$
2.  $x_{i+1} = x_i + P\boldsymbol{a}$
3.  $Q = [Kr_{i+1} = K(f - Ax_{i+1}), (KA)Kr_{i+1}, \ldots, (KA)^{s-1}Kr_{i+1}]$
4.  Compute $\mu^0, \ldots, \mu^{2s-1}$
    **EndIf**
**EndFor**
```

```
Scalar Work Routine
  If ($i = 0$) then
    Form and Decompose $W_0$
    Solve $W_0 a = m_0$
  Else
    Solve $W_{i-1} b^j + c^j = 0$, $j = 1, \ldots, s$
    Form and Decompose $W_i$
    Solve $W_i a = m_i$
  EndIf
  Return
  End
```

Vector products needed to advance one iteration in the s-step CG are performed by multiplying one vector by powers of the matrix. It does not seem possible to take advantage of this fact for efficient use of local memories unless the matrix has a regular sparsity structure. We will demonstrate this possibility for a linear system arising from the numerical solution of the model problem.

In the next two sections we describe the two different choices of K which are used for the efficient theoretical implementation of the preconditioned s-CG in Section 7. In the case of the incomplete Cholesky preconditioning, we also derive a completely parallelizable form of the preconditioner. The usefulness of this new form for the efficient implementation will become clear in Section 7.

5. Incomplete Cholesky preconditioned conjugate gradient (ICCG)

If A is the matrix resulting from the symmetric 2-D model problem, then the zero entries are given by the set $P = \{(i, j): i - j \neq 0, 1, n\}$. We demand that the inverse of the IC preconditioning matrix, $K^{-1} = LDL^{\mathrm{T}}$, has zero entries given by P, and we denote the nonzero entries of K^{-1} by $\tilde{a}(i)$, $\tilde{b}(i)$, $\tilde{c}(i)$. They can be derived by the recurrences

$$\tilde{b}(i) = b(i), \qquad \tilde{c}(i) = c(i),$$

$$\tilde{a}(i) = a(i) - \frac{[\tilde{b}(i-1)]^2}{\tilde{a}(i-1)} - \frac{[\tilde{c}(i-n)]^2}{\tilde{a}(i-n)}, \quad 1 \leqslant i \leqslant n^2$$

with the undefined elements replaced by zeros. We can scale the linear system symmetrically ($[DAD](D^{-1}x) = Df$ where D is a diagonal matrix) to obtain $\tilde{a}(i) = 1$. Then

$$A = K^{-1} + R = (I - E - F)(I - E - F)^{\mathrm{T}} + R$$

where E^{T} is a matrix consisting of the upper diagonal elements $b(i)$, F^{T} of the upper diagonal elements $c(i)$, and R is the error in the approximation of K^{-1} to A.

To determine $v = Ku$ we must solve

$$(I - E - F)z = u \quad \text{and} \quad (I - E^{\mathrm{T}} - F^{\mathrm{T}})v = z$$

or, in block form

$$(I - E_j)z_j = u_j + F_j z_{j-1} \quad \text{and} \quad \left(I - E_j^{\mathrm{T}}\right)v_j = z_j + F_j^{\mathrm{T}} v_{j+1}$$

for $j = 1, \ldots, n$ to complete the forward and the backward incomplete Cholesky steps. Note that v_j, z_j are vectors of n components.

5.1. Vectorizable IC

Note that for the forward step (similarly for the backward step) there are two dependencies in the expression defining each component of the block vector z_j, one on the preceding component (of the same block j) and one on the components of the preceding block z_{j-1}. The same is true for the components of v_j. Thus the computation must be performed serially. The first dependency can be removed if we use a series expansion for $(I-E_j)^{-1}$ [17]; the equation becomes

$$z_j=\left(I+E_j+E_j^2+\cdots+E_j^m\right)\left(u_j+F_jz_{j-1}\right)$$

where $m=2$ is usually sufficient for a good approximation [17]. This computation can only be vectorized or parallelized by blocks (since a block dependency persists). Block vectorization is efficient but parallelization is problematic unless the block size is much larger than the number of processors times the length of the vector pipeline.

5.2. Parallelizable IC

In order to have a parallel preconditioner (PICCG) we must remove the block dependency. This is achieved by using a series expansion for $(I-E_j-F_j)^{-1}$. Using the first m terms of the Von Neuman series, the approximate computation of $Ku=(I-E^{\mathrm{T}}-F^{\mathrm{T}})^{-1}(I-E-F)^{-1}u$ amounts to

$$z=\left[I+(E+F)+\cdots+(E+F)^m\right]u,$$
$$v=\left[I+(E^{\mathrm{T}}+F^{\mathrm{T}})+\cdots+(E^{\mathrm{T}}+F^{\mathrm{T}})^m\right]z.$$

Next we show that the selection of a small value for m is sufficient in the truncated power series approximation of $(I-E-F)^{-1}$. We can write

$$A=\bar{K}^{-1}+S+R$$

where $\bar{K}^{-1}$ is the approximation to K^{-1} if the truncated series expansion is used. Following the analysis [17] we now show that the norm of the matrix S (truncation error in Von Neuman series) is comparable to the norm of R (approximation error in incomplete Cholesky decomposition), so that we can expect that the preconditioner is effective. If we write $W=E+F$, then

$$(I+W+\cdots+W^m)^{-1}=(I-W)(I-W^{m+1})^{-1}.$$

It follows that $\bar{K}^{-1}=(I-W)(I-W^{m+1})^{-1}[(I-W)(I-W^{m+1})^{-1}]^{\mathrm{T}}$. By simple manipulation we get

$$(I-W)(I-W^{m+1})^{-1}=I-W+(I-W)W^{m+1}(I-W^{m+1})^{-1}=I-W+\bar{S}.$$

Therefore the matrix S can be expressed as

$$S=-\bar{S}(I-W)^{\mathrm{T}}-(I-W)\bar{S}^{\mathrm{T}}-\bar{S}\bar{S}^{\mathrm{T}}.$$

The matrix $I-W$ is bounded: $\|I-W\|\leqslant 1+\beta+\gamma$, where $\beta=\max\{b(i)\}$, $\gamma=\max\{c(i)\}$. We can now obtain a bound on the norm of the matrix $\bar{S}$ for small β and γ:

$$\|\bar{S}\|\leqslant(1+\beta+\gamma)(\beta+\gamma)^{m+1}\left(1-(\beta+\gamma)^{m+1}\right)^{-1}\approx(1+\beta+\gamma)(\beta+\gamma)^{m+1}.$$

Neglecting the $\bar{S}\bar{S}^{\mathrm{T}}$ term we obtain

$$\|S\|\leqslant 2(1+\beta+\gamma)^2(\beta+\gamma)^{m+1}.$$

The set of zero entries of R contains the set $P=\{(i,\ j):\ |i-j|\neq n-1\}$. Moreover, the two nonzero diagonals have $b(i)c(i-1)$ as entries. Hence $\|R\|=2\max_i|b(i)c(i-1)|$.

Suppose A is the 5-point difference operator obtained from the discretization of the 2-D Poisson equation with Dirichlet boundary conditions on the unit square. Then $b(i) = c(i) = 0.25$, and $\|R\| \approx 0.15$. The bound on $\|S\|$ is ≈ 0.57 for $m = 2$. Thus the bound on the truncation error is comparable to the approximation error made using the incomplete factorization. This shows that the parallelizable incomplete Cholesky preconditioning is effective.

The PICCG for $m = 2$ is worth using for the 2-D model problem if the number of steps needed for convergence is no more than half that of plain CG. This follows because $16N$ ops are needed to form $\bar{K}u$ compared to the $19N$ ops of one step of plain CG. This is the case in the test problems that we include here. Since $(I + W + W^2 + W^3)u$ can be computed as $(I + W^2)(I + W)u$ which needs $19N$ ops for the 2-D case, we can choose $m = 3$ in the series truncation.

6. Polynomial preconditioned CG

If polynomial preconditioning is used, then $K = q(A)$, where $q(A)$ is a polynomial approximation to the inverse of the matrix A [10]. A stable way to compute Kr_i is

$$\left(\prod_{j=1}^{k} (A - \rho_j)\right) r$$

where ρ_j are the roots of the degree k polynomial $q(\lambda)$.

One choice for q is the Chebyshev polynomial with roots in the interval $[\lambda_{\min}, \lambda_{\max}]$. Then

$$q_k(\lambda) = \frac{(1 - c(\lambda)/c(0))}{\lambda}$$

where

$$c(\lambda) = T\left(\frac{\lambda_{\max} + \lambda_{\min} - 2\lambda}{\lambda_{\max} - \lambda_{\min}}\right)$$

where $T(\lambda)$ is the Chebyshev polynomial of degree $k + 1$. This choice of polynomial preconditioner minimizes the

$$\|I - q(A)A\| = \max_{\lambda_i \in \sigma(A)} |1 - \lambda_i q(\lambda_i)|$$

where $\sigma(A)$ is the spectrum of A. This requires an estimate of the spectrum of A. The choice of a polynomial can also be based on minimizing other norms. Johnson et al. [10] have reported tests where polynomial preconditioning was used based on minimization of the L_2-norm of $|1 - \lambda_i q(\lambda_i)|$ over the interval $[\lambda_{\min}, \lambda_{\max}]$ with respect to a weight function w; they report results comparable to that obtained using the Chebyshev polynomials.

When both CG and polynomial PCG are iterated to convergence, the residual polynomials generated by the two methods should have comparable degrees for polynomial PCG to be efficient. This means that approximately the same number of matrix-vector products must be performed in either case. On a parallel system, the inner products, vector updates and matrix-vector products may be running at different speeds. Thus polynomial PCG may involve more matrix-vector products than CG and still be faster.

7. Implementation of preconditioned s-CG with efficient use of local memory

In this section we show how the different parts of s-CG can be implemented efficiently on a vector processor with local memory. Such a system can have either a 'register-to-register' (e.g.

Table 2
Ops for 5 steps of CG (versus one step of 5-CG) and (mem. transfers)/ops

Vector operation	5-CG/ops	5-CG/ratio	CG/ops	CG/ratio
Dotproduct	$20N$	11/20	$20N$	1
Linear combination	$60N$	16/70	$30N$	3/2

ALLIANT FX/8 or CRAY-2) or a 'memory-to-memory' (e.g. ETA-10) organization. In the first case the functional units are supplied with operands from the registers; in the second case, operands are brought directly from the memory of the system. Although all computations in one iteration can be performed with one sweep through the data we will consider separately the three parts (inner products, linear combinations, matrix-vector products) because it is easier to implement and compare to CG. We consider only the case $s = 5$.

7.1. Linear combinations

For the linear combinations we partition the vectors $p_i^1, \ldots, p_i^s$, $p_{i-1}^1, \ldots, p_{i-1}^s$ and x_{i-1}, x_i into equal subvectors of length m such that $(2s + 2)m \leqslant$ local memory size. The DO loop for all the linear combinations consists of an outer loop of N/m steps and an inner loop of m steps. By using matrix notation we can describe this as follows (where the subscript k designates a horizontal section of length m):

```
**Do** $k = 1,\ N/m$
    $P_k = P_k + Q_k[\boldsymbol{b}^1, \ldots, \boldsymbol{b}^s]$
    $(x_{i+1})_k = (x_i)_k + P_k \boldsymbol{a}$
**EndDo**
```

7.2. Inner products

The ten inner products

$$(r_i,\ Kr_i),\ (AKr_i,\ Kr_i),\ \left((AK)^2 r_i,\ Kr_i\right), \ldots, \left((AK)^4 r_i,\ (KA)^4 Kr_i\right),$$
$$\left((AK)^5 r_i,\ (KA)^4 Kr_i\right)$$

are computed. Note that the ratio for executing the inner products is about 1/2 because 11 vector references are needed. We must compute 10 inner products involving 11 vectors efficiently using the local memory. We partition the vectors in N/m equal subvectors of length m. We need 10 subvectors (of length m, residing in the local memory throughout the computation) holding the partial results of the inner products. Thus $(11m + 10m) \leqslant$ local memory size. The DO loop for all the inner products consists of an outer loop of N/m steps and an inner loop of m steps computing simultaneously the 10 inner products. In Table 2 we show the work and ratio of global memory references per ops for the inner products and linear combinations of one iteration of 5-CG and five iterations of CG. Although the number of ops doubles for the linear combinations the data locality is improved.

7.3. Matrix-vector products

The multiplication by A can be written in vector form:

$$\begin{aligned} Ap(i) &= c(i-n) * p(i-n) + c(i) * p(i+n) + b(i-1) * p(i-1) \\ &\quad + b(i) * p(i+1) + a(i) * p(i). \end{aligned}$$

Assuming $s = 5$, we need to form the following matrix vector products:

$$Kr_i,\ (KA)Kr_i,\ (KA)^2 Kr_i,\ (KA)^3 Kr_i,\ (KA)^4 Kr_i,\ A(KA)^4 Kr_i.$$

(a) *Polynomial preconditioning:* Multiplication by the preconditioning matrix amounts to $(\prod_{j=1}^{\mu}(A-\rho_j))u$. We can combine the multiplication by A and one factor of K or two factors of K and keep all the data in the local memory. Let a horizontal section of order n be the submatrix $A_k=[C_{i-1}, T_k, C_k]$, $k=1,\ldots,n$. Also, let u_k, v_k be subvectors (of order n) of u, v corresponding to the block A_k. If the local memory can simultaneously accommodate two full sections of A and seven full subvectors, we can carry out the computation

```
$v_1 = A_1 u_1$
**Do** $k = 1,\ n-1$
    $v_{k+1} = (A_{k+1} - \rho)[u_k^{\mathrm{T}}, u_{k+1}^{\mathrm{T}}, u_{k+2}^{\mathrm{T}}]^{\mathrm{T}}$
    $w_k = (A_k - \tau)[v_{k-1}^{\mathrm{T}}, v_i^{\mathrm{T}}, v_{k+1}^{\mathrm{T}}]^{\mathrm{T}}$
**EndDo**
**Compute** $w_n$
```

(where ρ, $\tau=0$, or ρ_j), while keeping the matrix in the local memory. If these blocks do not fit in the memory, they must be further sectioned.

This idea can be generalized to do Au, $(A-\rho_1)Au,\ldots,(\prod_{j=1}^{k}(A-\rho_j))Au$ while the required entries of A reside in the local memory. However, $k+1$ sections of the matrix and $3(k+1)$ subvectors must fit in the local memory. This can be useful even on a sequential machine when the two levels of memory that must be used efficiently are the main memory and a slow secondary storage device.

The same idea can be applied to the 3-D problem. Here the horizontal sections of order n^2 are: $A_k=[D_k, \overline{T}_k, D_k]$, $1\leqslant k\leqslant n$. For a reasonable resolution without need of secondary storage, $n^3=10^6$ (because of the main memory limits), so $n^2=10^4$ and a good portion of a section can be kept in a local memory of size 16k (CRAY-2, ALLIANT FX/8).

(b) *Incomplete Cholesky preconditioning:* The vectorizable IC involves the forward and backward incomplete Cholesky steps

$$\left(I-E_j\right)z_j=u_j+F_j z_{j-1} \quad\text{and}\quad \left(I-E_j^{\mathrm{T}}\right)v_j=z_j+F_j^{\mathrm{T}}u_{j+1}$$

which cannot be combined (keeping the matrices in local storage) because the forward step must be completed before the backward step can start. Thus we can only combine either the backward step (of the multiplication by K) with the multiplication by A in forming the $(KA)^j r$. The parallelizable IC involves multiplication (of a vector) by the uni-diagonal matrices E, E^{T}, F, F^{T}. Since the nonzeros of these matrices are the entries of A, the computation can be organized in a way similar to the Doloop of the polynomial preconditioning. Multiplication by A and K will be combined to read the data once from the main memory.

8. The experimental environment

The experiments were conducted on the ALLIANT FX/8 multiprocessor system at the Center for Supercomputing Research and Development of the University of Illinois.

The FX/8 is an example of a supercomputer architecture with memory hierarchy. The configuration of the FX/8 contains 8 Computational Elements (CEs), which communicate to each other via a concurrency control bus used as a synchronization device. Each CE has a computational clock cycle of 170 ns. The maximum performance of one CE is 11 Mflops (million ops/s) for single precision and 5.9 Mflops for double precision computations. Thus when the 8 CEs run concurrently the peak performance can reach 47.2 Mflops. Each CE is connected via a crossbar switch to a shared cache of 16k (64 bit) words, implemented in four quadrants. This connection is interleaved and provides a peak bandwidth of 47.12 MW/s. The cache is connected to an 8 MW interleaved global memory via a bus with a bandwidth of about 23.5 MW/s for sequential read and about 19 MW/s for sequential write access. The system also has 6 interactive processors (IPs) used for operating system related functions and I/O operations.

Each CE is a pipelined vector processor with eight 64-bit vector registers of length 32, eight 64-bit scalar registers, as well as eight address registers. Operands for vector instructions come from vector registers or vector and scalar registers. The vector multiply and add instructions can be overlapped. It is worth noting that the vector multiply takes two cycles per element while the vector add/subtract/convert take one cycle and division take eight cycles. Multiprocessing is realized by concurrency instructions which permit a loop to be executed concurrently across more than one processor in an interleaved mode or by concurrent call of a subroutine which assign one task to each CE.

The ALLIANT FX/8 optimizer and compiler restructure a FORTRAN code based on data dependency analysis for scalar, vector, and concurrent execution. A FORTRAN program can execute in one of the following modes: scalar, vector, scalar concurrent, vector-concurrent, or concurrent-outer/vector inner. We illustrate the modes of execution on the following loop:

DO 1 $I = 1, N$
1 $A(I) = A(I) + S$

For $N = 8192$ we have

- *Scalar:* $A(1), A(2), \ldots, A(8192)$,
- *Vector:* $A(1:32), A(33:64), \ldots, A(8161:8192)$.
- *Concurrent:*

  CE_1: $A(1), A(9), \ldots, A(8185)$
  CE_2: $A(2), A(10), \ldots, A(8186)$
  ...
  CE_8: $A(8), A(16), \ldots, A(8192)$,
- *Vector concurrent:*

  CE_1: $A(1:249:8), \ldots, A(7937:8185:8)$
  CE_2: $A(2:250:8), \ldots, A(7938:8186:8)$
  ...
  CE_8: $A(1:256:8), \ldots, A(7944:8192:8)$,
- *Concurrent outer/vector inner:*

  CE_1: $A(1:32), \ldots, A(992:1024)$
  CE_2: $A(1025:1056), \ldots, A(2017:2048)$
  ...
  CE_8: $A(7169:7200), \ldots, A(8161:8192)$.

There is a timing facility which is accessible via a FORTRAN subroutine call and seems to give stable results with a resolution of 10^{-1} s measurements. The computational rates which will be presented were run with a large serial (non-concurrent) outer loop in order to obtain reliable timing data. To do time measurements the programs were run more than three times. Although the execution was not in single user mode no other sizable jobs were running at the same time, and the timing variations were of the order of one percent.

9. Computational rates of CG and s-CG parts

Table 3 contains the rates of the different parts of the computation in one iteration of the 5-step and standard CG respectively. All the rates cited are for vector lengths between 10^4 and

Table 3
Computational rates for the 5-CG and CG parts

Vector operation	5-CG/Mflops	CG/Mflops
Vector update	–	5.5
Dotproduct	15	8
Norm	–	14
Matrix-vector (A)	8	8
Matrix-vector (K)	7	7
Linear combination	22	–

10^5. The matrix vector product runs at ≈ 14 Mflops from the cache. However for sizes of the matrix A exceeding the cache capacity this rate is ≈ 8 Mflops. The implementation discussed in Section 7 was only followed for the linear combinations and the inner products. Combining the matrix-vector operations Av and A^2v with proper cache management did not produce rates higher than 8 Mflops. Thus this part of the theoretical implementation was not realized. The low rate of the matrix product (even from the cache) is mainly due to the low rate of the vector product operation. This part, involving half the ops of CG applied to the model problem, has not been sped up and it is a slow part. Thus, the speedup of 5-CG compared to CG is not expected to be very good, taking into account the fact that there is an extra matrix-vector product (in 5-CG) per five steps of CG.

It is worth noting that the linear combinations run at 16 Mflops if vector concurrent mode is used and at 22 Mflops if concurrent outer/vector inner mode is used. Since there are twice as many ops in the linear combinations as in the vector updates and the rate for the linear combinations is four times that of the vector updates, we have sped up this part by a factor of two.

The inner products were computed by assigning each one of eight inner products to one processor and computing each one of the two remaining norms separately. If all ten inner products are carried out together in concurrent outer/vector inner mode, then this may yield a higher rate. It is worth noting that there is a primitive function DOTPRODUCT, which is designed to compute only one inner product at a time in vector mode on a single processor, or vector concurrent mode. The inner products rate may have been higher if two inner products were computed by keeping the same data in the vector registers. Since the rate for the two inner products for CG is 11 Mflops, we have sped up this part by only about fifty percent.

10. Test problems

We include test results on two problems with the matrix being the Laplace operator for the standard and 5-step CG, and the vectorizable preconditioned standard and 5-step ICCG. We also include test results for the parallelizable preconditioned ICCG. The first problem is a discretized PDE with known solution. The second problem is a linear system. The matrix was stored in three diagonals of order N to simulate the general five-point difference operator.

Problem 1. $-(au_x)_x - (bu_y)_y = g$ on the unit square with homogeneous boundary conditions and $a \equiv b \equiv 1$ and $u(x, y) = e^{xy} \sin(\pi x) \sin(\pi y)$.

Problem 2. The linear system $Az = f$ where A is the pentadiagonal matrix of problem 1 and $A^{-1}f(i) = \sqrt{i}$.

Table 4
Execution times for the CG on Problem 1 and Problem 2

$\sqrt{N}$	Problem 1		Problem 2	
	Steps	Time/s	Steps	Time/s
64	136	1.08	196	1.5
100	209	4.66	307	7.2
128	266	10.21	395	15.66
160	331	21.26	496	32.28
200	412	41.39	621	62.26
256	525	92.2	797	140.51
300	613	145.01	936	221.22

Table 5
Execution times for the 5-CG Problem 1 and Problem 2

$\sqrt{N}$	Problem 1		Problem 2	
	Steps	Time/s	Steps	Time/s
64	27	1.1	39	1.58
100	42	4.24	62	6.13
128	53	8.68	79	13.19
160	66	16.92	99	25.1
200	83	32.91	124	49.26
256	107	70.48	160	105.62
300	123	110.98	187	168.15

Table 6
Execution times for the VICCG on Problem 1 and Problem 2

$\sqrt{N}$	Problem 1		Problem 2	
	Steps	Time/s	Steps	Time/s
64	52	1.08	75	1.45
100	72	3.44	115	5.18
128	90	7.17	147	11.37
160	111	14.02	182	22.56
200	137	25.32	217	39.14
256	174	52.26	276	85.23
300	203	84.62	324	133.8

Table 7
Execution times for the 5-VICCG on Problem 1 and Problem 2

$\sqrt{N}$	Problem 1		Problem 2	
	Steps	Time/s	Steps	Time/s
64	11	1.27	16	1.54
100	15	3.4	23	5.45
128	18	6.58	30	10.47
160	22	11.64	37	19.43
200	28	22.15	44	34.98
256	35	45.74	55	72.2
300	41	75.55	65	120.01

Table 8
Execution times for the PICCG Problem 1 and Problem 2

$\sqrt{N}$	Problem 1		Problem 2	
	Steps	Time/s	Steps	Time/s
64	56	0.99	85	1.61
100	85	4.31	132	6.85
128	107	9.29	160	14.18
160	133	18.4	198	27.97
200	165	36.07	247	55.5
256	209	75.39	316	115.3
300	245	122.96	371	189

The termination criterion used for preconditioned CG was $(r_i, Kr_i)^{1/2} < 10^{-6}$ and $(r_i, r_i)^{1/2} < 10^{-6}$ for all the other cases.

Tables 4–7 contain test results for CG, 5-CG, VICCG and 5-VICCG. The dimension N of the problems varied from 64^2 to 300^2. The number of iterations and execution time were measured for each dimension. It is worth noting that the number of CG (VICCG) iterations is five times the 5-CG (5-VICCG) iterations applied to the same problem of the same size. This verifies Theorem 4.1. We can compute the speedup factor for the various methods from the results in Tables 4–7. These factors are

$$\text{CG/5-CG} \approx 1.3, \qquad \text{VICCG/5-VICCG} \approx 1.15.$$

The matrix-vector multiply by K has a rate of about 7 Mflops and $12N$ ops and in the preconditioned CG the slow parts involve about seventy percent of the total number of ops and this accounts for the drop in the speedup factor.

Table 8 shows the performance of the proposed Parallel ICCG method (PICCG). Comparing it to the CG method (both in number of steps and execution times), we conclude that it is a reasonably good preconditioner. The performance of the vectorizable ICCG is far better mainly because PICCG involves $16N$ ops (to compute Kv) compared to $12N$ for VICCG for the truncated series approximation with $m = 2$. Also, block parallelization is sufficient for eight CEs with vector register length 32. For example for $n = 256$, there is a complete set of data for all CEs if parallelization is carried out block-wise. Finally interprocessor communication is not a problem because of the fast shared cache and the fact that multiplication by the preconditioner K is slow (7 Mflops).

11. Conclusions

We have presented the s-step preconditioned conjugate gradient method for symmetric and positive definite systems of linear equations. For the incomplete Cholesky and the polynomial preconditioning we showed theoretically how this method can be efficiently implemented on a system with memory hierarchy. Numerical tests on linear systems arising from the discretization of elliptic PDEs show that on a multiprocessor system with memory hierarchy the s-step preconditioned CG (for $s = 5$) is faster than the standard preconditioned CG. The implementation of the s-step preconditioned CG on message passing architectures remains to be done.

We have extended the vectorizable incomplete Cholesky preconditioning [17] to a parallelizable preconditioning (PICCG). A theoretical analysis of the error in truncating the two power series involved in the multiplication by the preconditioning matrix shows that keeping the first two or three terms is sufficient for the preconditioning to be effective. Tests included here showed that PICCG is effective in terms of reducing the work versus the plain CG for the 2-D problem. However the vectorizable ICCG was faster on the parallel system (with eight processors) used for the experiments. This is because there is an overhead in ops involved in PICCG versus VICCG. However the VICCG has high communication overhead on massively parallel systems. Further tests on parallel systems with many processors are required to determine the advantages PICCG.

Acknowledgment

We thank W. Jalby of INRIA/CSRD University for all his advice in the efficient implementation of s-CG on the ALLIANT FX/8 and professor A. Sameh for providing access to the computing facilities of the CSRD of the University of Illinois.

References

[1] L. Adams, m-Step preconditioned conjugate gradient methods, *SIAM J. Sci. Statist. Comput.* **6** (2) (1985).

[2] D. Barkai, K.J.M. Moriarty and C. Rebbi, A modified conjugate gradient solver for very large systems, in: *Proc. 1985 IEEE International Conference on Parallel Processing* (1985) 284–290.

[3] P. Concus, G.H. Golub and G. Meurant, Block preconditioning for the conjugate gradient method, *SIAM J. Sci. Statist. Comput.* **6** (1) (1985).

[4] A.T. Chronopoulos and C.W. Gear, s-Step iterative methods for symmetric linear systems, *J. Comput. Appl. Math.* **25** (1) (1989) 153–168.

[5] A.T. Chronopoulos, A class of parallel iterative methods implemented on multiprocessors, Ph.D. Thesis, Technical Report UIUCDCS-R-86-1267, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

[6] J.J. Dongarra and D.C. Sorensen, Linear algebra on high-performance computers, in: M. Feilmeier, G. Joubert and U. Schendel, eds., *Parallel Computing 85* (North-Holland, Amsterdam, 1985).

[7] P.F. Dubois, A. Greenbaum and G.H. Rodrigue, Approximating the inverse of a matrix for use in iterative algorithms on vector processors, *Computing* **22** (1979) 257–268.

[8] M. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Standard* **49** (1952) 409–436.

[9] W. Jalby and U. Meier, Optimizing matrix operations on a parallel multiprocessor with a memory hierarchy, in: *Proc. International Conference on Parallel Processing* (1986).

[10] O.G. Johnson, C.A. Micchelli and G. Paul, Polynomial preconditioners for conjugate gradient calculations, *SIAM J. Numer. Anal.* **20** (2) (1983).

[11] T.L. Jordan, A guide to parallel computation and some CRAY-1 experiences, in: G. Rodrigue, ed., *Parallel Computations* (Academic Press, New York, 1982).

[12] G. Meurant, The block preconditioned conjugate gradient method on vector computers, *BIT* **24** (1984) 623–633.

[13] D.P. O'Leary, The block conjugate gradient algorithm and related methods, *Lin. Alg. Appl.* **29** (1980) 293–322.

[14] Y. Saad, Practical use of polynomial preconditionings for the conjugate gradient method, *SIAM J. Sci. Statist. Comput.* **6** (4) (1985).

[15] Y. Saad, On the Lanczos method for solving symmetric linear systems with several right-hand sides, *Math. Comp.* **48** (178) (1987).

[16] R. Varga, *Matrix Iterative Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1962).

[17] H.A. Van Der Vorst, A vectorizable variant of some ICCG methods, *SIAM J. Sci. Statist. Comput.* **3** (3) (1982).

[18] H.A. Van Der Vorst, The performance of FORTRAN implementations for preconditioned conjugate gradients on vector computers, *Parallel Comput.* **3** (1986) 49–58.

[19] J. Van Rosendale, Minimizing inner product data dependencies in conjugate gradient iteration, in: *Proc. IEEE International Conference on Parallel Processing* (1983).