Journal of Computational and Applied Mathematics 25 (1989) 153–168
North-Holland

s-step iterative methods for symmetric linear systems *

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 15 February 1988
Revised 16 June 1988

Abstract: In this paper we introduce s-step Conjugate Gradient Method for Symmetric and Positive Definite (SPD) linear systems of equations and discuss its convergence. In the s-step Conjugate Gradient Method iteration s new directions are formed simultaneously from $\{r_i, Ar_i, \ldots, A^{s-1}r_i\}$ 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.

Keywords: Iterative methods, s-step, conjugate gradient, convergence.

1. Introduction

Accurate numerical solution of mathematical problems derived from modeling physical phenomena often requires a capacity of computer storage and a sustained processing rate that exceed the ones offered by the existing supercomputers. Such problems arise from oil reservoir simulation, electronic circuits, chemical quantum dynamics and atmospheric simulation to mention just a few.

There is an enormous amount of data that must be manipulated to solve these problems with a reasonable accuracy. These data are stored in slower memory layers (shared memory vector multiprocessors) or in the private memory of each processor for the message passing machines.

Memory contention on shared memory vector multiprocessor systems constitutes a severe bottleneck for achieving their maximum performance. The same is true for global communication cost on a message passing system. Thus numerical algorithms should not only be suitable for

* 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.

vector and parallel processing but they must provide good data locality. That is the organization of the algorithm should be such that data can be kept as long as possible in fast registers or local memories and have many arithmetic operations performed on them. This means that the

$$Ratio = (Memory\ References)/(Floating\ Point\ Operations).$$

must be as low as possible. For example vector operations like the vector updates

$$v \leftarrow v + cu$$

with a ratio $\frac{3}{2}$ may yield worse performance than linear combinations

$$v + \sum_{i=1}^{k} c_i u_i, \quad k \geqslant 2.$$

which provide a lower ratio of $(k-2)/2k,\ k \geqslant 2$.

Iterative methods are an efficient way to obtain a good numerical approximation to the solution of $Ax = b$ when the matrix A is large and sparse. The Conjugate Gradient (CG) method [14] is a widely used iterative method for solving such systems when the matrix A is symmetric and positive definite. Generalizations of CG exist for nonsymmetric problems.

In an *s-step* generalization of an iterative method, s consecutive steps of the one-step method are performed simultaneously. This means, for example, that the inner products (needed for s steps of the one-step method) can be performed simultaneously and the vector updates are replaced by linear combinations.

In this paper we introduce an s-step Conjugate Gradient Method and an s-step Conjugate Residual Method and discuss their convergence. The computational work and storage increase slightly (for the s-step symmetric methods) compared to their one-step counterparts. However, their parallel properties and data locality are improved so that the s-step methods are expected to have superior performance on vector and parallel systems. This is because the s-step method can be organized so that only sweep through the data per iteration is required and the $2s$ inner products required for one s-step iteration are executed simultaneously.

We should point out that the s-step CG presented here is different from two iterative methods with which it may seem to overlap in the goals achieved. These methods are the block CG [17] and the Lanczos algorithm for solving linear systems [20,13].

The block CG is used to solve $AX = B$ with dimension $X = N \times m$. This, for example, is the case when CG is used to solve $Ax = b$ for many (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 an orthonormal basis $V_m = [v_1, \ldots, v_m]$ is built for the Krylov space $\{r_0, Ar_1, \ldots\}$ starting from the residual vector $r_0 = b - Ax$. 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 [20]. The Lanczos algorithm forms serially the vectors v_j, $j = 1, \ldots, m$ using a matrix multiply with the preceding vector and two inner products. Thus it has the same shortcomings for parallel processing as the standard CG method.

Next we review different formulations of the standard Conjugate Gradient method. Then we present an s-step Steepest Descent method. In Sections 4 and 5 we derive s-step formulations for the Conjugate Gradient, the Conjugate Residual. In Section 6 we discuss the restriction on s to avoid severe orthogonality loss between the direction subspaces. In Section 7 and 8 we present numerical tests and conclusions.

2. The conjugate gradient method

Next we present the conjugate gradient method in three different forms. Algorithms 2.2, 2.3 are stable modifications of the original algorithm [14] but they are more suitable for vector and parallel processing and do better memory management.

Algorithm 2.1. The conjugate gradient method (CG).
Choose x_0
$p_0 = r_0 = f - Ax_0$
For $i = 0$ **Until** Convergence **Do**
1. Compute and Store Ap_i
2. Compute (p_i, Ap_i)
3. $a_i = (r_i, r_i)/(p_i, Ap_i)$
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}, r_{i+1})
7. $b_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$
8. $p_{i+1} = r_{i+1} + b_i p_i$
EndFor.

Storage is required for the entire vectors x, r, p, Ap and maybe the matrix A. Note that step 3 (or step 6) must be completed before the rest of the computations in the same step can start. This forces double access of vectors r, p, Ap from the main memory at each CG step.

Algorithm 2.2
Choose x_0
$p_0 = r_0 = f - Ax_0$
Compute and Store Ap_0
$a_0 = (r_0, r_0)/(Ap_0, p_0)$, $b_0 = 0$
For $i = 1$ **Until** Convergence **Do**
1. $x_i = x_{i-1} + a_{i-1} p_{i-1}$
2. $r_i = r_{i-1} - a_{i-1} Ap_{i-1}$
3. $p_i = r_i + b_{i-1} p_{i-1}$
4. Compute and Store Ap_i
5. Compute (Ap_i, Ap_i), (p_i, Ap_i), (r_i, r_i)
6. $a_i = (r_i, r_i)/(p_i, Ap_i)$
7. $b_i = (a_i^2(Ap_i, Ap_i) - (r_i, r_i))/(r_i, r_i)$
EndFor.

Computationally the only difference between Algorithms 2.1 and 2.2 is the computation of b_i. Assuming that fast local storage for sections of vectors exists, steps 1–5 can be performed with one read of the data from the main memory. The scalars computed in steps 6 and 7 are used in the next iteration. Also, the inner products needed in a single iteration can be executed simultaneously.

In the Algorithm 2.2, which can be found in [16], [19] and [3], three inner products are required for stability reasons. Note that (r_i, r_i) could be computed before computing r_i by use of the formula $(r_i, r_i) = a_{i-1}^2(Ap_{i-1}, Ap_{i-1}) - (r_{i-1}, r_{i-1})$ but the resulting algorithm would not be stable [19]. It has been in general observed that precomputing inner products involving the vectors p_i, r_i by using recursion formulas based only on inner products of p_j, r_j, $j = 0, \ldots, i-1$ may lead to unstable algorithms. Van Rosendale [21] derived such recursive "look-ahead" formulas for the CG method.

Next we present another modification of Algorithm 2.1, which is stable based on our experiments. Unlike Algorithm 2.2 two inner products are computed per iteration but an additional vector update is required. Also, no inner product is precomputed before the required vectors p_i, r_i become available.

Algorithm 2.3
Choose x_0
$p_0 = r_0 = f - Ax_0$
Compute and Store Ar_0
$a_0 = (r_0, r_0)/(Ar_0, r_0)$, $b_{-1} = 0$
For $i = 0$ **Until** Convergence **Do**
1. $p_i = r_i + b_{i-1}p_{i-1}$
2. $Ap_i = Ar_i + b_{i-1}Ap_{i-1}$
3. $x_{i+1} = x_i + a_i p_i$
4. $r_{i+1} = r_i - a_i Ap_i$
5. Compute and Store Ar_{i+1}
6. Compute (r_{i+1}, r_{i+1}), (Ar_{i+1}, r_{i+1})
7. $b_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$
8. $a_{i+1} = (r_{i+1}, r_{i+1})/[(Ar_{i+1}, r_{i+1}) - (b_i/a_i)(r_{i+1}, r_{i+1})]$

EndFor.

We have used the identity

$$(Ap_i, p_i) = (Ar_i, r_i) - (b_{i-1}/a_{i-1})(r_i, r_i).$$

For the Conjugate Residual equivalent no such increase occurs. Storage is required for the entire vectors x, r, p, Ap, Ar and maybe the matrix A. Assuming that fast local storage for sections of vectors exists, steps 1–5 can be performed with one read of the data from the main memory. The scalars computed in steps 7 and 8 are used in the next iteration. Also, the inner products can be executed simultaneously.

Algorithm 2.3 is a variant of CG (or the CR equivalent) and seems more promising than Algorithm 2.1 for parallel processing because the two inner products required to advance each step can be executed simultaneously. Also, one sweep through the data is required allowing better management of slower memories. This algorithm is the s-CG algorithm for $s = 1$.

Next we will try to generalize this to an algorithm which does one memory sweep per s steps.

3. An s-dimensional steepest descent method

Solving an SPD $Ax = f$ linear system of equations using the CG method is equivalent to minimizing a quadratic function

$$E(x) = (x-h)^{\mathrm{T}} A (x-h)$$

where $h = A^{-1} f$ is the solution of the system. This error functional is also minimized at each CG iteration by the choice of the direction vector and the steplength. Here we will examine the possibility of forming direction spaces instead of single direction vectors (as in CG), and minimizing the error functional over each space.

Definition 3.1. The s-dimensional affine space

$$L_i^s = \left\{ x_i + \sum_{j=0}^{s-1} a_j A^j r_i : a_j \text{ scalars and } r_i = f - Ax_i \right\}$$

will be called the *s-dimensional space of steepest descent* of $E(x)$ at x_i.

Since A is not derogatory, $r_i, Ar_i, \ldots, A^{s-1} r_i$ are linearly independent as long as the minimal polynomial of r_i has degree greater than s. In the *optimum s-gradient method* for minimizing the $E(x)$, the point x_{i+1} is defined to be the unique point in the space L_i^s for which $E(x)$ assumes a minimum. Existence and uniqueness follows from the positive definiteness of A. This method has been described and analyzed in [4], [15] and [12].

Algorithm 3.1. The optimum s-gradient method
x_0, $r_0 = f - Ax_0$
For $i = 0$ **Until** Convergence **Do**
  $x_{i+1} = x_i + a_i^1 r_i + \cdots + a_i^s A^{s-1} r_i$
  Select a_i^j to minimize $E(x)$ over L_i^s
  $r_{i+1} = r_i - a_i^1 A r_i - \cdots - a_i^s A^s r_i$ or, $r_{i+1} = f - Ax_{i+1}$
EndFor

Since x_{i+1} minimizes $E(x)$ over the s-dimensional space L_i^s and r_{i+1} is the gradient of $E(x)$ at x_{i+1} it is necessary and sufficient that r_{i+1} be orthogonal to this space. Equivalently, r_{i+1} must be orthogonal to $\{r_i, A^1 r_i, \ldots, A^{s-1} r_i\}$. Thus $a_i^1, \ldots, a_i^s$ can be determined by the s conditions

$$\begin{aligned} &(r_i, r_i) + a_i^1 (r_i, Ar_i) + \cdots + a_i^s (r_i, A^s r_i) = 0, \qquad \ldots, \\ &(A^{s-1} r_i, r_i) + a_i^1 (A^{s-1} r_i, Ar_i) + \cdots + a_i^s (A^{s-1} r_i, A^s r_i) = 0. \end{aligned}$$

Definition 3.2. For $k = 0, \pm 1, \pm 2, \ldots$, let the *moments* μ_i^k of r_i be defined by

$$\mu_i^k = r_i^{\mathrm{T}} A^k r_i.$$

The parameters $a_i^1, \ldots, a_i^s$ can be determined by solving the $s \times s$ system of the "normal equations". Since $(A^p r_i, A^q r_i) = (r_i, A^{p+q} r_i) = \mu_i^{p+q}$, this system has the form

$$\mu_i^0 + \mu_i^1 a_i^1 + \cdots + \mu_i^s a_i^s = 0,$$
$$\mu_i^1 + \mu_i^2 a_i^1 + \cdots + \mu_i^{s+1} a_i^s = 0,$$
$$\vdots$$
$$\mu_i^{s-1} + \mu_i^s a_i^1 + \cdots + \mu_i^{2s-1} a_i^s = 0.$$

The matrix of this system is the matrix of the moments of r_i

$$M_i = \left(\mu_i^{j+k}\right), \quad 1 \leqslant j,k \leqslant s$$

M_i is symmetric positive definite as long as $r_i, \ldots, A^{s-1} r_i$ are linearly independent. Then $a_i^1, \ldots, a_i^s$ are uniquely determined. Furthermore, $a_i^s \neq 0$ because of the assumption that the minimal polynomial of r_i has degree greater than s.

Note that the optimum s-gradient method is a steepest descent method and that the first iterate is (in exact arithmetic) equal to the sth iterate of the CG method. The work for a single step is $4sN$ multiplications and $4sN$ additions and s matrix vector products and $O(s^3)$ operations to solve for the a_i^j. The storage is $s+1$ vectors and maybe the matrix A. This contrasts with the $5sN$ multiplications and $5sN$ additions and s matrix vector products needed for the s steps of CG.

Although in the past the optimum s-gradient method has been compared to CG [15] our tests show behaviour analogous to one dimensional steepest descent methods. This is plausible because no sequence of conjugate directions was formed. It should be noted that the condition number of the matrix of moments increases prohibitively when $s > 10$.

The optimum s-gradient method is attractive for parallel processing because we can perform the matrix vector products one after another without halting to calculate parameters. Inner products for one iteration can be carried out together or coupled with the matrix vector products. Finally linear combinations involving more than two vectors have replaced the vector updates.

Next we try to generalize the optimum s-gradient method to an s-dimensional conjugate gradient method.

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

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 $\{r_0, \ldots, A^{is} r_0\}$. Then 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. Note that at each iteration the new residual is computed directly ($r_i = f - Ax_i$). This is because we never compute the vectors Ap_i^j which is used in computing r_{i+1} from r_i.

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**
Select a_i^j to minimize $E(x)$ in
$x_{i+1} = x_i + a_i^1 p_i^1 + \cdots + a_i^s p_i^s$
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 $\{ p_{i+1}^1, \ldots, p_{i+1}^s \}$, $\{ p_i^1, \ldots, p_i^s \}$

$$p_{i+1}^1 = r_{i+1} + b_i^{(1,1)} p_i^1 + \cdots + b_i^{(1,s)} p_i^s$$
$$p_{i+1}^2 = Ar_{i+1} + b_i^{(2,1)} p_i^1 + \cdots + b_i^{(2,s)} p_i^s$$
$$\vdots$$
$$p_{i+1}^s = A^{s-1} r_{i+1} + b_i^{(s,1)} p_i^1 + \cdots + b_i^{(s,s)} p_i^s$$

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. Note that for $i = 0$: $w_0 = M_0$; i.e. the matrix of inner products initially coincides with the matrix of moments of r_0.

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 these vectors):

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

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

$$W_{i-1} \boldsymbol{b}^1 + \boldsymbol{c}^1 = 0, \quad \ldots, \quad W_{i-1} \boldsymbol{b}^s + \boldsymbol{c}^s = 0, \tag{3.1a}$$

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

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

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

$$W_i \boldsymbol{a} = m_i \equiv \left[(r_i, p_i^1), \ldots, (r_i, p_i^s) \right]^{\mathrm{T}}. \tag{3.1b}$$

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, and $B = [b^1, \ldots, b^s]$. Then the following equalities hold true.

$$P_i = R_i + P_{i-1} B, \qquad r_i = r_{i-1} - AP_i \boldsymbol{a}.$$

Note that by definition r_i is orthogonal to P_{i-1} and P_i is A-conjugate to P_{i-1}.

Lemma 4.1. *The residual r_i at the ith step is orthogonal to the space R_{i-1}.*

Proof. We have that

$$R_{i-1} = P_{i-1} - P_{i-2}B.$$

Since r_i is orthogonal to the space P_{i-1} we only need to show that r_i is orthogonal to the space P_{i-2}. This holds from $r_i = r_{i-1} - Ap_{i-1}\boldsymbol{a}$, the fact that r_{i-1} is orthogonal to the space P_{i-2}, and the fact that the space P_{i-1} is A-conjugate to the space P_{i-2}. □

Proposition 4.1. *Under the assumption that the matrices W_i and W_{i-1} are nonsingular the linear systems* (3.1a), (3.1b) *have a nontrivial solution if and only if $r_i \neq 0$.*

Proof. It suffices to show that $r_i \neq 0$ implies that $\boldsymbol{b}^1, \ldots, \boldsymbol{b}^s$ and $\boldsymbol{a}$ are nonzero vectors. If $\boldsymbol{c}^k = 0$ for some k then $(A^{k-1}r_i, Ap_{i-1}^1) = \cdots = (A^{k-1}r_i, Ap_{i-1}^s) = 0$. This implies that $A^{k-1}r_i$ is orthogonal to $r_i - r_{i-1}$ and by Lemma 4.1 we conclude that $A^{k-1}r_i$ is orthogonal to r_i. Hence, $r_i = 0$. Now, $m_i = [(r_i, r_i), \ldots, (r_i, A^{s-1}r_i)]^{\mathrm{T}}$ because r_i is orthogonal to the space P_{i-1}. Thus $m_i \neq 0$ as long as $r_i \neq 0$. □

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*

$$V_i = \left\{ r_0, Ar_0, \ldots, A^{(i+1)s-1}r_0 \right\}.$$

(4) *r_i is orthogonal to V_{i-1}.*

Proof. We use induction on i. For $i = 1$ the proof follows from the discussion about the s-dimensional steepest descent method. Let us assume that (1)–(4) hold for $i > 1$. Since $P_i = R_i + P_{i-1}B$ is A-conjugate (by definition) to P_{i-1} it suffices to show that R_i is A-conjugate to $P_0, \ldots, P_{i-2}$. Now write

$$P_j = R_j + \sum_{k=0}^{j-1} l_k[R_{k-1}]$$

where $l_k[R_{k-1}]$ is a linear combination of the vectors $R_0, \ldots, R_{j-1}$. Thus the proof of (1) has been reduced to proving (2).

Now $r_i = r_{i-1} - AP_{i-1}\boldsymbol{a}$ is orthogonal to P_{i-1} (by definition) and to $P_0, \ldots, P_{i-2}$ (by the induction hypothesis). Hence by (3) r_i is orthogonal to V_{i-1}, which proves (4). To prove (2) we must show that R_i is A-conjugate to R_j, $j = 0, \ldots, i-2$, or equivalently that r_i is orthogonal to $\{r_j, Ar_j, \ldots, A^{2s-1}r_j\}$. This holds if $\{r_j, Ar_j, \ldots, A^{2s-1}r_j\} \subset V_{i-1}$. And this holds (by the induction hypothesis on (3)) if the degree$(A^{2s-1}r_j) \leqslant is - 1$, or $(j+2)s - 1 \leqslant is - 1$, or $j \leqslant i - 2$. This proves (2).

The vectors P_j and R_j for $j = 0, \ldots, i$ are A-conjugate by blocks and the belong to the Krylov space V_i. Within each block the vectors are linearly independent. If the contrary is assumed then there exist a polynomial $p(\lambda)$ of degree m such that $q(A)r_0 = 0$ and $m < (i+1)s$. Which is a contradiction. This proves (3). $\square$

Corollary 4.0. If the initial vector x_0 is the same for CG and s-CG then the approximate solution x_i given by s-CG is the same (in exact arithmetic) as the iterate $\tilde{x}_{is}$ given by CG.

Proof. By Theorem 4.1 x_i minimizes $E(x)$ on the Krylov subspace V_i, which is exactly what $\tilde{x}_{is}$ does. $\square$

The following corollary simplifies the computation of the vectors c^j.

Corollary 4.1. *The right-hand side vectors* $c^1, \ldots, c^s$ *for the linear systems* (3.1) *become*

$$\begin{aligned}
c^1 &= \left[0, \ldots, 0, (r_i, A^s r_{i-1})\right]^{\mathrm{T}}, \\
c^2 &= \left[0, \ldots, 0, (r_i, A^s r_{i-1}), (Ar_i, A^s r_{i-1})\right]^{\mathrm{T}}, \\
&\vdots \\
c^s &= \left[(r_i, A^s r_{i-1}), \ldots, (A^{s-1} r_i, A^s r_{i-1})\right]^{\mathrm{T}}.
\end{aligned}$$

Proof. We use the definition of $c^1, \ldots, c^s$ and $p_{i-1}^1, \ldots, p_{i-1}^s$ and Theorem 4.1. $\square$

Using this result and the fact that A is symmetric we find that the vectors can be obtained from the s inner products

$$(A^s r_i, r_{i-1}), \quad (A^{s+1} r_i, r_{i-1}), \quad \ldots, \quad (A^{2s-1} r_i, r_{i-1}).$$

The following proposition reduces the computation of the vectors c^j to the first s moments of r_i.

Proposition 4.2. *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 Theorem 4.1 r_i is orthogonal to $\{A^k p_{i-1}^1, \ldots, A^k p_{i-1}^{s-k-1}\} \subset V_{i-1}$. Thus

$$\begin{aligned}
(A^k r_i, r_i) &= (A^k r_i, r_{i-1}) - \sum_{j=s-k}^{s} a_{i-1}^j (A^k r_i, A p_{i-1}^j) \\
&= (A^k r_i, r_{i-1}) - \sum_{j=s-k}^{s} a_{i-1}^j (A^{k+j} r_i, r_{i-1})
\end{aligned}$$

which proves the proposition. $\square$

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. If we write out p_i^l and p_i^j then since p_i^l is A-conjugate to the space P_{i-1} we get

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

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

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

Proof.

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

We now reformulate the s-CG algorithm taking into account the theory developed above. We will use

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

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

Algorithm 4.2. The s-step Conjugate Gradient Method (s-CG)
Select x_0
Set $P = 0$
Compute $Q = [r_0 = f - Ax_0,\ Ar_0, \ldots, A^{s-1} r_0]$
Compute $\mu^0, \ldots, \mu^{2s-1}$
For $i = 0$ **Until** Convergence **Do**
  Call ScalarWork
  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. $P = [r_{i+1} = f - Ax_{i+1},\ Ar_{i+1}, \ldots, A^{s-1} r_{i+1}]$
   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. $Q = [r_{i+1} = f - Ax_{i+1},\ Ar_{i+1}, \ldots, A^{s-1} r_{i+1}]$
   4. Compute $\mu^0, \ldots, \mu^{2s-1}$
  EndIf
EndFor

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

Vector storage is required for P, Q, x, f, and possibly A; the scalar storage is $O(s^2)$. The scalar work per iteration needed to set up and solve the $(s+1)$ systems of order s is $O(s^3)$ operations. The vector work per iteration is: $(s+1)$ matrix vector products, $2s$ inner products and $(s+1)$ linear combinations of the form $v + \Sigma_{j=1}^{s} c_j u_j$ for $2s(s+1)N$ operations. On the other hand the vector work for s iterations of CG is s matrix vector products, $2s$ inner products and $6sN$ operations for vector updates.

Thus for every s iterations of CG, s-CG performs the additional work of one matrix vector product (Av) and $[2s(s+1)N - 6sN] + O(s^3)$ floating point operations. The extra matrix vector product is introduced because the residual vector is computed directly unlike CG where it is the result of a vector update. If the matrix vector operation dominates the computation of a single iteration of CG then the larger s the less overhead results. However s is restricted because of stability reasons and the fact that the overhead due to linear combinations is $O(s^2 N)$.

The s-CG algorithm has the following advantages over CG for parallel processing:

(i) Steps 1–4 can be performed with one read of the data from the memory and efficient use of fast local storage if the matrix is narrow banded [9].

(ii) The $2s$ inner products for each step can be executed simultaneously. This improves over the CG algorithm. Where the two isolated inner products which are performed in each iteration constitute a bottleneck for parallel computation.

5. s-step conjugate residual method (s-CR)

If we replace (r_i, r_i) and (Ar_i, r_i) by (Ar_i, r_i) and (Ar_i, Ar_i) respectively in Algorithm 2.3 we obtain one form of the Conjugate Residual (CR) method. As in the one-dimensional CR case the error functional $\| f - Ax_{i+1} \|_2$, where $x_{i+1} = x_i + a_i^1 p_i^1 + \cdots + a_i^s p_i^s$, is minimized over the $(i+1)s$-dimensional translated Krylov subspace $x_0 + \{r_0, Ar_0, \ldots, A^{(i+1)s-1} r_0\}$. This gives the s-dimensional Conjugate Residual Method.

Algorithm 5.1. The s-step Conjugate Residual Method (s-CR)
Select x_0
Set $P = 0$
Compute $Q = [r_0 = f - Ax_0,\ Ar_0, \ldots, A^{s-1} r_0]$
Compute $\mu^1, \ldots, \mu^{2s}$

For $i = 0$ **Until** Convergence **Do**
Call ScalarWork
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. $P = [r_{i+1} = f - Ax_{i+1}, Ar_{i+1}, \ldots, A^{s-1}r_{i+1}]$
4. Compute $\mu^1, \ldots, \mu^{2s}$
Else
1. $P = P + Q[\boldsymbol{b}^1, \ldots, \boldsymbol{b}^s]$
2. $x_{i+1} = x_i + P\boldsymbol{a}$
3. $Q = [r_{i+1} = f - Ax_{i+1}, Ar_{i+1}, \ldots, A^{s-1}r_{i+1}]$
4. Compute $\mu^1, \ldots, \mu^{2s}$
EndIf
EndFor
ScalarWork Routine

In s-CR the moments $\mu^1, \ldots, \mu^{2s}$ are required instead of the moments $\mu^0, \ldots, \mu^{2s-1}$ for s-CG. This is the only difference in the computation. This is because we obtained the s-step generalization of the conjugate gradient method by storing the vectors r_i, $Ar_i, \ldots, A^s r_i$, which is done in the conjugate residual method (for $s = 1$). Storing the vectors Ap_i instead and using them in calculating the matrix W_i would amount in computing $O(s^2)$ inner products per iteration. Use of the vectors Ap_i in updating the residual r_i is avoided by computing the residual directly (thus adding an extra matrix vector product per iteration).

For CR we need both Ar_i and Ap_i, computing the latter via an extra vector update: $Ap_i = Ar_i + b_{i-1}Ap_{i-1}$. Thus the work overhead (compared to CR) is $2s(s+1)N - 8sN$ and one matrix vector product per iteration. Thus the overhead in vector operations of s-CR over CR is less severe than s-CG over CG.

6. Loss of orthogonality between the direction subspaces

In finite arithmetic the s-dimensional direction subspaces P_i are nt exactly mutually orthogonal. Hence, when we apply s-CG on the system $Ax = f$, we essentially solve the transformed system:

$$\sum_{l=1}^{n/s} \sum_{j=1}^{s} \left(Ap_l^j, p_i^k\right) a_l^j = \left(A^k r_i, r_i\right)$$

where $1 \leqslant i \leqslant n/s$ and $1 \leqslant k \leqslant s$. Now the diagonal $s \times s$ blocks of the matrix are the matrices W_i. Since $\boldsymbol{a} = W_i^{-1}\boldsymbol{m}_i$ we hope to have a good approximate solution at termination if the diagonal blocks dominate. Let the matrix $\boldsymbol{W}_i$ denote the $s \times s$ block

$$\left(Ap_i^1, p_{i+1}^1\right), \ldots, \left(Ap_i^s, p_{i+1}^1\right), \quad \ldots, \quad \left(Ap_i^1, p_{i+1}^s\right), \ldots, \left(Ap_i^s, p_{i+1}^s\right)$$

then a weaker requirement is $\| W_i^{-1}\overline{W}_i \| \ll 1$ in some operator norm. Since, $(Ap_i^1, p_{i+1}^j) = (Ap_i^1, A^{j-1}r_{i+1}) + \sum_{k=1}^{s}(p_i^k, Ap_i^1)b_k^j$ we can write the matrix $\overline{W}_i$ in the column form

$$\left[\left(W_i\boldsymbol{b}^1 + \boldsymbol{c}^1\right), \ldots, \left(W_i\boldsymbol{b}^s + \boldsymbol{c}^s\right)\right].$$

The condition above becomes

$$\|(\boldsymbol{b}^1 - \hat{\boldsymbol{b}}^1), \ldots, (\boldsymbol{b}^s - \hat{\boldsymbol{b}}^s)\| \ll 1$$

in some vector norm, where $\boldsymbol{b}^j$, $\hat{\boldsymbol{b}}^j$, $j = 1, \ldots, s$ are the true and computed scalars.

From the above we conclude that if the condition number of the matrix W_i is large it will introduce large errors in computing the scalars $\boldsymbol{b}^j$. Computing these scalars involves computing the right-hand side vectors $\boldsymbol{c}^j$ via the recurrence formulae (in Proposition 4.2) and solving of s linear systems each having coefficient matrix W_i. Since this matrix can be near the matrix of moments of r_i, it may have a relatively large condition number. The observed condition number of the matrix W_i for the test problem presented here was 10^{2+s} ($s \leqslant 5$), for double precision arithmetic. Inaccurate computation of the scalars $\boldsymbol{b}^j$ results in orthogonality loss between the direction subspaces P_i and thus slow convergence. However, for small s s-CG the convergence (based on our experiments) is as good as in CG and it verifies Corollary 4.0.

One way to alleviate the orthogonality loss without reducing the parallelism of the s-CG method is to A-orthogonalize the direction subspace P_{i-1}. This can be done simultaneous with the computation of $A^j r_i$, $j = 0, \ldots, s-1$ and prior to computing P_i. Then the computation is based on inverting a diagonal matrix in lieu of W_i. However, this would require additional inner products and linear combinations. Another way would be to periodically apply s consecutive steps of the standard CG method.

7. Numerical results

Experiments were conducted on the ALLIANT FX/8 multiprocessor system at the Center for Supercomputing Research and Development of the University of Illinois. Details will be given in a forthcoming paper [7]; the results are summarized here.

The FX/8 is an example of a supercomputer architecture with memory hierarchy. The configuration of the FX/8 contains 8 pipelined vector processors (CEs) which communicate to each other via a concurrency control bus used as a synchronization device. Each CE has eight vector registers and a computational clock cycle of 170 ns. The maximum performance of one CE is 11 Mflops (million flops/sec) 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/sec. The cache is connected to an 8 MW interleaved global memory via a bus with a bandwidth of about 23.5 MW/sec for sequential read and about 19 MW/sec for sequential write access. Thus accessing data from cache is about twice as fast as assessing it from the global memory. The ALLIANT FX/8 optimizer and compiler restructures a FORTRAN code based on data dependency analysis for scalar, vector, and concurrent execution.

Let us consider the second order elliptic PDE in two dimensions in a rectangular domain Ω in R^2 with homogeneous Dirichlet boundary conditions:

$$-(au_x)_x - (bu_y)_y + (eu)_x + (hu)_y + cu = g \tag{7.1}$$

where $u = H$ on $\partial\Omega$, and $a(x, y)$, $b(x, y)$, $c(x, y)$, $f(x, y)$ and $g(x, y)$ are sufficiently smooth functions defined on Ω, and a, $b > 0$, $c \geqslant 0$ on Ω. If we discretize (7.1) using the five-point

Table 1
Execution times for the CG and 5-CG

$\sqrt{N}$	Steps	Time/sec	Steps	Time/sec
64	136	1.08	27	1.1
100	209	4.66	42	4.24
128	266	10.21	53	8.68
160	331	21.26	66	16.92
200	412	41.39	83	32.91
256	525	92.2	107	70.48
300	613	145.01	123	110.98

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 (7.1) is self-adjoint and A is symmetric and weakly diagonally dominant [22]. 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

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

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

Problem. $-(au_x)_x - (bu_y)_y = g$ on the unit square with homogeneous boundary conditions and $a \equiv b \equiv 1$ and $u(x, y) = e^{xy}\sin(\pi x)\sin(\pi y)$. The matrix of the discretized problem was stored in three diagonals of order N to simulate the general fivepoint difference operator. We solved this problem using 5-step CG, 5-step CR and their one step counterparts on the ALLIANT FX/8. The termination criterion used was $(r_i, r_i)^{1/2} < 10^{-6}$. The results are shown in Tables 1 and 2. The speed-up factors are

$$\mathrm{CG}/5-\mathrm{CG} \approx 1.3, \qquad \mathrm{CR}/5-\mathrm{CR} \approx 1.5.$$

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

$\sqrt{N}$	Steps	Time/sec	Steps	Time/sec
64	133	1.3	28	1.2
100	201	5.66	40	4.17
128	252	12.24	52	8.95
160	307	23.63	62	16.59
200	376	46.6	76	31.54
256	471	97.8	94	64.07
300	544	158.8	110	103.17

Also, the one-step methods took (for convergence) five times the number of iterations taken by the 5-step methods. This was expected from the theory. This also tests the stability of the s-step methods.

8. Conclusions

We have introduced an s-step conjugate gradient method and showed that it converges. The resulting algorithm has better data locality and parallel properties than the standard one-step. The preliminary experiments demonstrate the stability of the new algorithm and its superior performance on parallel computers with memory hierarchy. A disadvantage of the s-step conjugate gradient method is that additional operations (compared to that of the CG method) are required. Also, for large $s > 5$ slow convergence has been observed due to loss of orthogonality among the direction subspaces. This problem is alleviated if preconditioning is used because the matrix W_i is then better conditioned. The design of a stable s-step conjugate gradient method with no additional vector operations compared to the one-step method remains an open question.

Acknowledgement

We thank W. Jalby of INRIA/CSRD University of Illinois at Urbana for suggesting the alternate storage scheme for the direction spaces and pointing out the superior data locality properties of s-CG. We also thank the anonymous referees whose comments helped enhance significantly the quality of presentation of this article.

References

[1] H. Akaike, On a successive transformation of probability distribution and its application to the analysis of the optimum gradient method, *Ann. Inst. Statist. Math. Tokyo* **11** (1959) 1–16.

[2] F.L. Bauer and A.S. Householder, Moments and characteristic roots, *Numer. Math.* **2** (1960) 42–53.

[3] D. Barkai, K.J.M. Moriarty and C. Rebbi, A modified conjugate gradient solver for very large systems, *IEEE Proc. 1986 Internat. Confer. Parallel Processing*, August 1985, pp. 284–290.

[4] M.S. Birman, Nekotorye ocenki dlja metoda naiskoreisego spuska, *Upsehi Matem. Nauk (N.S.)* **5** (1950) 152–155.

[5] P. Concus, G.H. Golub and D.P. O'Leary, A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in: J.R. Bunch and D. Rose, Eds., *Sparse Matrix Computations* (Academic press, New York, 1976).

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

[7] A.T. Chronopoulos and C.W. Gear, Implementation of s-step methods on parallel vector architectures, *IEEE Trans. Comput.*, submitted.

[8] A.T. Chronopoulos and C.W. Gear, Implementation of preconditioned s-step conjugate gradient methods on a multi processor system with memory hierarchy, *Parallel Computing*, submitted.

[9] A.T. Chronopoulos, A class of parallel iterative methods implemented on multiprocessors, Ph.D. thesis, Tech. Rep. UIUCDCS-R-86-1267, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

[10] J.W. Daniel, The conjugate gradient method for linear and nonlinear operator equations, *SIAM J. Numer. Anal.* **4** (1967) 10–26.

[11] H.C. Elman, Iterative methods for large, sparse, nonsymmetric systems of linear equations, Ph.D. thesis, Tech. Rep. 229, Department of Computer Science, Yale University, New Haven, CT, 1982.
[12] G.E. Forsythe, On the asymptotic directions of the s-dimensional optimum gradient method, *Numer. Math.* **11** (1968) 57–76.
[13] G.E. Golub and C.F. Van Loan, *Matrix Computations* (Johns Hopkins University Press, 1983).
[14] M. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. NBS* **49** (1952) 409–436.
[15] I.M. Khabaza, An iterative least-square method suitable for solving large sparse matrices, *Comput. J.* **6** (1963) 202–206.
[16] G. Meurant, The block preconditioned conjugate gradient method on vector computers, *BIT* **24** (1984) 623–633.
[17] D.P. O'Leary, The block conjugate gradient algorithm and related methods, *Linear Algebra Appl.* **29** (1980) 293–322.
[18] E. Stiefel, Uber einige Methoden der Relaxionsrechnung, *Z. Angew. Math. Physik* **3** (1952) 1–33.
[19] Y. Saad, Practical use of polynomial preconditioning for the conjugate gradient method, *SIAM J. Sci. Stat. Comput.* **6** (4) (1985).
[20] Y. Saad, On the Lanczos method for solving symmetric linear systems with several right-hand sides, *Math. Comput.* **48** (1987).
[21] J. van Rosendale, Minimizing inner product data dependence in conjugate gradient iteration, *Proc. IEEE Internat. Confer. Parallel Processing*, 1983.
[22] R. Varga, *Matrix Iterative Analysis* (Prentice Hall, Engledwood Cliffs, NJ, 1962).