A parallel implementation of the restarted GMRES iterative algorithm for nonsymmetric systems of linear equations

Rudnei Dias da Cunha

Centro de Processamento de Dados, Universidade Federal do Rio Grande do Sul, Brazil and Computing Laboratory, University of Kent, Canterbury, UK

and

Tim Hopkins

Computing Laboratory, University of Kent, Canterbury, UK

Received 7 April, 1993; revised 22 November 1993

We describe the parallelisation of the GMRES(c) algorithm and its implementation on distributed-memory architectures, using both networks of transputers and networks of workstations under the PVM message-passing system. The test systems of linear equations considered are those derived from five-point finite-difference discretisations of partial differential equations. A theoretical model of the computation and communication phases is presented which allows us to decide for which values of the parameter c our implementation executes efficiently. The results show that for reasonably large discretisation grids the implementations are effective on a large number of processors.

AMS(MOS) subject classifications: 65F10, 65N22, 65Y05, 65Y20.

1. Introduction

In [19] Saad and Schultz describe the Generalised Minimum Residual (GMRES) iterative algorithm for the solution of nonsymmetric non-singular systems of n linear equations,

$$Ax = b. \tag{1}$$

GMRES was developed to solve (1) when A is not positive real (i.e. the symmetric part of A, $1/2(A + A^T)$, is not positive definite). Two other algorithms, the Generalised Conjugate Residual (GCR) [12] and ORTHODIR [21] often fail to solve such systems; GCR sometimes suffers from breakdown (i.e. a variable becomes undefined due to a division by zero) and ORTHODIR is less numerically stable than GCR.

GMRES never breaks down, unless it has already converged to a solution and requires substantially less storage and arithmetic operations than GCR.

One difficulty that may arise with GMRES in extreme cases is that, because the storage space required by the algorithm increases with the number of iterations, it may be impossible to obtain a solution on a particular hardware system. For some systems, it is possible to use a restarted version, GMRES(c), which considerably reduces the amount of storage required.

In this paper, we consider parallelising the restarted version. We provide analytical and experimental results obtained for the two parallel implementations developed: one using occam2 on transputers and the other using FORTRAN 77 on a network of Sun workstations, under the PVM message-passing system [1].

A brief overview of the paper is as follows. Section 2 describes the GMRES(c) algorithm and its properties. The parallelisation of the method and different approaches proposed by other authors are discussed in section 3 and results and a complexity analysis of the parallel implementations are given in section 4. Finally we give our conclusions in section 5.

2. Description of GMRES(c)

We first consider the non-restarted GMRES algorithm. It is a modification of the Full Orthogonalisation method [18] which uses the Arnoldi process to compute an orthonormal basis $\{v_1, v_2, \ldots, v_k\}$ of the Krylov subspace $\mathcal{K}_k(A, v_1, k)$. The solution $x^{(k)}$ is taken as $x^{(0)} + Vy^{(k)}$, where V is the matrix whose columns are the vectors v_k computed by the Arnoldi process. The vector $y^{(k)}$ is obtained by solving the system $H^{(k)}y^{(k)} = \beta^{(k)}e_1$, where $H^{(k)}$ is a $k \times k$ upper Hessenberg matrix, whose entries are generated during the Arnoldi orthogonalisation, $\beta^{(k)} = \| r^{(0)} \|_2$, $r^{(0)} = b - Ax^{(0)}$, and e_1 is the canonical vector $(1,0,\ldots,0)^{\mathrm{T}}$ of dimension k.

GMRES possesses four important characteristics:

(1) It never breaks down unless convergence has already been achieved [19, proposition 2, p. 865];

(2) It converges in at most n iterations [19, corollary 3, p. 865];

(3) convergence is monotonic, i.e. $\| r^{(k+1)} \|_2 \leq \| r^{(k)} \|_2$. Strict monotonicity is possible if the field of values of A lie in the open right half-plane (see [17, p. 9]); and

(4) The number of vectors of order n that need to be stored grows linearly with k and the number of multiplications grows quadratically.

The last item above is the motivation for the use of a restarted version of GMRES. Instead of generating an orthonormal basis of dimension k, one chooses a value c, $c \ll n$, and generates an approximation to the solution using an orthonormal basis of dimension c. Clearly with this scheme the amount of storage needed may be estimated beforehand and kept within the limits available.

Although GMRES(c) does not break down [19, p. 865], it may, depending on the system and the value of c, produce a stationary sequence of residuals, thus not achieving convergence. A result given by Saad and Schultz [19, corollary 6, p. 867] shows that if A is positive real GMRES(c) converges for any $x^{(0)}$ provided that c is larger than a certain minimum value. This bound on c is not sharp and GMRES(c) may converge even if c is less than the minimum. Determination of this minimum value involves knowledge of the eigenvalue distribution of A and is thus useful in special cases only. It should be noted that the route of convergence of GMRES(c) increases or remains constant when c is increased.

The two operations in GMRES(c) that account for the majority of the execution time are the Arnoldi process and the solution of the least-squares problem

$$\overline{H}^{(c)}y^{(c)} = \beta^{(k)}e_1. \tag{2}$$

The first is rich in vector–vector operations and the implications of this property for the parallelisation of the method are discussed more fully in section 3. The solution of (2) is obtained using the QR factorisation of $\overline{H}^{(c)}$; $y^{(c)}$ being the solution of the triangular system

$$Ry^{(c)} = g, \quad g = Q\beta^{(k)}e_1. \tag{3}$$

For full details, see [19].

A feature of both GMRES and GMRES(c) is that the residual norm of the approximate solution can be obtained without explicitly computing $x^{(k)}$. The norm of the residual is given by

$$\|\beta^{(k)}e_1 - \overline{H}^{(c)}y^{(c)}\| = \|Q(\beta^{(k)}e_1 - \overline{H}^{(c)}y^{(c)})\| = \|g - Ry^{(c)}\|$$

and by (3) this norm is $|g_{k+1}|$ since the last row of R is zero.

2.1. PRECONDITIONING

We consider here the use of GMRES(c) applied to the *left*-preconditioned system,

$$N_m^{-1}Ax = N_m^{-1}b,$$

where N_m^{-1} is the preconditioning matrix obtained by the Neumann incomplete series,

$$N_m^{-1} = \left(\sum_{i=1}^{m} (I - D^{-1}A)^i \right) D^{-1}, \tag{4}$$

where $D = diag(A)$.

This preconditioner was first introduced by Dubois et al. [11] in the solution of symmetric positive-definite systems. In [4] and [9] we provide evidence that this polynomial preconditioner with $m = 1$ is effective for nonsymmetric systems as well. The main advantage of this preconditioner is that it is easily parallelised since it may be expressed as a sequence of matrix–vector products and vector accumulations.

2.2. ALGORITHMIC FORMULATION OF GMRES(c)

We present below an algorithmic formulation of GMRES(c).

```
Step 1.  $r^{(0)} = b - N_m^{-1}Ax^{(0)}$
Step 2.  $\beta^{(0)} = \| r^{(0)} \|_2$
         for $k = 0,1,\ldots$
Step 3.    $V_1 = r^{(k)}/\beta^{(k)}$
           for $j = 1,2,\ldots,c$
Step 4.      $R_{i,j} = V_i^T N_m^{-1} A V_j, \quad i = 1,2,\ldots,j$
Step 5.      $\hat{v} = N_m^{-1}AV_j - \sum_{i=1}^{j} R_{i,j}V_i$
Step 6.      $R_{j+1,j} = \| \hat{v} \|_2$
Step 7.      $V_{j+1} = \hat{v}/R_{j+1,j}$
Step 8.      % Apply previous Givens' rotations to column $j$ of $R$
Step 9.      % Compute Givens' rotation to zero element $R_{j+1,j}$
Step 10.     % Apply Givens' rotation to $g$
           endfor
Step 11.   Solve $Ry^{(c)} = g$
Step 12.   $x^{(k+1)} = x^{(k)} + Vy^{(c)}$
Step 13.   if $|g_{k+1}|$ satisfies tolerance stop.
Step 14.   $r^{(k+1)} = b - N_m^{-1}Ax^{(k+1)}$
Step 15.   $\beta^{(k+1)} = \| r^{(k+1)} \|_2$
         endfor
```

The subscript in V indicates the appropriate columns.

3. Parallelisation of GMRES

The parallelisation of GMRES has been studied by several authors [3, 14–16, 20]. The main approach is to use another formulation to compute the orthonormal basis, in order to avoid the large number of vector–vector operations present in the Arnoldi process. Joubert and Carey [15, 16] generate non-orthogonal bases of Krylov spaces which are then orthogonalised. The least-squares problem (2) is then solved using normal equations; since normal equations tend to have a larger condition number than that of the original problem, loss of accuracy and/or a reduced rate of convergence may occur. The hybrid algorithms proposed require a smaller computational effort than the original formulation of GMRES due to the use of shorter, three-term recurrences instead of the Arnoldi procedure.

Calvetti et al. propose that the orthonormal vectors are generated using a Newton basis. This results in the QR factorisation of a matrix with block structure; the factorisation is carried out in each block independently. Results are reported for an IBM3090-600S VF computer; speed-ups for a system of order n = 40000 and an orthonormal basis of dimension 10 are 1.58 and 2.00 using 2 and 4 processors respectively. Johan et al. used the GMRES(c) algorithm on Finite-Element problems using a Connection Machine CM-2/CM-200 computer. The system (1) is preconditioned using a Cholesky factorisation; symmetric preconditioning is used. The dimension of the Krylov basis for the fluid flow problems solved is small (5 to 15). The least-squares problem (2) is solved locally on the front-end computer. Parallelism is expressed using the data-parallel constructions found in FORTRAN 90.

More recently, Xu et al. have proposed a new version, α-GMRES, which increases the rate of convergence of GMRES. The method is a multilevel iterative scheme composed of an outer and an inner iteration; the system formed at each step of the outer iteration is then solved by GMRES. A block diagonal preconditioner and a damping factor α were used to enhance the convergence of GMRES. The least-squares problem (2) is solved locally in each processor. Results are given for an implementation on a Meiko Computing Surface for 2, 3 and 4 processors solving a system of order $n = 4305$ with speed-ups of 1.9, 2.8 and 3.5 respectively.

3.1. DETAILS OF OUR PARALLEL IMPLEMENTATION

A parallel version of GMRES(c) was obtained using parallel versions of the linear algebra operations found in the algorithm (see section 2.2) – matrix-vector products, inner-products, vector 2-norms, vector scalings and vector accumulations. Such a set of parallel linear algebra subroutines for distributed-memory architectures has been implemented on a transputer-based multiprocessor machine (see [6] and [7] for details). These have been successfully used to implement other iterative methods, for example, PCG, CGS and Bi-CGSTAB. The same algorithms developed for the transputer implementations were used to generate the FORTRAN 77/PVM versions (see [10]) and we found that little effort was needed for the conversion process.

The implementation of GMRES(c) presented here was developed to solve systems derived from the five-point, finite-difference discretisation of partial differential equations (PDEs). Discretising using square grids of $l \times l$ internal points leads to sparse systems of order $n = l^2$.

Our parallel implementation of GMRES(c) uses the Arnoldi process which is a stable implementation of the Gram–Schmidt process. We chose to use this instead of other formulations as those proposed in [3] and [15] because of its stability and ease of implementation.

In the next sections we describe the environments in which the implementations were developed and the parallelisation of the linear algebra operations including the partitioning of data among the processors.

3.1.1. Description of the environments

The occam2 implementation of GMRES(c) was developed under TDS2 on a Meiko Computing Surface using square grids of up to 25 T800-20 transputers each with 4Mbytes of memory. The linear algebra operations were implemented using loop-unrolling techniques, the loops being unrolled to 16 levels [8].

The FORTRAN 77/PVM version used up to 8 Sun SPARC 2 workstations each with 32Mbytes of memory and a cache memory of 64Kbytes under SunOS 4.1.3 using Ethernet connections. This implementation was developed using the Sun FORTRAN 77 compiler and PVM version 2.4.1; all code was compiled in optimised mode. Our applications use the PVM vsnd/vrcv routines which implement the *send/ receive* operations. These are faster than the usual snd/rcv routines and allow a point-to-point exchange of data between the processes through UNIX TCP sockets without using the PVM dæmons.

3.1.2. Vector–vector operations

We consider here the distributed computation of vector updates (scalings and accumulations) and inner-products. For both implementations, the vectors (of length n) described in the GMRES(c) algorithm, are partitioned into segments of contiguous elements; each processor holds a segment of at most $\lceil n/p \rceil$ elements.

Vector scalings and accumulations are disjoint element-wise; the operations performed in each element of the vector operands are independent of the others. Thus, the execution of such operations on a distributed-memory machine incurs no communication overhead between the processors.

For the FORTRAN 77/PVM implementation the inner-product is computed using a recursive-doubling algorithm for both the accumulation of the partial sums (computed locally in each processor) and the broadcast of the inner-product to the processors. These two phases are completed in $2\lceil \log_2 p \rceil$ steps. We stress here that with PVM 2.4.1 the broadcast operation for the vsnd operation is not implemented as a recursive-doubling algorithm but as $p-1$ *send* operations which is an unacceptable overhead.

The inner-product implemented in occam2 uses an algorithm similar to recursive-doubling, except that the number of steps required for completion of the accumulation and broadcast phases is $4\lfloor \sqrt{p}/2 \rfloor$. We note that for $1 < p < 64$, the number of steps required by this algorithm is less than or equal to that needed using the recursive-doubling algorithm.

3.1.3. Matrix–vector operation

For the occam2 version we want to exploit the four communication links available on the transputer. This leads us to use grids of $p = p_r \times p_c$ processors (for simplicity we will consider only square grids of $\sqrt{p} \times \sqrt{p}$ processors). Such a topology maps ideally onto the physical discretisation of the PDE. The vectors containing the coefficients that implicitly form A are partitioned using ***geometric distribution*** by ***blocks***. Each processor holds at most $\lceil l/\sqrt{p} \rceil \times \lceil l/\sqrt{p} \rceil$ elements of each of the coefficient vectors. This distribution allows the use of ***nearest-neighbour*** communication between the processors when computing matrix–vector products; i.e. each processor exchanges information with at most four neighbouring processors depending on the position of the processor in the grid. The number of grid points exchanged with each neighbour is $\lceil l/\sqrt{p} \rceil$.

In the FORTRAN 77/PVM version the hardware used has just one connection with the Ethernet network. In this case we use ***geometric distribution*** by ***panels*** and the processors (workstations) are logically connected as a linear processor array. Each processor then holds at most $l \times \lceil l/p \rceil$ elements of each of the coefficient vectors. Again, nearest-neighbour communications are used but each processor has at most two neighbours and, due to the partitioning used, each processor exchanges l grid points with its neighbour(s).

Figure 1 shows the two data partitioning schemes used. The black squares represent points of the PDE discretisation and the white surrounding squares and rectangles represent the processors. The interaction between the processors is indicated by the arrows.

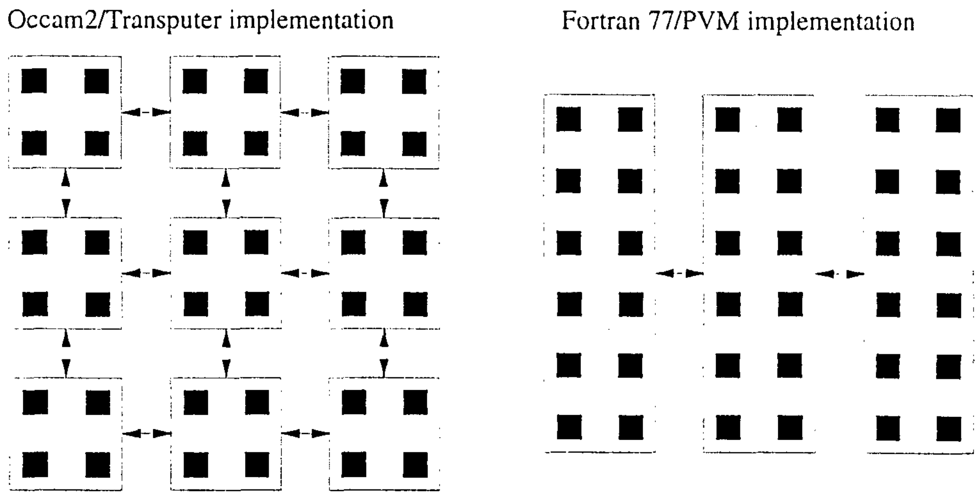

Occam2/Transputer implementation
Fortran 77/PVM implementation

Figure 1. Data partitioning and exchange for matrix–vector product.

3.1.4. *Other considerations*

For the solution of the least-squares problem (2) we need a triangular system solver. Triangular solvers are known to offer less efficiency than other linear algebra operations on distributed-memory machines. Our experiences have shown that efficiencies higher than 50% can only be obtained on such machines for large systems (more than 1000 equations) – see [2] and [5]. This rules out the use of a parallel triangular solver in our implementation, since c is usually small. Our approach to this problem was to let each processor have its own copy of R, $y^{(c)}$ and g and to solve the triangular system $Ry^{(c)} = g$ locally. This approach is similar to that used by Xu et al. [20].

The update of x in step 12 can be regarded as a sequence of vector scalings and sums by rewriting $x^{(k+1)} = x^{(k)} + Vy^{(c)}$ as $x^{(k+1)} = x^{(k)} + y_j^{(c)}V_j$, $j = 1, \dots, c$. These operations can then be performed independently in each processor.

The Arnoldi orthonormalisation process (steps 3–10) makes substantial use of vector–vector operations with low granularity, requiring $(c^2 + c)/2$ inner-products and c vector 2-norms. Due to the low granularity of these operations, the process has a large overall communication overhead which may only be overcome for large n. The communication load during the Arnoldi process is however minimised since a number of the operations in steps 5, 7, 11 and 12 are executed locally.

4. Results

The test problem used is the system derived from discretising

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + \frac{1}{2}\left(\frac{\partial au}{\partial x} + a\frac{\partial u}{\partial x}\right) = 1 \qquad (5)$$

on the unit square $R = (0, 0) \times (1,1)$, where $a = 20e^{3.5(x^2+y^2)}$. Dirichlet conditions $u = 0$ are imposed on ΩR and the first order terms are discretised using central differences.

Equation (5) was discretised on grids with $l = 64$ and 128, the order of the corresponding systems being $n = 4096$ and 16384. The Neumann polynomial preconditioner (4) of degree $m = 1$ was used in our experiments.

4.1. SCALABILITY OF THE IMPLEMENTATION

In tables 1 and 2 we list the number of iterations required for convergence and the execution time (in seconds) for the occam2, and FORTRAN 77/PVM implementations. The execution time of the occam2 implementation for n = 16384 and $p = 1$ is an estimated time, since the data does not fit in the memory size available; for the same reason we do not list the results for p = 4. Note that while increasing c reduces the number of iterations the execution time may still increase.

In terms of execution times, the Fortran 77/PVM implementation is considerably faster than the occam2 implementation due to the more powerful SPARC 2 processor, even at the expense of a higher communication latency in the Ethernet network (see section 4.2).

Table 1

Number of iterations and execution times (in seconds) of the occam2 implementation.

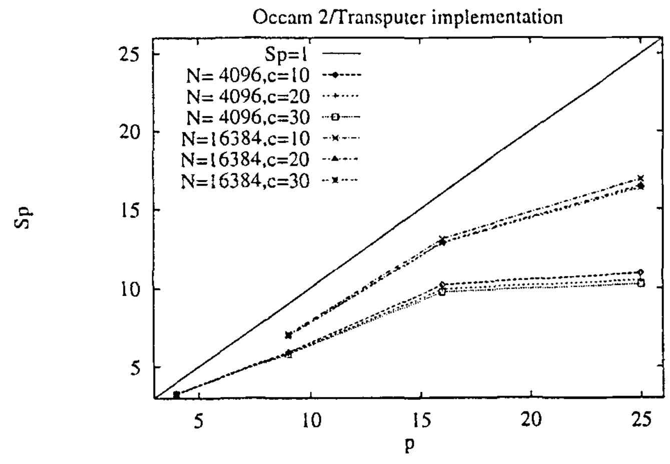
		$n = 4096$				
c	k^*	$p = 1$	$p = 4$	$p = 9$	$p = 16$	$p = 25$
10	611	2902.51	878.43	487.65	284.25	264.88
20	171	2302.04	700.88	391.89	231.58	219.01
30	96	2517.07	771.87	433.14	257.73	245.55
		$n = 16384$				
c	k^*	$p = 1$	$p = 4$	$p = 9$	$p = 16$	$p = 25$
10	4032	77612.87	–	10924.62	5916.36	4590.86
20	1258	68252.86	–	9756.82	5286.04	4128.55
30	469	49846.94	–	7150.41	3875.06	3047.14

Table 2

Number of iterations and execution times (in seconds) of the Fortran 77/PVM implementation.

		$n = 4096$			
c	k^*	$p = 1$	$p = 2$	$p = 4$	$p = 8$
10	611	1477.09	640.02	397.15	253.56
20	171	1119.20	477.52	271.89	202.21
30	96	1192.80	522.72	301.20	210.00
		$n = 4096$			
c	k^*	$p = 1$	$p = 2$	$p = 4$	$p = 8$
10	4032	45491.04	23052.96	12075.84	6017.76
20	1258	39148.96	19980.19	10145.77	5009.99
30	469	28192.76	14397.13	7348.06	3640.61

In figure 2 we show the speed-ups ($S_p = T_1/T_p$) of the parallel implementation of GMRES(c) for the solution of (5) with $c = 10$, 20 and 30, where T_1 is the execution time of a sequential implementation of GMRES(c) and T_p is the execution time of the parallel implementation on p processors.



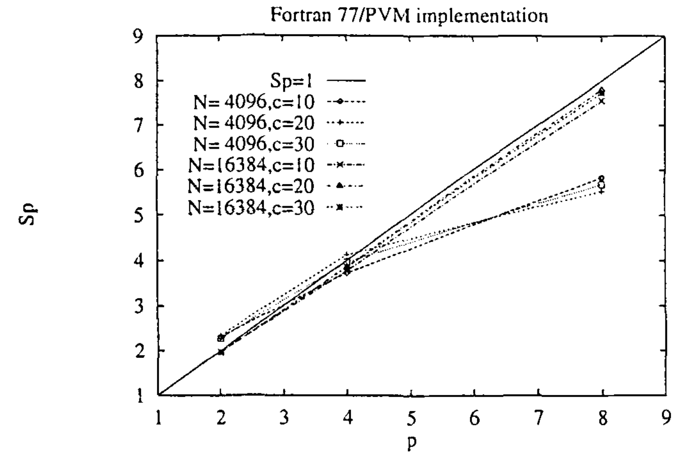


Figure 2. Speed-ups of GMRES(c).

We note that as n increases the amount of local computation and, hence, the speed-up increases. On the other hand increasing c increases the amount of data communicated between the processors (more matrix–vector products, inner products and vector 2-norms are required per iteration) and thus decreases the speed-up.

The graphs for the FORTRAN 77/PVM implementation show a superlinear effect which we attribute to the presence of the cache memory.

4.2. COMPLEXITY ANALYSIS OF THE PARALLEL IMPLEMENTATION

For fixed l and p, we would like to know which values of c will ensure an efficient solution. To help answer this we have developed an analytical model that describes the time taken to perform the floating-point operations and the communication within GMRES(c). For the occam2 model we consider only square grids of processors. If we consider the analytical speed-up

$$S_p^a = \frac{T_{flop}^{seq}}{T_{flop}^{par} + T_{comm}}, \tag{6}$$

then we should be able to gain some insight into which values of c an implementation is scalable with respect to the number of processors.

The model for the floating-point operations is taken as

$$T_{flop} = O_* T_* + O_+ T_+ + (kc + 2k + 1) T_{\sqrt{}}, \tag{7}$$

where T_*, T_+ and $T_{\sqrt{}}$ are the times taken to perform a double-precision multiplication, sum and square-root respectively, taking into account the overhead of addressing the operands.

Due to the partitioning of data, the number of arithmetic operations differs between the processors, especially for matrix–vector products, and depends on their relative positions within the processor grid. The expressions for O_* and O_+ for the parallel implementations are obtained for the processors with the highest number of operations. In the occam2 version these are processors which are not adjacent to an edge of the grid; for the FORTRAN 77/PVM implementation these processors are those not at the ends of the linear array.

Sequential

$$\begin{aligned} O_* &= (kc^2/2 + 17kc/2 + 6m(kc + k + 1) + 9k + 8)l^2 \\ &\quad - 4(m + 1)(kc + k + 1)l + 3kc^2 + 3kc + k, \\ O_+ &= (kc^2/2 + 13kc/2 + 6m(kc + k + 1) + 8k + 6)l^2 \\ &\quad - 4(m + 1)(kc + k + 1)l - 2kc - 2k - 1; \end{aligned}$$

occam2

$$\begin{aligned} O_* &= (kc^2/2 + 7kc/2 + m(kc + k + 1) + 4k + 3)\lceil l^2/p \rceil \\ &\quad + 5(m + 1)(kc + k + 1)\lceil l/\sqrt{p} \rceil^2 + 3kc^2 + 3kc + k, \\ O_+ &= (kc^2/2 + 5kc/2 + 2m(kc + k + 1) + 4k + 2)\lceil l^2/p \rceil \\ &\quad + 4(m + 1)(kc + k + 1)\lceil l/\sqrt{p} \rceil^2 - 3kc - 4k + 2\lfloor \sqrt{p}/2 \rfloor - 2, \end{aligned}$$

$$T_* = 4.364 \times 10^{-6}\text{s}, \quad T_+ = 4.114 \times 10^{-6}\text{s}, \quad T_{\sqrt{}} = 1.269 \times 10^{-5}\text{s};$$

FORTRAN 77/PVM

$$O_* = (kc^2/2 + 7kc/2 + m(kc + k + 1) + 4k + 3)\lceil l^2/p \rceil$$
$$+ (kc + k + 1)(5lm + 5l - 2m - 2)\lceil l/p \rceil + 3kc^2 + 3kc + k,$$
$$O_+ = (kc^2/2 + 5kc/2 + 2m(kc + k + 1) + 4k + 2)\lceil l^2/p \rceil$$
$$+ (kc + k + 1)(4lm + 4l - 2m - 2)\lceil l/p \rceil - 3kc - 4k + 2\lceil \log_2 p \rceil - 2,$$

$$T_* = 1.7 \times 10^{-6}\text{s}, \quad T_+ = 1.65 \times 10^{-6}\text{s}, \quad T_{\sqrt{\ }} = 8.1 \times 10^{-6}\text{s}.$$

For the communication model, we assume that the time taken to transmit x 64-bit words may be given in the form $T_w(x) = \alpha + \beta x$, where α is the start-up time of the communication and β is the time to transfer each word. These parameters are dependent on the communication software and hardware used and were determined by sending messages of different lengths between two processors. The total communication time for each parallel implementation is given for occam2 by

$$T_{comm} = (kc^2/2 + 3kc/2 + 2k + 1)4\lfloor \sqrt{p}/2 \rfloor T_w(1)$$
$$+ (m + 1)(kc + k + 1)4T_w\left(\lceil l/\sqrt{p} \rceil\right) \quad (8)$$

with $\alpha = 2.81 \times 10^{-4}$s, $\beta = 8.66 \times 10^{-6}$s, where the terms $4\lfloor \sqrt{p}/2 \rfloor T_w(1)$ and $4T_w\left(\lceil l/\sqrt{p} \rceil\right)$ represent the time taken to transfer the data during the calculation of the inner-products and matrix–vector products, respectively, and for the FORTRAN 77/PVM version by

$$T_{comm} = (kc^2/2 + 3kc/2 + 2k + 1)2(\lceil \log_2 p \rceil)T_w(1)$$
$$+ (m + 1)(kc + k + 1)2T_w(l), \quad (9)$$

with $\alpha = 2.70 \times 10^{-4}$s, $\beta = 1.21 \times 10^{-5}$s, where the terms $2\lceil \log_2 p \rceil T_w(1)$ and $2T_w(l)$ represent the time taken to transfer the data during the calculation of the inner-products and matrix–vector products respectively.

We note that equation (8) is an *upper-bound* for the communication time since it does not take into account the fact that use of the communication links in the grid of transputers is not synchronous – for example, it is possible that while one link is transferring data for a matrix–vector product another is transferring data for an inner-product.

To obtain the values of c, we may consider that an acceptable speed-up is obtained if $S_p^a \geq p/2$. If we consider $S_p^a = p/2$ then we may compute the roots of the quadratic equation in c

$$2T_{flop}^{seq} - p\left(T_{flop}^{par} + T_{comm}\right) = 0. \tag{10}$$

The roots of (10) then give the desired information about scalability:

- If the roots are complex, then there is no value of c for which $S_p^a \geq p/2$ is satisfied.
- If the roots are both negative, then for any value of c we achieve $S_p^a \geq p/2$.
- If one root is negative and the other is positive: select the positive root c^+ since $c > 0$ and take any integer number $c \leq \lfloor c^+ \rfloor$.
- Equation (10) cannot have two positive roots. Writing equation (10) in the form $Ec^2 + Fc + G = 0$ where E, F and G are functions of p, note that there can be two positive roots iff

$$(E < 0 \wedge F > 0 \quad \wedge G < 0) \vee (E > 0 \wedge F < 0 \quad \wedge G > 0) \tag{11}$$

If either $EF > 0$ or $EG < 0$ then the condition above is not satisfied. It can be shown (with the aid of MAPLE) that EF and EG have two real roots ($p'_{EF}, p''_{EF}, p'_{EG}$ and p''_{EG}, respectively) where $p'_{EF} = p'_{EG}$ and that, for p large, EF is positive. Figure 3 shows the format of the curves EF and EG. Since the interval $[p'_{EF}, p''_{EF}]$ is contained in $[p'_{EG}, p''_{EG},]$ it follows that when $EF < 0$, EG is also negative. Thus condition (11) is not satisfied and equation (10) cannot have two positive roots.

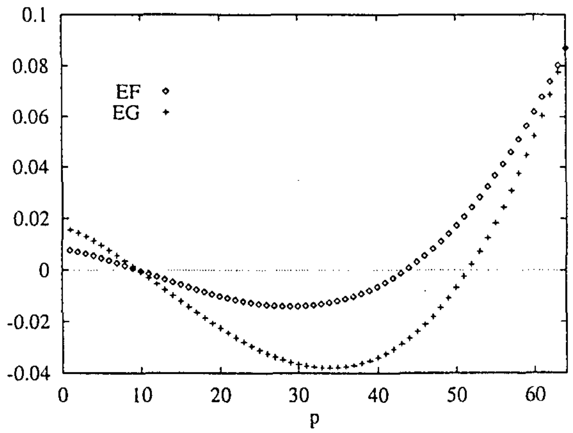


Figure 3. Format of curves *EF* and *EG*.

In table 3 we list the values of c obtained from the roots of (10) for some values of p and l with $k = 1$ and $m = 1$; the numerical values indicate the maximum c that can be chosen to obtain at least 50% of efficiency. Note that for the occam2 implementation the values of c are larger than those of the FORTRAN 77/PVM version.

Table 3

Values of c for which $S_p^a = p/2$: A = "any", N = "none".

		occam2 implementation								
		p								
l	n	4	9	16	25	36	49	64	81	100
32	1024	292	11	N	N	N	N	N	N	N
64	4096	A	A	18	8	2	N	N	N	N
128	16384	A	A	A	A	29	15	6	4	2
256	65536	A	A	A	A	A	A	348	68	21
512	262144	A	A	A	A	A	A	A	A	A
1024	1048576	A	A	A	A	A	A	A	A	A

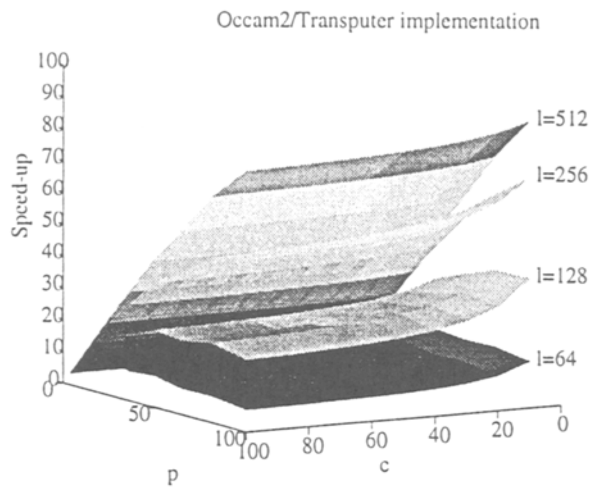
		Fortran 77/PVM inplementation								
		p								
l	n	4	9	16	25	36	49	64	81	100
32	1024	52	N	N	N	N	N	N	N	N
64	4096	A	160	7	N	N	N	N	N	N
128	16384	A	A	A	60	12	2	N	N	N
256	65536	A	A	A	A	A	A	A	58	9
512	262144	A	A	A	A	A	A	A	A	A
1024	1048576	A	A	A	A	A	A	A	A	A

In addition the occam2 implementation is more effective than the FORTRAN 77/PVM version for small l. For large l, both implementations are capable of providing good scalability for any value of c with the number of processors p, considered.

A question that arises is that of the effect of the choice of m in the scalability of our implementation. If we compute the roots of (10) for $m > 1$ we find that $S_p^a \geq p/2$ is possible for larger values of c. When $m = 0$, we also note that for larger values of p it is not possible to achieve the required speed-up for any values of c. Conversely, for $m > 1$, the desired speed-up is obtained for some values of p which are not possible for $m = 1$.

The choice of m to achieve convergence in the shortest possible time depends on the properties of the system. Our experiences show that, in the nonsymmetric case the Neumann polynomial preconditioner can be used for $m = 1$ but higher degrees usually cause an increase in the number of iterations required for convergence. This effect is due to divergence of the Neumann series to A^{-1}. However, we would like to point out that least-squares polynomial preconditioners have been used with higher degrees (typically $m = 8$) by Holter et al. [13].

Figure 4 shows the graphs of equation (6) for different values of l, c and p ($k = 1$ and $m = 1$) using the corresponding equations for the models of the



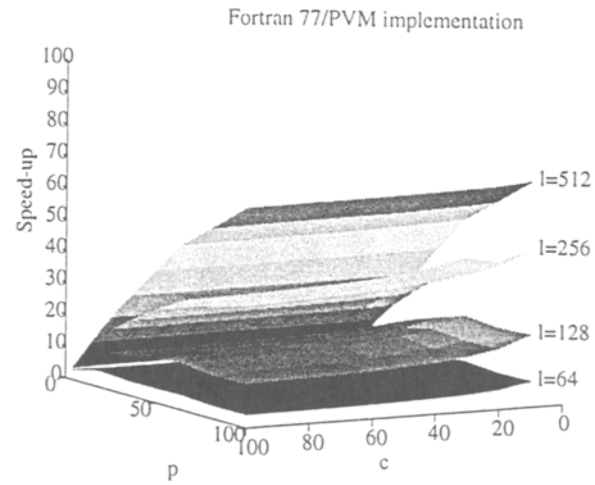


Figure 4. Analytical speed-up of GMRES(c).

implementations. It can be seen that as the problem size increases it is possible to solve the system efficiently on an increasing number of processors with a larger value of c (note that c is always chosen to be considerably smaller than n). This in turn may allow for a reduction in the overall execution time due to the smaller number of iterations required for convergence when using higher dimensions in the orthonormal basis.

5. Conclusions

A parallel version of the GMRES(c) algorithm has been implemented using parallel implementations of a set of linear algebra operations on distributed-memory computers. The parallel algorithm was tested using a transputer based machine and a network of workstations using the PVM message-passing system. The implementations have been shown to be scalable with respect to the number of processors, especially for large problems.

A theoretical model for the computation and communication phases has been developed and, based on this model, we show that the communication overheads imposed by the Arnoldi orthonormalisation process may be overcome for large n. The occam2 implementation is more efficient for smaller problem sizes, mainly due to the small latency of communication. For larger problems, both the occam2 and the FORTRAN 77/PVM versions are capable of providing speed-ups of more than $p/2$ regardless of the value chosen for c. The main implication of this is that, for the parallel algorithm presented here, it is possible to effectively use networks of workstations even with high latency communications hardware.

An effect that has been detected only in the FORTRAN 77/PVM implementation is that of superlinearity. We attribute this to the existence of a cache memory on the SPARC2 processor. Thus it may be possible to use this implementation with a larger value of c than that predicted by the model and still obtain a good speed-up.

As a final note we have also implemented parallel versions of GMRES(c) for generic systems and for problems derived from the seven-point finite-difference discretisation of PDEs in three dimensional regions and have obtained a behaviour similar to that discussed here.

Acknowledgement

We would like to thank the referees for their suggestions which helped to improve this paper.

References

[1] A. Beguelin, J. Dongarra, A. Geist, R. Manchek and V.S. Sunderam, A user's guide to PVM – Parallel Virtual Machine, Research Report ORNL/TM-11826, Oak Ridge National Laboratory (1992).

[2] R.H. Bisseling and J.G.G. Van der Vorst, Parallel triangular solving on a mesh network of transputers, SIAM J. Sci. Statist. Comp. 12(1991)787–799.

[3] D. Calvetti, J. Petersen and L. Reichel, A parallel implementation of the GMRES method, in: *Numerical Linear Algebra*, ed. L. Reichel, A. Ruttan and R.S. Varga (de Cruyter, Berlin, 1993) pp. 31–46.

[4] R.D. da Cunha, A study on iterative methods for the solution of systems of linear equations on transputer networks, PhD Thesis, Computing Laboratory, University of Kent at Canterbury (July, 1992).

[5] R.D. da Cunha and T.R. Hopkins, The parallel solution of triangular systems of linear equations, in: *High-Performance Computing II – Proc. 2nd Symp. on High Performance Computing*, ed. M.Durand and F. El Dabagni (North-Holland, Amsterdam, 1991) pp. 245–256. Also as Report No. 86, Computing Laboratory, University of Kent at Canterbury, UK.

[6] R.D. da Cunha and T.R. Hopkins, The parallel solution of partial differential equations on transputer networks, in: *Transputing for Numerical and Neural Network Applications* (IOS Press, Amsterdam, 1992) pp. 96–109. Also as Report No. 17/92, Computing Laboratory, University of Kent at Canterbury, UK.

[7] R.D. da Cunha and T.R. Hopkins, The parallel solution of systems of linear equations using iterative methods on transputer networks, in: *Transputing for Numerical and Neural Network Applications* (IOS Press, Amsterdam, 1992) pp. 1–13. Also as Report No. 16/92, Computing Laboratory, University of Kent at Canterbury, UK.

[8] R.D. da Cunha and T.R. Hopkins, Increasing the performance of occam using loop-unrolling, Report No. 4/93, Computing Laboratory, University of Kent at Canterbury (April 1993); to appear in Transputer Commun.

[9] R.D. da Cunha and T.R. Hopkins, Parallel preconditioned Conjugate-Gradient methods on transputer networks, Report No. 5/93, Computing Laboratory, University of Kent at Canterbury (April 1993); to appear in Transputer Commun. 1(1993).

[10] R.D. da Cunha and T.R. Hopkins, PIM 1.0 – the Parallel Iterative Methods package for systems of linear equations User's Guide – FORTRAN77 version, Internal Report, Computing Laboratory, University of Kent at Canterbury (November 1993).

[11] 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.

[12] S.C. Eisenstat, H.C. Elman and M.H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, SIAM J. Numer. Anal. 20(1983)345–357.

[13] W.H. Holter, I.M. Navon and T.C. Oppe, Parallelizable preconditioned Conjugated Gradient methods for the Cray Y-MP and the TMC CM-2, Technical report, Supercomputer Computations Research Institute, Florida State University (December 1991).

[14] Z. Johan, T.J.R. Hughes, K.K. Mathur and S.L. Johnsson, A data parallel finite-element method for computational fluid-dynamics on the Connection Machine system, Comp. Meth. Appl. Mech. Eng. 99(1992)113–134.

[15] W.D. Joubert and G.F. Carey, Parallelizable restarted iterative methods for nonsymmetric linear systems I – theory, Int. J. Comp. Math. 44(1992)243–267.

[16] W.D. Joubert and G.F. Carey, Parallelizable restarted iterative methods for nonsymmetric linear systems 2 – Parallel implementation, Int. J. Comp. Math. 44(1992)269–290.

[17] N.M. Nachtigal, S.C. Reddy and L.N. Trefethen, How fast are nonsymmetric iterations?, Numerical Analysis Report, 90-2. Department of Mathematics, Massachusetts Institute of Technology (March 1990).

[18] Y. Saad, Krylov subspace methods for solving large unsymmetric systems, Math. Comp. 37(1981) 105–126.

[19] Y. Saad and M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statist. Comp. 7(1986)856–869.

[20] X. Xu, N. Qin and B.E. Richards, Alpha-GMRES – a new parallelizable iterative solver for large sparse nonsymmetrical linear-systems arising from CFD, Int. J. Numer. Meth. Fluids 15(1992)613–623.

[21] D.M. Young and K.C. Jea, Generalized Conjugate-Gradient acceleration of nonsymmetrizable iterative methods, Lin. Alg. Appl. 34(1980)159–194.