

A PARALLEL VARIANT OF GMRES(m)

ERIC DE STURLER

Delft University of Technology  
Faculty of Technical Mathematics and Informatics  
P.O. box 356, NL-2600 AJ Delft, The Netherlands

**Abstract:** In the usual implementation of GMRES(m) [3] the computationally most expensive part is the Modified Gram-Schmidt process (MGS). It is obvious, that the MGS process is not well parallelizable on distributed memory multiprocessors, since the inner products act as synchronization points and thus require communication that cannot be overlapped. Furthermore, as all orthogonalizations must be done sequentially, MGS generates a large number of short messages, which is relatively expensive. Especially on large processor grids the time spent in communication in the MGS process may be significant. For this reason a variant of the usual GMRES(m) algorithm is considered, called modGMRES(m), which first generates the vectors that span the Krylov space and then combines the MGS steps for a group of vectors. It is shown, on real world problems, that the modGMRES(m) method can yield a considerable gain in time per iteration. Numerical experience suggests that the total number of iterations remains about the same as for GMRES(m).

Introduction

As described in [2] at SHELL's KSEPL the reservoir simulator Bosim has been parallelized on a Meiko Computing Surface, a transputer based parallel computer. The parallelization is based on a 2-D domain decomposition, where the reservoir is divided among a 2-D grid of processors, in addition there is a master processor, which handles the initialization and input/output. In Parallel Bosim the largest part of the computation is done in Fortran. On each processor, the Fortran program runs in parallel with an Occam process. The Occam processes on different processors form a harness that takes care of the communication. When Fortran on a processor has to communicate, it sends the data to the local Occam process and then continues until its next communication. Meanwhile, the Occam processes concurrently take care of the communication and see to it that the data is returned to Fortran on the receiving processor(s). In this way communication and computation can be overlapped. Within this Parallel Bosim package the GMRES(m) and modGMRES(m) method were implemented and compared. In Bosim the convergence in the linear solver is checked by a separate routine, which needs the residual vector. Therefore the convergence in the linear solver is only checked after each complete (mod)GMRES(m) cycle.

The modGMRES(m) method

Let  $Ax = b$  be the preconditioned system, and let  $r = b - Ax$  be the residual. Let  $v_1$  be the normalized residual, then from  $v_1$  a suitable set of vectors  $\hat{v}_2, \dots, \hat{v}_{m+1}$ , which span the Krylov space, is generated (see below). Then in the MGS process these vectors are orthogonalized. Because the Krylov space is generated in a different way, the Hessenberg matrix  $H_{m+1} = V^T AV$  must be computed from the coefficients of the MGS process, where  $V$  is the matrix  $[v_1, v_2, \dots, v_{m+1}]$ . The rest of the method is analogous to the GMRES(m) method. See however also [1].

In the MGS process the vectors  $v_1, \hat{v}_2, \dots, \hat{v}_{m+1}$  are orthogonalized in the following steps:

1. orthogonalize  $\hat{v}_2, \dots, \hat{v}_{m+1}$  on  $v_1$ , normalize  $\hat{v}_2$ , which gives  $v_2$
2. orthogonalize  $\hat{v}_3, \dots, \hat{v}_{m+1}$  on  $v_2$ , normalize  $\hat{v}_3$ , which gives  $v_3$

Each step can be done blockwise, because the orthogonaliza-

tions in one step are all independent. As the innerproducts computed on each processor are strictly local, these local coefficients must be accumulated over the processor grid to compute the global coefficients. This is done blockwise by the routines `accs(array,len)`, which sends the array to Occam and returns so that Fortran can continue, while Occam performs the actual accumulation in parallel, and `accr(array,len)`, which receives the accumulated values from Occam. The MGS process is implemented as follows (locally on each processor):

```
do i = 1, m - 1
  nbl = max(1, (m - i)/2)
  do k = i, i + nbl - 1
    h(k, i) = v_i^T v_{k+1}
    accs(h(i, i), nbl)
  do k = i + nbl, m - 1
    h(k, i) = v_i^T v_{k+1}
    accr(h(i, i), nbl)
  v_{i+1} = v_{i+1} - h(i, i) * v_i
  h(m, i) = v_{i+1}^T v_{i+1}
  accs(h(i + nbl, i), m - i - nbl + 1)
  do k = i + 1, i + nbl - 1
    v_{k+1} = v_{k+1} - h(k, i) * v_i
    accr(h(i + nbl, i), m - i - nbl + 1)
  do k = i + nbl, m - 1
    v_{k+1} = v_{k+1} - h(k, i) * v_i
  h(i, i + 1) = sqrt(h(m, i))
  v_{i+1} = h(i, i + 1)^{-1} * v_{i+1}
```

In this implementation communication is mostly overlapped and also time is saved by combining small messages, corresponding to one group of orthogonalizations, in one large message, which saves startup times and Fortran-Occam communication.

The generation of the vectors, that span the Krylov space, can be handled in two ways. If the condition number of the preconditioned system is sufficiently small and  $m$  is also relatively small (e.g. 10 or 20), the vectors can be generated as  $v_1, Av_1, A^2v_1, \dots, A^m v_1$ , where  $A$  is the preconditioned matrix; this will be referred to as version 1. However for larger  $m$  and/or a preconditioned matrix  $A$  with a large condition number, the matrix  $[v_1, Av_1, \dots, A^m v_1]$  will be so poorly conditioned that orthogonalization with the MGS algorithm will not produce a set of sufficiently orthogonal  $v_i$  and consequently will result in an inaccurate representation of  $H_{m+1}$ . Therefore the  $v_i$  might be generated as follows:

```
do i = 1, m
  \hat{v}_{i+1} = \hat{v}_i - d_i A \hat{v}_i, (\hat{v}_1 = v_1)
```

where the  $d_i$  are parameters, which should be chosen in such a way that the condition number of the matrix  $[v_1, \hat{v}_2, \dots, \hat{v}_{m+1}]$  is sufficiently small. This will be referred to as version 2. The computation of the  $d_i$  can be based, for example, on the eigenvalues of  $H_{m+1}$ . The exact computation of these parameters is however outside the scope of this paper.

Computational and Communication Costs of GMRES(m) and modGMRES(m)

The total computational costs will be expressed in terms of the timings of the main computational kernels. The computational costs for GMRES(m) on each processor are given by:

$$C_{gmres}^{cp} = \frac{1}{2}(m^2 + 3m)T_{dot} + \frac{1}{2}(m^2 + 3m)T_{dotpy} + (m + 1)T_{scal} + (m + 1)T_{mat} + (m + 1)T_{prec} + T_{incon} \tag{1}$$

$$C_{m,1}^{cp} = \frac{1}{2}(m^2 + 3m)T_{ddot} + \frac{1}{2}(m^2 + 3m)T_{daxpy} + (m+1)T_{dscal} + (m+1)T_{mat} + (m+1)T_{prec} + T_{lincon} + T_{hess}(m) \quad (2)$$

and for version 2 by:

$$C_{m,2}^{cp} = \frac{1}{2}(m^2 + 3m)T_{ddot} + \frac{1}{2}(m^2 + 5m)T_{daxpy} + (m+1)T_{dscal} + (m+1)T_{mat} + (m+1)T_{prec} + T_{lincon} + T_{hess}(m) \quad (3)$$

where *lincon* is the routine that checks whether convergence is reached and *hess* is the routine that computes  $H_{m+i}$  in *modGMRES(m)*. The communication costs for *GMRES(m)* are given by:

$$C_{m,1}^{cm} = \frac{1}{2}(m^2 + 3m)T_{accsum} \quad (4)$$

where *accsum* is a Fortran routine that sends one number to Occam, waits for Occam to accumulate the values over the processor grid and receives back the global result. The communication costs for *modGMRES(m)* are given by:

$$C_{m,1}^{cm} = 2mT_{accs} + 2mT_{accr} + T_{ovh}(m) + T_{noc} \quad (5)$$

$$T_{ovh}(m) = 2mT_c + \frac{1}{2}m^2T_{cpn} \quad (6)$$

where  $T_{ovh}$  indicates the time overhead measured in the *daxpy*'s and *ddot*'s which overlap the accumulation,  $T_c$  is some constant time and  $T_{cpn}$  gives a time increase per number.  $T_{noc}$  indicates the cost of non-overlapped communication/accumulation.

### Results

The *GMRES(m)* and the *modGMRES(m)* method were compared simulating a real reservoir for a large number of iterations. As the simulation had to be done on a fairly small machine (1 master and 24 gridnodes), whereas the *modGMRES(m)* method will only be really advantageous on a relatively large processor grid, involving 100 or more processors, a small reservoir was simulated on a 1D (line) processor grid. This gives 24 communication steps for accumulating a distributed value over the grid, which compares to a processor grid of some 150 processors. The number of unknowns in this problem was 2138, which gave a maximum of 105 unknowns on a processor. With an average of some 90 unknowns per processor and 150 processors this compares to a reservoir with about 13500 (active) gridblocks, which is a medium scale reservoir model. This produced the following average timings on the busiest processor:

computational costs	time (ms)	communication costs	time (ms)
$T_{daxpy}$	0.30594	$T_{accsum}$	0.62041
$T_{ddot}$	0.24707	$T_{accs}$	0.16000
$T_{dscal}$	0.12354	$T_{accr}$	0.06400
$T_{mat}$	2.8251	$T_c$	0.35625
$T_{prec}$	2.4320	$T_{cpn}$	0.16293
$T_{lincon}$	3.0336	$T_{noc}$	6.9964
$T_{hess}(10)$	1.2060		
$T_{hess}(20)$	7.4160		
$T_{hess}(50)$	96.046		

Inserting these timings in (1) and (4) for *GMRES(m)*, in (2), (5) and (6) for *modGMRES(m)* version 1, for  $m = 10$  or  $20$ , and in (3), (5) and (6) for *modGMRES(m)* version 2, for  $m = 50$ , leads to the following tables:

m	time (ms)		time diff. (%)
	<i>modGMRES(m)</i>	<i>GMRES(m)</i>	
10	126.12	138.49	10
20	313.43	385.91	23
50	1390.2	1832.3	32

m	efficiency (%)	
	<i>modGMRES(m)</i>	<i>GMRES(m)</i>
10	66	60
20	66	54
50	63	47

These theoretical efficiency figures are based upon the model described above, with 13500 grid blocks, a processor grid of 150 processors and one master processor. The number of communication steps for accumulating a distributed value is 24 and the maximum number of blocks on a processor is 105. Furthermore communication costs, other than in the global accumulation of distributed values, are negligible. For *GMRES(10)*, *GMRES(20)*, *modGMRES(10)* and *modGMRES(20)* the overall timings were also determined experimentally on the busiest processor, which leads to the following table, containing the average time per iteration:

m	time (ms)		time diff. %
	<i>modGMRES(m)</i>	<i>GMRES(m)</i>	
10	126.09	137.13	9
20	314.58	383.94	22

From these tables it is obvious that the *modGMRES(m)* method can yield a substantial improvement in time per iteration. Furthermore, for these simulations, there was no difference in the total number of iterations between *GMRES(m)* and *modGMRES(m)*. With respect to the (theoretical) parallel efficiency, it can be seen that, for increasing  $m$ , the decrease in efficiency of *modGMRES(m)* is much less than for *GMRES(m)*. This is explained by the fact, that the communication costs increase exponentially with  $m$ , and these are much higher for *GMRES(m)* than for *modGMRES(m)*.

### References

- [1] Chronopoulos A.T., Gear C.W., *s*-step iterative methods for symmetric linear systems, *J. Comp. Appl. Math.* 25 (1989) 153-168 North-Holland
- [2] van Daalen D.T., Hoogerbrugge P.J., Meijerink J.A., Zeestraten R.J.A., Publication 924, Koninklijke/Shell Exploratie en Productie Laboratorium Rijswijk, The Netherlands, Shell Research R V 1989
- [3] Saad Y., Schultz M.H., *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*, *SIAM J.Sci.Stat.Comp* 7 (no. 3) July 1986