Communication Cost Reduction for Krylov Methods on Parallel Computers

E. de Sturler[1] and H. A. van der Vorst[2]

[1] Swiss Scientific Computing Center CSCS-ETHZ, Via Cantonale, CH-6928 Manno, Switzerland
[2] Mathematical Institute, Utrecht University, Budapestlaan 6, Utrecht, The Netherlands

Abstract. On large distributed memory parallel computers the global communication cost of inner products seriously limits the performance of Krylov subspace methods [3]. We consider improved algorithms to reduce this communication overhead, and we analyze the performance by experiments on a 400-processor parallel computer and with a simple performance model.

1 Introduction

In this paper, we study possibilities to reduce the communication overhead introduced by inner products in Krylov subspace methods and the influence of this reduction on the performance. We use the iterative methods GMRES(m) [7] and CG [6] as representatives for the two different classes of Krylov subspace methods (i.e. with short and with long recurrences).

We will restrict ourselves to problems that have a strong data locality, which is typical for many finite difference and finite element problems. A suitable domain decomposition approach preserves this locality more or less independent of the number of processors, so that the matrix vector product requires communication with only a few nearby processors.

2 Reformulation of Algorithms

We investigate two ways of improvement. The first way is to reschedule the operations such that we can overlap communication with computation. For CG this is done without changing the numerical stability of the method [2]. For GMRES(m) it is achieved by reformulating the modified Gram–Schmidt orthogonalization (MGS) [3, 4], after generating a basis for the Krylov subspace using polynomials for stability. The second way, for GMRES(m), is to assemble the results of the local inner products of a processor in one message and accumulating them collectively.

For GMRES(m), the steps that differ from the standard algorithm are given in Fig. 1. A good strategy for the selection of the parameters d_i (for the polynomials) is discussed in [1]. We can implement the MGS by first orthogonalizing all

```
{ create polynomial basis: }
v̂_1 = v_1 = r/‖r‖_2
for i = 1, 2, ..., m do
    v̂_{i+1} = v̂_i − d_i A v̂_i
end

{parallel modified Gram-Schmidt: }
for i = 1, 2, ..., m do
    split v̂_{i+1}, ..., v̂_{m+1} into two blocks
    local inner products (LIPs) block_1

    ‖ { accumulate LIPs block_1
      { compute LIPs block_2

    update v̂_{i+1}; LIP for ‖v̂_{i+1}‖_2;
    place this LIP into block_2

    ‖ { accumulate LIPs block_2
      { update vectors block_1

    update vectors block_2
    normalize v̂_{i+1}
end
```

Fig. 1. The generation of the basis vectors and the implementation of the MGS

```
parCG:

    Choose x_{-1} = x_0;
    r_0 = b − A x_0;
    p_{-1} = 0; α_{-1} = 0;
    s = L^{-1} r_0;
    ρ_{-1} = 1;
    for i = 0, 1, 2, ... do
(1)     ρ_i = (s, s);
        w_i = L^{-T} s;
        β_{i-1} = ρ_i/ρ_{i-1};
        p_i = w_i + β_{i-1} p_{i-1};
        q_i = A p_i;
(2)     γ = (p_i, q_i);
        x_i = x_{i-1} + α_{i-1} p_{i-1};
        α_i = ρ_i/γ;
        r_{i+1} = r_i − α_i q_i;
(3)     compute ‖r‖;
        s = L^{-1} r_{i+1};
        if accurate enough then
            x_{i+1} = x_i + α_i p_i
            quit
    end;
```

Fig. 2. The parCG algorithm

basis vectors on the first vector, then on the second, and so on, because we have all the basis vectors available. This permits the collective accumulation of large groups of inner products instead of the one-by-one accumulation in standard GMRES(m) with MGS. We can improve the MGS even further by splitting the orthogonalizations on one vector in two groups. Then we can overlap the accumulation for one group with the computations for the other group, as is shown in Fig. 1. We refer to this version of GMRES(m) as parGMRES(m).

We follow the approach suggested in [2] to reduce the communication overhead for preconditioned CG. We assume that the preconditioner K can be written as $K = LL^T$. We overlap the communication in the inner products at lines (1), (2) and (3) with the computations in the following line; see Fig. 2. We split the preconditioner to create an overlap for the inner products (1) and (3), and we create extra overlap possibilities by doing the update for x corresponding to the previous iteration step after the inner product (2). We refer to this version of CG as parCG.

3 Performance Model

We will model the performance of one cycle of GMRES(m) and parGMRES(m), and of one iteration of CG and parCG using equations derived from [5]. The purpose of this model is to provide insight in the parallel performance, not to give very accurate predictions. Most notably it ignores the communication in the matrix-vector product. For GMRES(m) this is not important, but for CG this leads to predictions that are too optimistic. The runtime of GMRES(m) is given by

$$T_P = \left(2(m^2+3m) + 4n_z(m+1)\right) t_{fl} \frac{N}{P} + \left((m^2+3m)(t_s+3t_w)\right)\sqrt{P}, \qquad (1)$$

where N is the total number of unknowns, P is the number of processors, m is the restart parameter, n_z is the average number of non-zeroes per row of the matrix and in the preconditioner (ILU or LL^T), t_{fl} is the (average) time for a double precision floating point operation, and t_s and t_w are the message start-up time and the word transmission time (between neighbouring processors). The factor $\sqrt{P}$ comes from the use of a square processor grid; it is the order of the diameter of the processor grid (graph). The runtime of the parGMRES(m) version is given by

$$\begin{aligned} T_P = {} & (m^2+4m+4n_z(m+1))t_{fl}\frac{N}{P} + \\ & \max\left((m^2+2m)t_{fl}\frac{N}{P} + 16mt_s + (8m^2+40m)t_w \ , \right. \\ & \left. (4mt_s + (2m^2+10m)t_w)\sqrt{P} \right), \end{aligned} \qquad (2)$$

where max() depends on whether there is enough local work to overlap all communication. The runtime of CG is given by

$$T_P = (9+4n_z)t_{fl}\frac{N}{P} + 6(t_s+3t_w)\sqrt{P}. \qquad (3)$$

The runtime of parCG is given by

$$T_P = (9+2n_z)t_{fl}\frac{N}{P} + \max(2n_z t_{fl}\frac{N}{P} + 24(t_s+3t_w), 6(t_s+3t_w)\sqrt{P}). \qquad (4)$$

The speedups curves from the model are given in Figs. 3 and 4. We used the following parameter values (from the experiments), $N = 10000$, $m = 50$ for (par)GMRES(m), $n_z = 5$, $t_{fl} = 3.00\mu s$, $t_s = 5.30\mu s$, and $t_w = 4.80\mu s$.

The speedup of GMRES(m) levels off very quickly due to increasing communication costs. The speedup of parGMRES(m) stays very close to perfect speedup until the number of processors reaches the point where we can no longer overlap all communication (the acute point); this is where the two arguments of the max() in (2) are equal. We see that the speedup of parGMRES(m) then also starts to level off.

For CG and parCG, we see the same effects as for (par)GMRES(m). The number of processors where we can no longer overlap all communication is smaller, because we have less computation to overlap with.

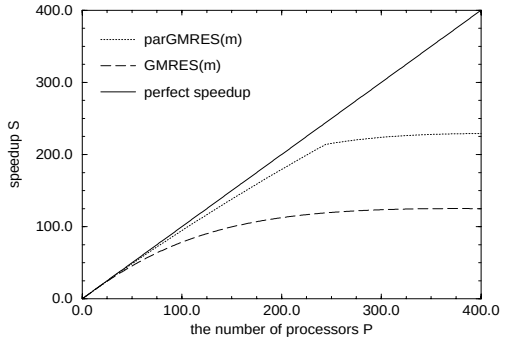

Fig. 3. The modeled performance of GMRES(m) and parGMRES(m)

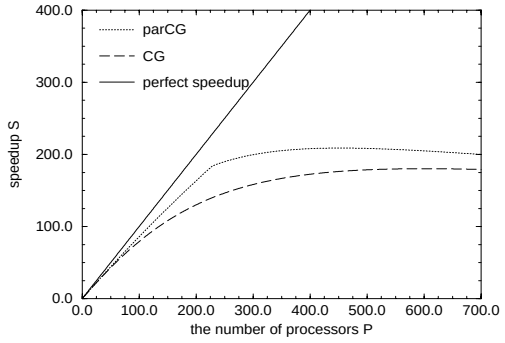

Fig. 4. The modeled performance of CG and parCG

4 Results

Below, we give results on a 400-transputer (T800) Parsytec Supercluster at the Koninklijke/Shell-Laboratorium in Amsterdam. The processors are connected in a fixed 20×20 mesh, of which arbitrary submeshes can be used. We have solved a convection diffusion problem (only diffusion for CG) with 10000 unknowns discretized by finite volumes, resulting in the familiar five-diagonal matrix.

Table 1 gives the measured runtimes for one cycle, the speed-ups, and the efficiencies for GMRES(50) and parGMRES(50). The speed-ups and efficiencies are computed relative to the estimated, sequential runtime of the GMRES(50) cycle, $T_1 = 190.$ s, because the problem did not fit on a single processor. The cost of communication spoils the performance of GMRES(m) completely for large P, as shown in Fig. 3. The performance of parGMRES(m) is much better. The speedups for 100 and 196 processors are almost optimal, because all communication can be overlapped. As predicted by the performance model, we

Table 1. measured runtimes for GMRES(m) and parGMRES(m)

processor grid	GMRES(50)			parGMRES(50)		
	T (s)	E (%)	S	T (s)	E (%)	S
10×10	2.47	76.8	76.8	1.93	98.2	98.2
14×14	1.90	50.9	99.8	1.05	92.1	181.
17×17	1.66	39.5	114.	0.891	73.6	213.
20×20	1.75	27.1	108.	0.851	55.7	223.

cannot overlap all communication for 289 processors, and the efficiency starts to decrease; for 400 processors the speedup is not much better. Note that, except for 100 processors, the runtime of GMRES(50) is about twice that of parGMRES(50). The estimated runtimes for GMRES(50) and parGMRES(50) are given in Table 2. A comparison with the measured timings indicates that the model is quite accurate. For 400 processors, neglected costs start playing a role.

Table 3 gives the measured runtimes for one iteration step, the speedups, and the efficiencies for CG and parCG. The speedups and efficiencies are computed relative to the measured, sequential runtime of the CG iteration, which is given by, $T_1 = 0.788$s. We observe that the performance of CG also levels off quickly, even though the number of inner products is small. This is in agreement with the performance model; see also [3]. As indicated in Fig. 4, if we increase P the difference in runtime between CG and parCG increases until we can no longer overlap all communication, then the difference decreases again. In the parCG algorithm, we only try to overlap communication, mainly by overlap with the preconditioner. The preconditioner used is not very expensive,so the potential improvement is relatively small. Note, however, that this improvement comes virtually for free. Moreover, when the computation time for the preconditioner is large or even dominant, the improvement may also be large. For many problems this may be a realistic assumption.

In Table 4 we show estimates for the runtimes of the CG algorithm and the parCG algorithm. Just as for (par)GMRES(m), the estimates for (par)CG are relatively accurate except for the 20×20 processor grid. Again, this is probably

Table 2. Estimated runtimes for GMRES(m) and parGMRES(m)

processor grid	GMRES(50) (s)	parGMRES(50) (s)
10×10	2.42	2.01
14×14	1.70	1.08
17×17	1.54	0.853
20×20	1.52	0.828

Table 3. Measured runtimes for CG and parCG, speed-up and efficiency compared to the measured, sequential runtime of CG

processor grid	CG			parCG		
	T (ms)	S	E (%)	T (ms)	S	E (%)
10×10	10.7	73.6	73.6	10.2	77.3	77.3
14×14	6.90	114.	58.3	5.84	135.	68.8
17×17	6.09	129.	44.8	5.29	149.	51.5
20×20	5.59	141.	35.2	5.04	156.	39.1

Table 4. Estimated runtimes for CG and parCG

processor grid	CG (ms)	parCG (ms)
10×10	10.7	10.0
14×14	6.66	5.57
17×17	5.71	4.66
20×20	5.00	4.25

caused by costs neglected in the model.

5 Conclusions

We have studied the implementation of GMRES(m) and CG for distributed memory parallel computers. These algorithms represent two different classes of Krylov subspace methods, and their parallel properties are quite representative. The experiments show how the global communication in the inner products degrades the performance on large processor grids. This is also indicated by our performance model. We have considered alternative algorithms for GMRES(m) and CG in which the actual cost for global communication is decreased by reducing synchronization and start-up times, and overlapping communication with computation. Our experiments indicate this to be a successful approach.

References

1. Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. Technical Report 91-03, University of Kentucky, 1991.
2. J. W. Demmel, M. T. Heath, and H.A. van der Vorst. Parallel numerical linear algebra. *Acta Numerica* Vol 2, Cambridge Press, New York, 1993
3. E. De Sturler. A parallel restructured version of GMRES(m). Technical Report 91-85, Delft University of Technology, Delft, 1991.
4. E. De Sturler. A parallel variant of GMRES(m). In R. Vichnevetsky, J. H. H. Miller, editors, *Proc. of the 13th IMACS World Congress on Computation and Applied Mathematics* , IMACS, Criterion Press Dublin 1991, pp 682–683.
5. E. De Sturler and H. A. Van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on Parallel Distributed Memory Computers. Technical Report 832, Mathematical Institute, University of Utrecht, Utrecht, 1993
6. M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1954.
7. Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

This article was processed using the LaTeX macro package with LLNCS style