On the implementation of incomplete factorization preconditioning on workstation clusters

Y. Notay *
Service de Métrologie Nucléaire
Université Libre de Bruxelles (C.P. 165)
50, Av. F.D. Roosevelt, B-1050 Brussels, Belgium.
email : ynotay@ulb.ac.be

September 1994

The linear solver is an important numerical kernel in many real life applications. When the system is symmetric positive definite and arises from discrete PDEs, the conjugate gradient is a method of choice. Its efficiency relies however on the preconditioning, and the parallelization brings a tradeoff, the best sequential preconditioners being difficult to parallelize while the truly parallel algorithms compare unfavourably in term of global arithmetic work.

In particular, the parallelization on workstation clusters running PVM appears especially challenging, since each processor is generally sufficiently powerful to run efficiently sequentially, while the communications remain relatively slow compared with the speed of floating point calculation.

Description of the method

The method we aim at presenting here is based on the following idea.

Considering a PDE on a given domain Ω, the latter is divided in n_t non overlapping subdomains, and we assume that, by performing local discretisations on the different subdomains, it results a set of "local system matrices" A_t and right hand side b_t, $t = 1, \ldots, n_t$, such that the global system matrix and right hand side correspond to their assembly.

However, in our scheme, this global system is never explicitly assembled. Rather, we consider as system matrix the block diagonal matrix A_d whose diagonal blocks are the different "local" matrices :

$$A_d = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_{n_t} \end{bmatrix} . \qquad (1)$$

*Supported by the "Fonds National de la Recherche Scientifique", Chercheur qualifié.

Accordingly, we define as right hand side

$$b_d \;=\; \begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix} . \tag{2}$$

To these quantities corresponds an extended variable set in which internal boundary nodes are represented a number of times equal to the number of subdomains to which they belongs.

This leads us to define on this extended set the operator Σ :

$$\Sigma \;:\; \forall x \;:\; \forall i \;:\; (\Sigma x)_i \;=\; \sum_{\substack{j \text{ corres-} \\ \text{ponding to the} \\ \text{same node as } i}} x_j \quad , \tag{3}$$

which corresponds thus to a basic communication routine exchanging each component of a given vector for the sum of the different components corresponding to the same gridpoint.

Then, it can be proven [1] that CG iterations performed on this extended variable set with A_d as system matrix, b_d as right hand side, and the multiplication by Σ as preconditioning step, are completely equivalent to unpreconditioned CG iterations performed on the original system. By equivalent, we mean that each component of the approximate solution vector on the extended set is at each step equal to the value that one would obtain at the corresponding gridpoint by performing regular iterations on the original system.

This is however just a way, maybe convenient but not necessarily optimal, to implement the unpreconditioned CG method. The point of interest is that this approach still holds when considering preconditioning steps of the form

$$g \;=\; U_d^{-1} \Sigma L_d^{-1} \; r \tag{4}$$

where U_d and L_d are respectively upper and lower triangular, and block diagonal similarly to A_d, so that they involve only purely local computation. Under given conditions on these matrices, the CG solution will be here equivalent to a CG solution performed on the original system with incomplete factorization preconditioning.

These conditions are not difficult to meet provided that one considers domain decomposed orderings, i.e. that one numbers separately the interior nodes of each subdomain, reserving the last numbers for the internal boundary nodes. This implies that the preconditioner acts more locally than with a standard lexicographic ordering, and thus some deterioration of the convergence rate.

This is the price to pay to keep, for this preconditioned version, exactly the same amount of communication per iteration as for the unpreconditioned version. The important is that we don't pay too much : the locality in the preconditioner is just that needed to allow parallelism on the given number of processors, and the number of iterations remains far less than that required with classical purely parallel preconditioners.

When the discretisation is sufficiently regular, it is even possible to improve the method by ordering part of the boundaries before rather than after the interior nodes, which decreases the locality in the corresponding approximate factorization. According to this, Σ may be splitted :

$$\Sigma \; = \; \Sigma_\ell \, \Sigma_f \; , \tag{5}$$

and the preconditioning step writes then

$$g \; = \; \Sigma_f U_d^{-1} \Sigma_\ell L_d^{-1} \Sigma_f \; r \; . \tag{6}$$

Numerical resuls

The method has been implemented on a 8 workstation IBM RS6000 cluster running PVM. Results for some test problems are given on Table 1. All problems arise from discrete second order elliptic PDEs on square (2D) or cubic (3D) domains, and the total number of unknowns is about h^{-d}, where d is the space dimension. We refer to [2] for more details and additional results.

In the 1 processor case, the method reduces to a standard CG solution with incomplete factorization preconditioning with respect a lexicographic ordering, which may fairly be considered as a reference method within the framework of this family of method, and thus serve as basis for the appreciation of the parallel version.

Looking at the CPU times, it turns out that the parallelization is especially interesting when the memory requirements of the sequential version reach the available in core memory.

On the other hand, efficiencies may appear relatively poor for the smallest load per processor, but one has to take into account that this concerns only cases where the program executes sequentially in less than a very few seconds, i.e. cases for which the interest of the parallelization is anyway doubtful.

Finally, we obtain efficiencies slightly less than 1 in intermediate cases, showing that our method allows a successful parallelization even below the limit where the sequential version suffers from the recourse to virtual memory. This conclusion could be strengthened if the memory per processor were higher. Indeed, we had here 32 MB per node, which is relatively small for such powerful processors and such memory intensive computations.

This shows that the communication problem has been well mastered by the proposed technique, while the varations in the convergence rates are not significant for the considered numbers of processors.

We also report on the real time needed to execute the program. Indeed, although it is subject to large variations from run to run, this is one of the parameter the user is interested in when parallelizing an application. Moreover, it appears that part of the costs inherent to the execution of the program are hidden when considering only the user CPU time, maybe because a process which is waiting for data is automatically unloaded even if no other program is running on the machine. So, the tendency to have a low use of CPU time when parallelizing with small load per processor on the one hand, or when resorting to virtual memory on the other hand, has been confimed by various runs.

2D PROBLEM A

h^{-1}	144			288			576		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	1.36	(38)	39%	7.92	(56)	38%	47.7	(82)	30%
4	.333	(32)	12%	1.92	(48)	21%	10.5	(72)	29%
8	.246	(35)	6%	1.09	(51)	16%	5.94	(76)	22%

2D PROBLEM B

h^{-1}	144			288			576		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	2.20	(60)	25%	12.5	(88)	46%	76.8	(131)	44%
4	.635	(54)	17%	3.19	(79)	47%	16.8	(115)	70%
8	.514	(55)	8%	1.88	(82)	18%	9.41	(120)	39%

3D PROBLEM A

h^{-1}	36			72		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	2.24	(23)	14%	96.9	(33)	4%
8	.321	(19)	12%	3.61	(31)	32%

3D PROBLEM B

h^{-1}	36			72		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	4.35	(39)	44%	195.	(59)	2.7%
8	.644	(36)	13%	6.61	(54)	32%

Table 1: Solution time (in seconds) and number of iterations for tests problems on IBM cluster. %CPU is the ratio between the CPU time spent for the solution part (reported in the first column) and the total elapsed time during the execution of the whole program

Acknowledgements

This work presents research results of the Belgian Incentive Program "Information Technology" - Computer Science of the future, initiated by the Belgian State - Prime Minister's Service - Federal Office for Scientific, Technical and Cultural Affairs (Contract No. IT/IF/14). The Scientific responsability is assumed by its author.

This work was also supported by IBM through a research contract between ULB and IBM.

References

[1] Y. Notay, *On the parallelization of approximate factorization preconditioning*, Tech. Rep. Report IT/IF/14-12, Université Libre de Bruxelles, 1993.

[2] Y. Notay, *An efficient parallel discrete PDE solver*, Tech. Rep. Report NM-IBM 94-01, Université Libre de Bruxelles, 1994.