Theor Chim Acta (1991) 79: 169–174

Theoretica Chimica Acta

Designing portable parallel software for linear algebra

P. R. Amestoy, M. J. Daydé, and I. S. Duff
CERFACS, 42 Av. G. Coriolis F-31057 Toulouse Cedex

Received September 1, 1990/Accepted November 13, 1990

Summary. In the last decade, the development in computer architectures has strongly influenced and motivated the evolution of algorithms for large-scale scientific computing. The unifying theme of the parallel algorithm group in CERFACS is the exploitation of vector and parallel computers in the solution of large-scale problems arising in computational science and engineering. The choice of a portable approach often leads to a loss in the average performance per computer with respect to a machine dependent implementation of the code. However, we show that, in full linear algebra as well as in sparse linear algebra, efficiency and portability can be combined. To illustrate our approach, we discuss results obtained on a wide range of shared memory multiprocessors including the Alliant FX/80, the IBM 3090E/3VF, the IBM 3090J/6VF, the CRAY-2, and the CRAY Y-MP.

Key words: Gaussian elimination – Sparse linear equations – Multifrontal method – Vectorization – Parallelization – Elimination tree

1. Introduction

We consider the direct solution of linear equations:

$$Ax = b$$

on parallel vector computers with a global shared memory. Designing a portable approach and understanding the influence of computer architecture on the performance of our codes is a very important aspect of our work. We thus analyse results obtained on a variety of computers including the eight processor Alliant FX/80, the three processor IBM 3090E/3VF, the six processor IBM 3090J/6VF, the four processor CRAY-2, and the eight processor CRAY Y-MP. Even though our target computers all belong to the same class of shared memory multiprocessors, a more precise look at the architectures shows significant differences. Architectural differences will then be used to analyse the behaviour of the algorithms and to evaluate the impact of the architecture on the performance.

One key idea for combining efficiency with portability in linear algebra is to use basic linear algebra kernels [9]. These kernels are termed the BLAS, for Basic Linear Algebra Subprograms. Different levels of BLAS are available. The level terminology arises from the fact that if the vectors and matrices involved are of order N, the Level 1 BLAS provides vector computations of order $O(N)$, the Level 2 BLAS provides matrix-vector computations of order $O(N^2)$, and the Level 3 BLAS provides matrix-matrix computations with $O(N^3)$ operations. The architecture of shared memory parallel computers uses a hierarchy of memory (shared memory, local memory, vector registers, ...). All the arithmetic computations are performed at the top of this hierarchy. Therefore the key to efficiency is to keep active data as close as possible to the top of hierarchy. The use of higher Level BLAS provides this capability (see [10] and [11]) since the ratio number of operations over number of memory references increases with the level of the BLAS. One can consider that the main feature of BLAS is to mask the details of the architecture while providing high performance. The increased granularity of higher Level BLAS also allows more efficient parallelization. Note that parallel BLAS libraries should be implemented in the next few years, especially in conjunction with the forthcoming LAPACK Library [8]. A complete study of the efficient implementation of the Level-3 routines is available in [6] and [7] and we show, in this paper, how it can be used to design portable parallel software for linear algebra.

2. Parallelization of full linear algebra

We consider the solution of dense systems of equations $Ax = b$. The idea is to express the LU factorization in terms of a block algorithm using Level 3 BLAS [10]. To simplify our discussion we concentrate on the parallel implementation of one of the block LU factorization, the KJI-SAXPY algorithm [6]. We have studied, in [7], two approaches for parallelizing the KJI-SAXPY algorithm. The simplest one exploits the parallelism only within the BLAS and will be referred to as *parallel* BLAS. Another possibility, which provides more parallelism, consists in parallelizing the blocked KJI-SAXPY algorithm. This will be referred to as the *parallel LU* version of the algorithm. We compare, in Table 1, the parallelism obtained within the computational kernels with that obtained over the kernels.

In Table 1, *nproc* designs the number of available processors. We also report the performance of various manufacturer supplied routines: PDGEFA from the Para-Linpack/FX Library on the Alliant, SGEFA from the CRAY SCILIB

Table 1. Performance in Megaflops of the KJI-SAXPY algorithm on a full 1000-by-1000 system; manufacturer library performance is given as reference

		Multiprocessor version			Manufacturer library	
Computer	*nproc*	1 proc	Parallel BLAS	Parallel *LU*	Routine	Mflops
Alliant FX/80	8	11	61	57	PDGEFA	39
IBM 3090J/6VF	6	89	294	418	DGEF	97
CRAY-2	4	355	802	831	SGEFA	353
IBM 3090E/3VF	3	62	132	182	DGEF	72

Library, and DGEF from the IBM ESSL Library. These results show that though an optimal implementation depends on the characteristics of the target computer, a portable and efficient code can be designed using Level 3 BLAS. The parallel version of LU factorization provides, in average, only relatively small performance improvement over the version using only parallel BLAS. On both IBMs the gain in using a parallelized algorithm is evident but it is still an open question to decide whether it outweigh the loss of portability. On the Alliant FX/80, we can efficiently parallelize all the kernels, including Level 1 BLAS, using microtasking. Because of this, we notice in Table 1 that the parallel version of the LU is not faster than the version using only parallel BLAS. The Level 3 BLAS can be used to exploit parallelism and a very high uniprocessor performance can be combined with efficient parallelization. Of course, the performance obtained is closely linked to the efficiency of the Level 3 BLAS available on the target computer. The efficient implementation of BLAS depends on the availability of low cost synchronization tools. The Alliant loop-level parallelism is a good example of this. In this case the parallelization of the BLAS based on loop-level parallelism is generally straightforward.

3. Sparse LU factorization based on a multifrontal approach

We then examine the case when A is sparse. Our algorithm is based on a multifrontal approach introduced by Duff and Reid (see [3] and [4]). One important feature of the multifrontal method is that the nonzeros in the pivot row and column have already (in an assembly step) been gathered into a small dense submatrix, so called a *frontal matrix*, so that all updating operations can be performed with a regular stride. Furthermore, in a multifrontal approach, the independence of the successive steps of elimination due to sparsity is expressed in terms of a dependence graph referred to as *the elimination tree*. Each edge of the elimination tree corresponds to an assembly while each node is associated with an elimination process on a frontal matrix.

The degree of parallelism coming from the elimination tree is combined with a node level parallelism which exploits parallelism within Level 3 BLAS. The tuning of the code is then handled through machine dependent parameters designed to take into account the main architectural differences between our target computers.

We analyse, in Table 2, the influence of parallelism within Level 3 BLAS on the global parallelism of the method. We report in Table 2 results obtained on

Table 2. Speedup study of the multifrontal factorization on a medium-size sparse matrix

	1 proc.	3 procs		4 procs		6 procs		8 procs	
	Mflops	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
IBM 3090E	44	1.9	2.4						
CRAY-2	176	1.7	2.0	1.8	2.3				
IBM 3090J	60	1.9	2.5	2.0	3.0	2.1	3.8		
CRAY Y-MP	216	1.9	2.7	2.1	3.3	2.3	4.1		
Alliant FX/80	8	1.7	2.4	1.8	3.0	1.9	3.9	1.9	4.3

In columns (1) we exploit only parallelism from the tree; in columns (2) we combine the two levels of parallelism

a medium-size sparse matrix (3948 × 3948) from the Harwell-Boeing set of sparse matrices [5]. Our vectorized version of the multifrontal code [2], leads to high uniprocessor performance (column (2) of Table 2) between 1/3 and 1/2 of the peak performance of a single processor. We notice, in Table 2, the importance of the second level of parallelism which balances that lost from the elimination tree. The main reason why the additional level of parallelism is so effective is that the frontal matrices are always larger near the root and so the main benefit from nodal parallelism (through Level 3 BLAS) occurs just when the parallelism from the tree becomes less.

Table 2 also illustrates some of the architectural differences between the target computers. We notice the good behaviour of the IBMs in a parallel mode. The second level of parallelism makes great use of the local cache of the IBMs, and we observe a small speedup improvement on three processors of the IBM 3090J/6VF with respect to the IBM 3090E/3VF which is certainly due to the increase in the size of the local cache on the IBM 3090J/6VF. Obviously, although the CRAY-2 is a very powerful vector computer, it has some limitations in its use in a multiprocessor mode. The relatively poor speedup obtained on the CRAY-2 mainly comes from the increase in the memory access conflicts when running in a multiprocessor environment. The Alliant FX/80 is not a very impressive vector computer and we reach only a third of the peak performance on one processor. We also notice in columns (1) of Table 2 that the potential parallelism of the elimination tree is not fully exploited by the Alliant FX/80. In fact, on the Alliant FX/80, the management of parallel sections is based on binary semaphores which are managed by software and are quite costly. Furthermore, the design of the shared cache memory of the Alliant FX/80 is also a possible reason for the increase in memory conflict in a multiprocessor environment. However, because of the small minimum task granularity of the Alliant FX/80, good speedup is obtained with the second level of parallelism (see columns (2) in Table 2). Last, but not least, is the CRAY Y-MP on which we observe very high uniprocessor performance (more than two-thirds of the peak performance) combined with quite impressive speedup ratios. Note that this is the only computer on which we did not run in dedicated mode so that better performance in multiprocessor mode might be obtainable. On a full set of results, we can illustrate better the main architectural differences between our target computers [1].

With a multifrontal approach, the numerical factorization involves assembly steps and full factorization of the frontal matrices. The assembly step requires a small number of costly indirect floating-point operations and index-searching operations. Therefore, to further improve the run time, one can build an elimination tree which involves larger frontal matrices and less indirect operations. One way to increase the ratio of the nodal work (full eliminations) and the work in the assembly is to allow amalgamation of adjacent nodes of the elimination tree. Although this can lead to more entries in the factors and more overall operations, it can be easily controlled in our portable code by a single parameter. This was exploited in [3] to improve vectorization on the CRAY-1 and has been explored in depth in [1].

We notice, in Fig. 1, that the percentage decrease in the number of operations involved in the assembly process changes faster than the percentage increase in the total number of operations. We also observe, in Fig. 1, that the number of nodes in the elimination tree decreases sharply for low levels of node amalgamation. Moreover, the assembly process involves, on average, three

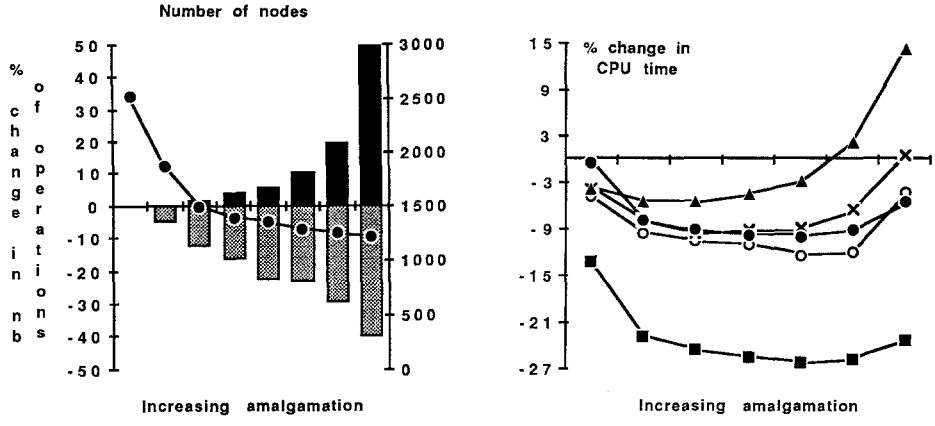

Fig. 1. Study of the effect of node amalgamation on the number of bperations. ■ % change in the total number of operations; ▨ % change in the number of operations in the assembly; · ● - number of nodes in the elimination tree

Fig. 2. Sensitivity to node amalgamation. 1 processor of: · ■ - CRAY Y-MP; · ○ - CRAY-2; × - IBM 3090/VF; · ● - ETA 10-P; - ▲ - Alliant FX/80

memory accesses per floating-point operation so that, even if we increase the overall number of operations, we can obtain an improvement with this node amalgamation technique. This gain comes from the difference between the Megaflop rate of the assembly process and the Megaflop rate of the elimination process [1].

Figure 2 shows that the five target computers are affected differently by node amalgamation. We notice that, in terms of maximum percentage improvement, the Alliant FX/80 and the CRAY Y-MP behave in an opposite way, the IBM 3090E/3VF and the ETA 10-P perform identically, while CRAY-2 is slightly better. We obtain around 10% CPU time improvement on one processor of the IBM 3090E/VF, the ETA 10-P, and the CRAY-2 while on the Alliant FX/80 we only observe a 5 percent decrease in the CPU time. On the CRAY Y-MP, we reach more than 25 percent CPU time improvement. We observe in Fig. 2 that, after a certain amount of fill-in, we can go on amalgamating nodes on the CRAY-2, the ETA 10-P and the CRAY Y-MP, while, on the IBM 3090E/VF and the Alliant FX/80, the CPU time already starts to increase. Furthermore, in a parallel environment, we notice [1] that node amalgamation assists the parallelism of the method although this must be considered as a side effect of the amalgamation strategy and should not be in general expected.

4. Concluding remarks

We have studied the design of parallel software for linear algebra. Therefore, apart from parallel extensions to the Fortran language, we have run the same code on all our target computers. The choice of a portable approach often leads to a loss in the average performance per computer with respect to a machine dependent implementation of the code. However, we have shown that, in linear algebra, one can combine portability and efficiency both in terms of Megaflop rates and speedups on a large range of shared memory multiprocessor machines.

The key idea for combining efficiency with portability has been the use of Level 3 BLAS. In full LU factorization, we have shown that parallel BLAS captures a large part of the potential parallelism of the block LU factorization. In sparse matrix factorization, the parallelization of the BLAS can be exploited to provide an additional level of parallelism that balances that coming from the elimination tree. We have also explained how we can enhance vectorization of the multifrontal method by amalgamating nodes of the elimination tree, even if this amalgamation introduces additional fill-in in the factors. A machine dependent parameter has been introduced to control the amount of modification of the elimination tree and we have noticed that its optimal value is a function of the ratio between the performance of indirect and direct operations of the target computer. Finally, we achieve 890 Megaflops for a sparse solver with a medium size sparse matrix on the CRAY Y-MP using 6 processors.

References

1. Amestoy PR (1990) Factorization of large unsymmetric matrices based on a multifrontal approach in a multiprocessor environment. PhD Thesis. CERFACS Report TH/PA/91/2
2. Amestoy PR, Duff LS (1989) Int J Supercomputer Appl 3(3):41
3. Duff LS, Reid JK (1983) ACM Trans Math Softw 9:302
4. Duff LS, Reid JK (1984) SIAM J Sci Stat Comput 5:633
5. Duff IS, Grimes RG, Lewis JG (1989) ACM Trans Math Softw 15:1
6. Daydé MJ, Duff LS (1989) Int J of Supercomputer Appl 3:40
7. Daydé MJ, Duff LS (1990) CERFACS report TR/TA/90/30
8. Demmel JW, Dongarra JJ, Du Croz J, Greenbaun A, Hammarling S, Sorensen DC (1987) Prospectus for the development of a linear algebra library for high performance computers. Report TM-97, Mathematics and Computer Science Division, Argonne National Laboratory
9. Dongarra JJ, Du Croz J, Hammarling S, Hanson RJ (1988) ACM Trans Math Softw 14:1 and 18
10. Dongarra JJ, Du Croz J, Duff LS, Hammarling S (1988a) ACM Trans Math Softw 16
11. Dongarra JJ, Du Croz J, Duff IS, Hammarling S (1988b) ACM Trans Math Softw 16