Package of Iterative Solvers for the Fujitsu VPP500 Parallel Supercomputer

Zbigniew Leyk
School of Mathematical Sciences
Australian National University
Canberra ACT 0200, Australia
e-mail: Zbigniew.Leyk@anu.edu.au

Murray L. Dow
Supercomputer Facility
Australian National University
Canberra ACT 0200, Australia
e-mail: M.Dow@anu.edu.au

Abstract

In the paper we report progress to date on our project to implement iterative methods on Fujitsu's VPP500 vector-parallel supercomputer.

1 Introduction

In many scientific computations we need to solve very large systems of linear equations. As examples we can mention systems resulting from discretization of partial differential equations by finite difference or finite element methods. Many of these systems cannot be solved by direct methods because of limitations of time and/or memory. Instead, iterative methods are used. Comparing with direct methods, they are often faster and do not have many restrictions with respect to memory. In addition, it is easier to implement iterative methods efficiently on high-performance computers.

We started to write iterative routines for the VPP500 in 1993 taking part in the Fujitsu-ANU Parallel Scientific Subroutine Library project. The first implemented methods were the preconditioned conjugate gradient method [1] and MGCR method [3]. The goal was to deliver routines of high quality and good performance to solve symmetric and nonsymmetric systems of linear equations. These routines were sent to Fujitsu in March 1994. At that time we decided to (re)implement iterative methods using the reverse communication technique in place of passing all data as subroutine parameters. This approach gives the developers and advanced users greater flexibility in writing application codes. For common users, user-friendly interfaces are provided. A similar approach is used in the Thinking Machines' CMSSL library [5].

During writing and testing the iterative routines we used the smallest configuration of the VPP500 with only 4 processors. However, the development has been aided by the use of two other Fujitsu computers: the VP2200 vector computer (the same vectorizing compiler as on the VPP500) and the AP1000 parallel computer with 128 nodes. Moreover, the code can run with no changes on sequential machines and on the VP2200 which is important for debugging purposes, and for checking vectorization and optimization of the code.

The package of iterative solvers which is described in the paper is planned to be incorporated into Fujitsu SSLII/VPP, Parallel Scientific Subroutine Library for the VPP500. The second author would like to thank D. Kincaid for his kind permission to use parts of ITPACK [2] and NSPCG [4] packages.

2 Overview of the Fujitsu VPP500 Supercomputer

Massively parallel processing (MPP) computers use a very large number (often thousands) of slow processors. This architecture has been supposed to deliver performance which cannot be achieved by conventional scalar and vector computers. However, recent benchmarks show that MPP computers have lower performance and exhibit inferior performance compared with high speed vector processing systems. There are many reasons for this situation. Among others are low network performance, dependence on cache, insufficient processor-memory bandwidth and also the fact that compilers on MPP computers are unable to generate highly parallel code.

The Fujitsu VPP500 vector-parallel supercomputer offers a different approach, called vector parallel processing, see [6]. It uses a small number of high speed vector processors as basic elements of the computer architecture. This gives much better performance than it can be obtained on MPP computers.

The VPP500 has a performance range of 6.4 to 355 Gflops and a main memory capacity from 1 to 222 Gbytes. The system scalably supports between 4 and 222 processors interconnected by a proprietary high-bandwidth crossbar network. Each processing element (PE) is a multiple-pipe vector computer and has a peak speed of 1.6 Gflops. The PE and control processor interconnection bandwidth is 400 Mbytes/s uni-directional or 800 Mbytes/s bi-directional per node. The choice of a crossbar for the VPP500 has many advantages comparing with other network topologies like tree, mesh or torus (see [6]).

The VPP500 is a fully MIMD machine with scalar, vector and data transfer units on each PE. It has distributed global memory as well as local memory implemented by dynamic partitioning of the main storage of each PE for each user process. The VPP500 has a single address space. References to the global memory are expensive and best used only for communication. Access to parallel facilities is given through VPP Fortran, a Fortran-77 based language with compiler directives to handle the parallel programming aspects. By inserting compiler directives, existing Fortran-77 codes can be ported and should be able to take advantage of the virtual global memory model.

3 Package of iterative methods for the VPP500

We want to solve a large system of linear equations of the form

$$Ax = f, \tag{1}$$

where A is a large (and often sparse) matrix of size $n \times n$. We solve (1) using polynomial (Krylov-based) iterative methods like conjugate gradient (CG) [1] or MGCR [3].

The success of linear algebra packages like LINPACK or LAPACK was mainly due to the fact that dense and banded matrices have only a few natural ways of storing them. Thus, it is easy to design the corresponding data structures which can be generally approved. For iterative methods it is quite the opposite. Appropriate data structures have to be chosen to store sparse matrices and data structures are often different for different matrices. Moreover, on parallel machines one must make decisions about how vectors and sparse matrices should be distributed across the parallel processors. Any such a decision depends on the application and architecture of the computer. This is probably the main reason of existence of so many packages of iterative methods on sequential and lately on parallel computers. It seems that the only way out of this situation is to use special software techniques. This can be reverse communication interface in Fortran-77 or so called "data format free" or "data structure neutral" approach for C, C++ and Fortran-90. Using these techniques we avoid specifying storage formats in iterative routines.

We have set several goals while designing the iterative solvers package for parallel computations. From our point of view the important features are: scalability, efficiency (high performance), adaptivity to the user requirements, extensibility, wide scope (wide range of input problems and data structures supported) and in less extent portability. Important numerical features are stability and robustness of iterative methods used. Although portability is not important for us it must be supported since we test our code not only on the VPP500 but often on other machines like the VP2200 or AP1000.

We have chosen the reverse communication interface (RCI) in implementing iterative methods. We need flexibility offered by RCI to design the package so that it is easily adaptable to the steadily advancing state of the art in the area of iterative methods where every year a few new iterative solvers and/or preconditioners are proposed. Moreover, we employ hierarchical design of modular components which facilitates customization according to users' needs. A common user does not need to know how RCI works since RCI is on a lower level than user's applications. The use of more than one level of interface means that simplicity is not sacrificed for the sake of more sophisticated users.

Methods which are implemented or are planned to be implemented in the package are the conjugate gradient, GMRES, MGCR, QMR, transpose-free QMR (TFQMR) and Bi-conjugate stabilized method. We want to stress that some iterative methods require not only matrix by vector Av products but also A^Tv products.

4 Storage formats for sparse matrices

Among the types of computations encountered on parallel machines, sparse computations are very challenging. One reason of this situation is that sparse matrices are often irregularly structured and this implies that costly irregular communications are needed between processors. The other reason is that in sparse matrix computations the number of arithmetic operations is often of the same order as number of data transfers. As a result, computation time can be dominated by communication time rather than arithmetic operations time. Therefore it is important to keep communication costs down.

We can distinguish three basic operations in polynomial iterative methods (without preconditioning). These are: inner products, vector update operations and matrix by vector products. There is no problems with parallelizing inner products and vector update operations. It can be done on almost any parallel machine in an efficient way. But it is not so with matrix by vector products. Such products depend on a storage format of the matrix and usually involve indirect addressing.

There are many ways of storing sparse matrices; probably as many as scientific applications involving such matrices. But only a few of them can be considered as having the best potential for parallel computations. The most promising is the diagonal format where nonzero diagonals of the matrix are stored as columns in a two-dimensional array. A separate array of offsets of each diagonal is necessary. The banded format is a special case of the diagonal format. Unfortunately, only a small class of matrices can be represented efficiently by the diagonal format. By efficiently we mean that a number of zeros which are saved (for instance for padding) is a small fraction of a number of all the elements saved. An advantage of the diagonal format is that it does not involve indirect addressing.

The Ellpack format is more general. In some way it is a generalization of the diagonal format. We store nonzero elements of each row of the matrix in a row of a two-dimensional array, and column indices to these elements are stored in a corresponding row of another two-dimensional (integer) array. This format can be inefficient if there are rows with a large number of nonzero elements and the rest of rows have only a few nonzero elements per row.

The coordinate format is most general and consists of three one-dimensional arrays: a real array containing the non-zero elements of the matrix (in any order), and two integer arrays containing corresponding row and column indices. (Because any matrix can be represented efficiently by this format it is commonly used in packages of iterative solvers.) The coordinate format is rather difficult for parallelization.

In our package we use all the above mentioned formats for storing sparse matrices. For any of these formats there exists at least one corresponding matrix by vector product.

To broadcast a global vector to local (non-distributed) vectors situated on each processor we use our communication routine which is based on standard VPP500 communication routines. The routine is the main tool used for computing the matrix by vector operation ($v = A * x$). The communication routines copies elements from a global vector to local vectors and only elements corresponding to the bandwidth elements of the matrix are copied. It gives very good performance and since global communication between processors is a bottleneck on the VPP500 (as well as other parallel computers) any improvement in communication costs gives an overall improvement of computations. If the bandwidth is large, a second communication method is provided, in which each processor packs and sends those elements that adjacent processors need.

5 Preconditioners

To speed up iterative methods we usually apply preconditioners. Unfortunately, many preconditioners accelerate iterative methods insignificantly. Better preconditioners used on sequential machines are incomplete LU and Cholesky factorizations, and relaxation type preconditioners as SOR and SSOR. They are not so efficient on parallel machines.

We have included or plan to include in the package of iterative solvers a variety of preconditioners. At this stage we want to determine which preconditioners for the conjugate gradient method are most efficient. The user will have full information on the performance of these preconditioners. The methods implemented so far are block Jacobi method, the method of Neumann truncated series and its block version, and block incomplete Cholesky method. The block size is chosen so that there is no interprocessor communication between different blocks. Block preconditioners trade off better parallel properties against slower convergence, since information is not propagated from processor to processor.

For nonsymmetric matrices we plan to implement the method of Neumann truncated series, incomplete LU factorization and wavefront ordering, and their block variants. We are also interested in implementation of multigrid and domain decomposition methods.

The basic operations in preconditioners are the same as in unpreconditioned iterative methods. Moreover, we have additional operations like solving diagonal or sparse triangular systems of equations. Permutations are sometimes necessary.

6 Conclusions

We are implementing iterative methods on the VPP500 parallel computer. During this process we have met with different kind of problems. It is easy to notice that performance on the VPP500 depends critically on the type of matrices taken for computations. In sparse computations, it is important to take advantage of the structure of the matrix. There can be a big difference between the performance obtained from a matrix stored in the diagonal format and one stored in a more general format. Therefore it is necessary to choose an appropriate format for a matrix used in computations.

Preliminary tests show that implementation of the package is scalable with respect to the number of processors, especially for large problems. It is becoming clear for us that the traditional efficient preconditioning techniques result only in a speedup of factor 2 at best. We need to look for new preconditioners more adjusted for parallel computations. The polynomial preconditioning approach is attractive because of the negligible preprocessing cost involved. We favour the reverse communication interface for added flexibility necessary for doing tests with different storage formats and preconditioners.

We can conclude that it is crucial to experiment with existing parallel machines to better understand the effects that are difficult to derive from theory, such as impact of communication costs or ways of storing data.

References

[1] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–435, 1952.

[2] D. Kincaid and T. Oppe. *Numerical Methods*, chapter ITPACK on supercomputers. Lecture Notes in Mathematics 1005. Springer-Verlag, 1982.

[3] Z. Leyk. Modified generalized conjugate residuals method for nonsymmetric systems of linear equations. In D. Stewart, H. Gardner, and D. Singleton, editors, *Proceedings of the 6th Biennial Conference on Computational Techniques and Applications: CTAC93*, pages 338–344, Singapore, 1994. World Scientific.

[4] T. Oppe, W. Joubert, and D. Kincaid. An overview of NSPCG: a nonsymmetric preconditioned conjugate gradient package. *Comput. Phys. Comm.*, 53:283–293, 1989.

[5] Thinking Machines Corp., Cambridge, Massachusetts. *CMSSL for CM Fortran: CM-5 Edition*, 1993.

[6] T. Utsumi, M. Ikeda, and M. Takamura. Architecture of the VPP500 supercomputer. In *Proceedings of Supercomputing '94*, Washington, D.C., 1994. To appear.