A Portable Implementation of ARPACK for Distributed Memory Parallel Architectures

K. J. Maschhoff and D. C. Sorensen

March 12, 1996

Abstract

ARPACK is a package of Fortran 77 subroutines which implement the Implicitly Restarted Arnoldi Method used for solving large sparse eigenvalue problems. A parallel implementation of ARPACK is presented which is portable across a wide range of distributed memory platforms and requires minimal changes to the serial code. The communication layers used for message passing are the Basic Linear Algebra Communication Subprograms (BLACS) developed for the ScaLAPACK project and Message Passing Interface(MPI).

1 Introduction

One objective for the development and maintenance of a parallel version of the ARPACK [3] package was to construct a parallelization strategy whose implementation required as few changes as possible to the current serial version. The basis for this requirement was not only to maintain a level of numerical and algorithmic consistency between the parallel and serial implementations, but also to investigate the possibility of maintaining the parallel and serial libraries as a single entity.

On many shared memory MIMD architectures, a level of parallelization can be accomplished via compiler options alone without requiring any modifications to the source code. This is rather ideal for the software developer.

For example, on the SGI Power Challenge architecture the MIPSpro F77 compiler uses a POWER FORTRAN Accelerator (PFA) preprocessor to automatically uncover the parallelism in the source code. PFA is an optimizing Fortran preprocessor that discovers parallelism in Fortran code and converts those programs to parallel code. A brief discussion of implementation details for ARPACK using pfa preprocessing may be found in [6]. The effectiveness of this preprocessing step is still dependent on how suitable the source code is for parallelization. Since most of the vector and matrix operations for ARPACK are accomplished via BLAS and LAPACK routines, access to efficient parallel versions of these libraries alone will provide a reasonable level of parallelization.

Unfortunately, for distributed memory architectures the software developer is required to do more work. For distributed memory implementations, message passing between processes must be explicitly addressed within the source code and numerical computations must take into account the distribution of data. In addition, for the parallel code to be portable, the communication interface used for message passing must be supported on a wide range of parallel machines and platforms. The Basic Linear Algebra Communication Subprograms (BLACS) [7] developed for the ScaLAPACK project provide a natural level of message passing for linear algebra computations and were found to be very suitable for the communication requirements of Parallel ARPACK (PARPACK). We have found BLACS to be easy to use and its simplification of message passing to be convenient.

A Message Passing Interface (MPI) [9] implementation of PARPACK is also available. Although an alpha release for MPI-BLACS is currently available, we have experienced some difficulty in portability for non MPICH [10] implementations and decided to provide an alternative implementation to accommodate vendor supplied MPI implementations until portability issues with MPI-BLACS have been resolved.

2 Parallel ARPACK

The parallelization paradigm found to be most effective for ARPACK on distributed memory machines was to provide the user with a Single Program Multiple Data (SPMD) template. The reverse communication interface is one of the most important aspects in the design of ARPACK and this feature

lends itself to a simplified SPMD parallelization strategy. This approach was used for previous implementations of ARPACK [2] and is simple for the the user to implement. The reverse communication interface feature of ARPACK allows the PARPACK codes to be parallelized internally without imposing a fixed parallel decomposition on the matrix or the user supplied matrix-vector product. Memory and communication management for the matrix-vector product can be optimized independent of PARPACK. This feature enables the use of various matrix storage formats as well as calculation of the matrix elements on the fly.

The calling sequence to ARPACK remains unchanged except for the addition of the BLACS context (or MPI communicator). Inclusion of the context (or communicator) is necessary for global communication as well as managing I/O. The addition of the context is new to this implementation and reflects the improvements and standardizations being made in message passing [9, 7].

2.1 Data Distribution of the Arnoldi Factorization

The numerically stable generation of the Arnoldi factorization

$$AV_k = V_k H_k + f_k e_k^T$$

where

A, $n \times n$ matrix
H_k, $k \times k$ - projection matrix (Upper Hessenberg)
V_k, $n \times k$ matrix, $k \ll n$ - Set of Arnoldi vectors
f_k, residual vector, $f_k V_k = 0$, length n

coupled with an implicit restarting mechanism [1] is the basis of the ARPACK codes. The simple parallelization scheme used for PARPACK is as follows.

- H_k replicated on every processor
- V_k is distributed across a 1-D processor grid. (Blocked by rows)
- f_k and workspace distributed accordingly

The SPMD code looks essentially like the serial code except that the local block of the set of Arnoldi vectors, V_{loc}, is passed in place of V, and n_{loc}, the dimension of the local block, is passed instead of n.

With this approach there are only two communication points within the construction of the Arnoldi factorization inside PARPACK: computation of the 2-norm of the distributed vector f_k and the orthogonalization of f_k to V_k using Classical Gram Schmidt with DGKS correction [5]. Additional communication will typically occur in the user supplied matrix-vector product operation as well. Ideally, this product will only require nearest neighbor communication among the processes. Typically the blocking of V is commensurate with the parallel decomposition of the matrix A. The user is free, however, to select an appropriate blocking of V such that an optimal balance between the parallel performance of PARPACK and the user supplied matrix-vector product is achieved.

An additional concern which must be addressed if this simple parallelization strategy is to be effective is scalability. Because H_k is replicated on every processor, operations on H due to the implicit restart mechanism within ARPACK are also replicated. This redundant work may present a "serial bottleneck" and could possibly prevent scalability if k grows with n as the problem size increases. The potential for such a bottleneck will hopefully be diminished when *locking* of converged vectors, as presented in [4], is implemented in the ARPACK codes. *Locking* , which is equivalent to deflation for the QR algorithm, is used to decouple converged Ritz values and associated vectors from the active part of the factorization. It is anticipated that locking may be used to keep the active portion of H_k small as well as to aid the convergence of eigenvalues with multiplicity greater than one. This feature is planned for the next release of ARPACK.

Another strategy which was tested was to use Parallel BLAS (PBLAS) [8] software developed for the ScaLAPACK project to achieve parallelization. The function of the PBLAS is to simplify the parallelization of serial codes implemented on top of the BLAS. The ARPACK package is very well suited for testing this method of parallelization since most of the vector and matrix operations are accomplished via BLAS and LAPACK routines.

Unfortunately this approach required additional parameters to be added to the calling sequence (the distributed matrix descriptors) as well as redefining the workspace data structure. Although there is no significant degradation in performance, the additional code modifications, along with the data

decomposition requirements, make this approach less favorable. As our parallelization is only across a one dimensional grid, the functionality provided by the PBLAS was more sophisticated than we required. The current implementation of the PBLAS (ScaLAPACK version 1.1) assumes the matrix operands to be distributed in a block-cyclic decomposition scheme.

2.2 Parallel Performance

To illustrate the potential scalability of Parallel ARPACK on distributed memory architectures some example problems have been run on the Maui HPCC SP2. The results shown in Table 1 attempt to illustrate the potential internal performance of the of the PARPACK routines independent of the users implementation of the matrix vector product.

In order to isolate the performance of the ARPACK routines from the performance of the user's matrix-vector product and also to eliminate the effects of a changing problem characteristic as the problem size increases, a test was comprised of replicating the same matrix repeatedly to obtain a block diagonal matrix. This completely contrived situation allows the workload to increase linearly with the number of processors. Since each diagonal block of the matrix is identical, the algorithm should behave as if *nproc* identical problems are being solved simultaneously (provided an appropriate starting vector is used). For this example we use a starting vector of all "1's". The only obstacles which prevent ideal speedup are the communication costs involved in the global operations and the "serial bottleneck" associated with the replicated operations on the projected matrix H. If neither of these were present then one would expect the execution time to remain constant as the problem size and the number of processors increase.

The matrix used for testing is a diagonal matrix of dimension $100,000$ with uniform random elements between 0 and 1 with four of the diagonal elements separated from the rest of the spectrum by adding an additional 1.01 to these elements. The problem size is then increased linearly with the number of processors by adjoining an additional diagonal block for each additional processor. For these timings we used the non-symmetric PARPACK code pdnaupd with the following parameter selections: mode is set to 1, number of Ritz values requested is 4, portion of the spectrum is "LM", and the maximum number of columns of V is 20.

Number of Nodes	Problem Size	Total Time (s)	Efficiency
1	100,000 * 1	40.53	
4	100,000 * 4	40.97	0.98
8	100,000 * 8	42.48	0.95
12	100,000 * 12	42.53	0.95
16	100,000 * 16	42.13	0.96
32	100,000 * 32	46.59	0.87
64	100,000 * 64	54.47	0.74
128	100,000 * 128	57.69	0.70

Table 1: Internal Scalability of Parallel ARPACK

3 Summary

We have presented a parallel implementation of the ARPACK library which is portable across a wide range of distributed memory platforms. The portability of PARPACK is achieve by utilization of the BLACS. We have found BLACS to be easy to use and have been quite satisfied with how little effort it takes to port PARPACK to a wide variety of parallel platforms. So far we have tested PARPACK on a SGI Power Challenge cluster using PVM-BLACS, on a CRAY T3D using Cray's implementation of the BLACS, on an IBM SP2 using MPL-BLACS, and on a network of Sun stations using PVM-BLACS. The MPI version of PARPACK has been tested on these platforms as well.

References

[1] D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method,* SIAM Journal on Matrix Analysis and Applications, 13(1):357-385, January 1992.

[2] D. C. Sorensen, *Implicitly-Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations,* (invited paper), in Parallel Numerical Algorithms: Proceedings of an ICASE/LaRC Workshop, May 23-25, 1994, Hampton, VA, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., Kluwer, 1995 (to appear).

[3] R.B. Lehoucq, D.C. Sorensen, P.A. Vu, and C. Yang, *ARPACK: Fortran subroutines for solving large scale eigenvalue problems,* Release 2.1

[4] R. B. Lehoucq and D.C. Sorensen *Deflation Techniques for an Implicitly Re-started Arnoldi Iteration* , To appear in SIAM Journal of Matrix Analysis

[5] J. Daniel, W.B. Gragg, L. Kaufman, and G.W. Stewart *Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization* , Mathematics of Computation, 30:772-795, 1976

[6] M.P. Debicki, P. Jedrzejewski, J. Mielewski, P. Przybyszewski, and M. Mrozowski *Application of the Arnoldi Method to the Solution of Electromagnetic Eigenproblems on the Multiprocessor Power Challenge Architecture,* Technical Report Number 19/95, Department of Electronics, Technical University of Gdansk, Poland.

[7] J. J. Dongarra and R. C. Whaley *LAPACK Working Note 94, A User's Guide to the BLACS v1.0,* , June 7, 1995

[8] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley *LAPACK Working Note 100, A Proposal for a Set of Parallel Basic Linear Algebra Subprograms* , May 1995

[9] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard* , International Journal of Supercomputer Applications and High Performance Computing, 8(3/4), 1994

[10] MPICH, *A Portable Implementation of MPI,* available from info.mcs.anl.gov/pub/mpi