SYISDA Users' Guide

SYISDA version 2.0 November 6, 1995

A Parallel Eigensolver Based on a Symmetric Invariant Subspace Decomposition Algorithm

Christian Bischof[†]
William George[†]
Steven Huss-Lederman[‡]
Xiaobai Sun[*]
Anna Tsao[‡]
Thomas Turnbull[‡]

[†]Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois

[‡]Center for Computing Sciences
Bowie, Maryland

[*]Department of Computer Science
Duke University
Durham, North Carolina

Chris Bischof
bischof@mcs.anl.gov

Steve Huss-Lederman
lederman@super.org

Elaine Jacobson
emj@super.org

Anna Tsao
anna@super.org

Tom Turnbull
turnbull@super.org

Xiaobai Sun
xiaobai@cs.duke.edu

ACKNOWLEDGMENTS

This work is partially supported by ARPA's Applied and Computational Mathematics Program originally under Contract DM28E04120, and currently ARPA order number C65.

Access to the Intel Paragon system operated by Caltech on behalf of the Center for Advanced Computing Research was provided by NSF.

Access to the IBM SP was provided by the High-Performance Computing Research Facility at Argonne National Laboratory. The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing.

CONTACT

Please feel free to send questions, comments, and problem reports to `prism@super.org`.

Contents

1 Introduction

The PRISM project is a joint effort of the Argonne National Laboratory, Center for Computing Sciences (formerly the Supercomputing Research Center), Duke University, University of California at Berkeley, and University of Kentucky. Its overall goal is the development of scalable, parallel eigensolvers for distributed memory systems.

The software discussed in this document relates to the packages developed for solving the symmetric eigenproblem which is joint work between Argonne National Laboratory, Center for Computing Sciences, and Duke University. The software contains two main packages:

- SYISDA:

  This is the symmetric eigensolver package. It is based on the Invariant Subspace Decomposition Approach (ISDA) as described in [1]. This method uses matrix multiplication as its dominant kernel to achieve high performance. The eigensolver can be run only on a square logical mesh of processors.

- BiMMeR:

  This package implements a variant of the Broadcast-Multiply-Roll (BMR) algorithm to perform matrix multiplication. It was developed as a core routine for the SYISDA package, but is available separately since matrix multiplication is an important kernel in many packages. It is described in PRISM Working Note #14. It is meant to be a distributed memory equivalent of DGEMM which is a Level 3 BLAS routine for serial computers. It works on any matrix size for an arbitrary number of processors where the processors are treated as a 2D topology. The current release of BiMMeR does not implement the $A^T \times B^T$ option of DGEMM. Users interested in this option should contact `prism@super.org`.

All computations for both the SYISDA and BiMMeR packages are in double precision only. Users interested in single precision should again contact `prism@super.org`.

2 Portability

2.1 BLAS and LAPACK

The SYISDA eigensolver routines utilize standard LAPACK and BLAS routines at the node level. The BiMMeR software only needs the BLAS routines. On several machines, such as the Intel Paragon (Delta) and the IBM SP, fast vendor supplied versions of these libraries are available. Portable versions of these libraries can be obtained from `netlib` [2]. The World Wide Web URL for netlib is

`http://www.netlib.org/index.html`.

Anonymous ftp access is available through the server at `ftp.netlib.org`.

2.2 Message Passing

All of the communication performed in the PRISM software uses the MPI message passing library [3]. Implementations of this library are available for the IBM SP system, Intel Paragon and Delta, Cray T3D, Meiko CS2, Thinking Machines CM-5, and others.

The source code in the current distribution contains two methods for performing collective communications. One method uses only MPI calls while the other method uses routines developed within the PRISM project. Compile directives determine which set of collective communication routines will be used in the library. We refer to the libraries built using the PRISM collective communication as the "pc" version ((P)RISM (c)ollective communications). We refer to the library built using only MPI communication routines as an "mc" version ((M)PI (c)ollective communications). Although the current versions of SYISDA and BiMMeR support both of these collective communication routines, we expect to update only the "mc" versions of these libraries.

2.3 Supported Systems

The PRISM software should run on any machines that support a version of the MPI message passing library for communication, and the BLAS and LAPACK linear algebra libraries for node computations. We have run a version of the PRISM software on each of the systems shown in Section 3.2.

3 Installation

3.1 Obtaining SYISDA

The SYISDA eigensolver is distributed as portable ANSI C source code in a single compressed tar file. This distribution file contains the entire directory tree of the SYISDA library. The latest distribution can be obtained from the anonymous ftp server `ftp.super.org` in the directory `pub/prism`. The latest version of the entire SYISDA library is in the distribution file `prism_syisda.tar.Z`. The WWW URL is

`ftp://ftp.super.org/pub/prism/prism_syisda.tar.Z`.

The file `prism_bimmer.tar.Z` contains only the matrix multiplication routines. Working notes and published papers on the PRISM project can also be found in this directory. See Section 9 for a description of the available PRISM working notes.

Create a directory to hold the SYISDA code, for example `~/SYISDA`, and place the distribution file in that directory. The distribution file can then be uncompressed and untarred with the UNIX command

`zcat` <DISTRIBUTION FILE> `| tar xf -`

where <DISTRIBUTION FILE> is the name of the SYISDA or BiMMeR distribution file.

If you have obtained a complete distribution of the SYISDA eigensolver, the SYISDA directory should now contain the following files and directories:

`doc/` : The documentation directory containing a PostScript version of this file as well as other documentation.

`create_syisda` : Bourne shell script to build the SYISDA libraries and test programs, including the BiMMeR matrix multiplication routines.

`create_bimmer` : Bourne shell script to build only the BiMMeR matrix multiply library.

`create_rrtrid` : Bourne shell script to build only the rrtrid (rank-revealing tridiagonalization) reduction library.

`code/` : All of the source code exists below this subdirectory in:

- `code/mm` : BiMMeR matrix multiply routines
- `code/mm_test` : test code for BiMMeR
- `code/utility` : utility routines
- `code/rrtrid` : rank-revealing tridiagonalization routines
- `code/rrtrid_test` : test codes for rrtrid (currently not available)
- `code/syisda` : main eigensolver routines
- `code/syisda_test` : test codes for eigensolver
- `code/include` : C include files

`code/lib/` : Library subdirectories. The libraries (*.a) built by `create_syisda`, `create_bimmer`, and `create_rrtrid` are stored in the directories named

`lib/`<ARCH>`/`<COMM>`/`

where <ARCH> is the machine architecture and <COMM> is the MPI communications library.

The values for <ARCH> and <COMM> are held in the environment variables `PRISM_MPI_ARCH` and `PRISM_MPI_COMM` which are used by the `Makefile`s to control the building of the libraries for the given system. The proper settings for these environment variables for the currently supported systems are given in the next section.

The libraries built by the `create_*` scripts are:

`lib`<MODULE><COMM><TEST>`.a`

where the possible suffixes are:

<MODULE> : one of `mm`, `rrtrid`, `syisda`, and `utility`, corresponding to the routines in the source code directories with the same name,

<COMM> : either 'p' or 'm' depending on whether (P)RISM or (M)PI collective communication routines are used (the PRISM version will be obsolete soon),

<TEST> : 't' if the library is a *test* version and empty if the library is a *production* version. The difference between the *test* and *production* versions of the libraries are described in sections 3.3 and 3.4 on building the libraries.

`make_include/` : This directory contains the files

`Make-`<ARCH>`-`<COMM>`.include`

where <ARCH> and <COMM> are as described previously. These files are read by the individual `code/*/Makefile`'s to set the compiler flags and other #define's as needed to compile the source code for each system.

If you have obtained a distribution consisting only of the the BiMMeR libraries, the `rrtrid` and `syisda` directories will not be present.

3.2 Setting Environment Variables

The compilation of the libraries is controlled by a set of shell environment variables which describe the target architecture including the hardware and software available. The Makefiles are similar to the ones provided by Argonne National Laboratory/Mississippi State for MPI but are expanded to include other MPI packages. The environment variables that need to be set before building the libraries are:

`PRISM_MPI_ARCH` : Machine architecture. Valid values are: `sun4`, `rs6000`, `paragon`, `intelnx`, `cray_t3d`, `meiko`.

`PRISM_MPI_COMM` : MPI communications package. Valid values are: `ch_p4`, `ch_eui`, `mpif`, `nx`, `ch_nx`, `t3d`, `ch_meiko`, `ch_cmmd`.

`PRISM_MPI_HOME` : Home directory of MPI.

`PRISM_LAPACK_LIB` : Linker flag needed to link with the LAPACK library.

`PRISM_BLAS_LIB` : Linker flag needed to link with the BLAS library.

`PRISM_MAKE_INCLUDE` : Location of the SYISDA make-include files provided with the distribution. This should be set to `../../make_include` if you have used the directory structure given in the distribution tar file.

Of course not all combinations of `PRISM_MPI_ARCH` and `PRISM_MPI_COMM` are valid. Below are the machines on which we have run and the settings for the `PRISM` environment variables (in csh syntax). Where the value is a "?", you need to fill in the location on your machine. If you are building only the BiMMeR library, the `LAPACK` library, and therefore the `PRISM_LAPACK_LIB` variable is not required. It is possible to add in new machines and architectures. You can either contact us or add the correct make_include file to try it out. We would appreciate being informed about successful ports to other platforms.

In addition to the systems listed below, we have attempted to run SYISDA on the Cray T3D using the Edinburgh version of MPI with no success. We have discovered that the `MPI_Bcast` routine in this version of MPI works only if the root of the broadcast is processor 0. However, the SYISDA library works correctly on the T3D using the MPICH version of MPI from Argonne National Laboratory and Mississippi State University.

```
## sun4 using ANL/MS mpich with p4:
setenv PRISM_MPI_ARCH sun4
setenv PRISM_MPI_COMM ch_p4
setenv PRISM_MPI_HOME ?
setenv PRISM_LAPACK_LIB ?
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/
```

```
## IBM SP using ANL/MS mpich:
setenv PRISM_MPI_ARCH rs6000
setenv PRISM_MPI_COMM ch_eui
setenv PRISM_MPI_HOME ?
setenv PRISM_LAPACK_LIB ?
## usually done with "-lblas"
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/

## IBM SP using IBM's MPIF version of MPI:
setenv PRISM_MPI_ARCH rs6000
setenv PRISM_MPI_COMM mpif
## this assumes your mpif is in the standard location
setenv PRISM_MPI_HOME /usr/lpp/mpif
setenv PRISM_LAPACK_LIB ?
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/

## Intel Paragon using Intel's nx version of MPI:
setenv PRISM_MPI_ARCH paragon
setenv PRISM_MPI_COMM nx
setenv PRISM_MPI_HOME ?
## use "/lapack_dir/lapack_paragon.a -lf"
setenv PRISM_LAPACK_LIB ?
## for BLAS, usually use "-lkmath"
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/

## Intel Delta using ANL/MS mpich:
setenv PRISM_MPI_ARCH intelnx
setenv PRISM_MPI_COMM ch_nx
setenv PRISM_MPI_HOME ?
## for LAPACK use "/lapack_dir/lapack_delta.a -lf"
setenv PRISM_LAPACK_LIB ?
## for BLAS, usually use "-lkmath"
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/

## Meiko CS2 using ANL/MS mpich:
setenv PRISM_MPI_ARCH meiko
setenv PRISM_MPI_COMM ch_meiko
setenv PRISM_MPI_HOME ?
setenv PRISM_LAPACK_LIB ?
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/
```

```
## Thinking Machine CM5 using cmmd:
setenv PRISM_MPI_ARCH sun4
setenv PRISM_MPI_COMM ch_cmmd
setenv PRISM_MPI_HOME ?
setenv PRISM_LAPACK_LIB ?
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/

## Cray T3D using ANL/MS mpich:
setenv PRISM_MPI_ARCH cray_t3d
setenv PRISM_MPI_COMM t3d
setenv PRISM_MPI_HOME ?
setenv PRISM_LAPACK_LIB ?
setenv PRISM_BLAS_LIB ?
setenv PRISM_MAKE_INCLUDE ../../make_include/
```

3.3 Building the BiMMeR Library

To build the BiMMeR library and test code, set the `PRISM_*` environment variables as described previously and then run the script

`create_bimmer` **option**

found in the home directory of SYISDA. With no command-line option, `create_bimmer` creates all versions of the libraries and test programs.

If `create_bimmer` does not execute, make sure the script is in the current directory and that your environment variable `PATH` contains the current directory, i.e. '.'. You may avoid altering your `PATH` variable by specifying the exact path to the script, i.e. enter the command `./create_bimmer` instead of `create_bimmer`.

During the building of the BiMMeR library, some machines will issue warning messages which refer to non-standard object file name extensions. These are due to our use of different object file extensions for test, `mc`, and `pc` versions of the libraries. For example, the test version of the BiMMeR library using only MPI communication (`mc`) is built from object files with the extension `.o_mt`. These warning can be safely ignored.

The command-line options recognized by `create_bimmer` are:

`new` : Recreate the libraries from scratch.

`pc` : Create production library using PRISM collective ops.

`mc` : Create production library using MPI collective ops.

`all` : Create both the `pc` and `mc` versions.

`pctest` : Create test library and test code using PRISM collective ops.

`mctest` : Create test library and test code using MPI collective ops.

`alltest` : Create both the `pctest` and `mctest` versions.

`pctestlib` : Create test library using PRISM collective ops.

`mctestlib` : Create test library using MPI collective ops.

`alltestlib` : Create both the `pctestlib` and `mctestlib` versions.

`complete` : This is the default operation which creates the `pc`, `mc`, `pctest`, and `mctest` versions.

`cleanobjs` : Remove object files. This cleans up after the build.

The `pctest`, `mctest` and `alltest` options rebuild both the BiMMeR library and the test code. The `pctestlib`, `mctestlib` and `alltestlib` options rebuild only the BiMMeR library without rebuilding the test code.

The production library removes optional prints (via CPP), checks that are expensive, and timing code. It is compiled with compiler optimization enabled (`-O` compiler switch). The test library has all the optional prints, checks and timers turned on. It is compiled for debugging (`-g` compiler switch). The BiMMeR test code needs the test library. People who want to link to our library will probably want to use the production library ("`create_bimmer mc`").

N.B. *The PRISM/MPI version (*`pc`*,* `pctest`*,* `pc_testlib`*) will soon become obsolete. It was used on machines where some MPI collective routines were failing.*

The BiMMeR test code is created in the `code/mm_test` directory. The libraries are created in the `code/lib/`<ARCH>`/`<COMM> directory. The BiMMeR code requires two libraries, libmm.a and libutility.a, which are linked in by the create scripts. See the README file in `code/mm` for general information on the BiMMeR source code.

3.4 Building the SYISDA Library

To build the eigensolver library and test code, set the `PRISM_*` environment variables as described previously and then run the script

`create_syisda` **option**

found in the home directory of SYISDA. With no command-line option, `create_syisda` creates all versions of the libraries and test programs.

If `create_syisda` does not execute, make sure the script is in the current directory and that your environment variable `PATH` contains the current directory, i.e. '.'. You may avoid altering your `PATH` variable by specifying the exact path to the script, i.e. enter the command `./create_syisda` instead of `create_syisda`.

During the building of the SYISDA library, some machines will issue warning messages which refer to non-standard object file name extensions. These are due to our use of different object file extensions for test, `mc`, and `pc` versions of the libraries. For example, the test version of the SYISDA library using only MPI communication (`mc`) is built from object files with the extension `.o_mt`. These warning can be safely ignored.

The command-line options recognized by `create_syisda` are:

`new` : Recreate the libraries from scratch.

`mc` : Create only the production `mc` version of the libraries.

`pc` : Create only the production `pc` version of the libraries.

`mctest` : Create only the test version of the libraries. Use MPI for all communications. This also compiles a test program, `prism_syisda_test_mc`, in the `code/syisda_test` directory that can be used to test the libraries.

`pctest` : Create only the test version of the libraries. Use the PRISM collective communications routines. This also compiles a test program, `prism_syisda_test_pc`, in the `code/syisda_test` directory that can be used to test the libraries.

`complete` : This is the default operation which includes the `pc`, `mc`, `pctest`, and `mctest` versions.

`cleanobjs` : Remove all of the object files. This cleans up after the build.

Both the `mc` and `pc` versions of the libraries use MPI for point-to-point communications. However, the `pc` version uses PRISM specific routines instead of MPI for all collective communications. The production library removes optional prints (via CPP), checks that are expensive, and timing code. It is compiled with compiler optimization enabled (`-O` compiler switch). The test library has all the optional prints, checks and timers turned on. It is compiled for debugging (`-g` compiler switch). The SYISDA test code needs the test library. People who want to link to our library will probably want to use the production library ("`create_syisda mc`").

N.B. *The PRISM/MPI version (*`pc`*,* `pctest`*) will soon become obsolete. It was used on machines where some MPI collective routines were failing.*

For the remainder of this document we will assume you will be building and using the `mc` versions of these libraries. References to libraries and programs with `mc` suffixes or prefixes will also apply to the corresponding `pc` versions.

If you want to rebuild the libraries for a different <ARCH> and/or <COMM> you need to reset the `PRISM_MPI_ARCH` and/or the `PRISM_MPI_COMM` environment variables and execute

```
create_syisda new mc
```

The script `create_syisda` also calls the scripts `create_bimmer` and `create_rrtrid` as needed to build the BiMMeR and rrtrid libraries. These scripts can also be executed independently if needed. The libraries built by these scripts will be placed in the directory `code/lib/`<ARCH>`/`<COMM>. It is safe to build multiple libraries in the same source tree as long as you use the "`new`" commmand-line option.

4 Using PRISM Routines

The PRISM routines were written in the C programming language. At the current time there is no portable way in MPI to pass opaque objects between C and Fortran since some implementations have chosen to use different implementations. The MPI Forum is currently addressing this problem and intends on correcting it in the near future. The difficulty for our routines is the MPI communicators supplied by the calling program. In some implementation this is safe or can be worked around. If you need to call out routines from Fortran then let us know what machine(s) and comm(s) you need and we will see if a patch can be made for now.

The SYISDA libraries require initialization before any of the main functions are called. Also, since these libraries use MPI to perform communication, the MPI library must be initialized. Therefore, before calling the BiMMeR or SYISDA libraries, the following calls must be made:

```
/* initialize MPI */
MPI_Init(&argc, &argv);

/* initialize PRISM */
prism_v_init_var();
```

The matrices are required to be distributed in a *virtual 2D torus wrap* data layout as described in [5] (also PRISM Working Note #14). For square processor meshes, which are required by the SYISDA library, the *virtual 2D torus wrap* data layout is compatable with HPF data layouts [4]. Some routines are available in `code/utility` that may be of assistance in creating arrays conforming to this layout. In particular, the routines: `prism_v_gindx_loc()`, `prism_v_gindx_lpanel()`, `prism_i_ldim()`, `prism_i_ldim_sbmtrx()`, and `prism_v_layout()` demonstrate the required mapping of array elements to processors.

4.1 Calling Convention

4.1.1 Function and Variable Naming Conventions

To avoid name conflicts, all of the globally visible functions in the BiMMeR and SYISDA libraries have the prefix `prism_`. The next letter in the function name indicates the data-type of the return value. Most variable names begin with a data-type indicator also. The data types are indicated as shown in the following table.

code	data type
`i` :	int
`c` :	char
`r` :	float (real)
`d` :	double
`e` :	enumeration
`p` :	pointer; A pointer is further specified by the type of value it is pointing to such as `p_i` for a pointer to an integer.
`s` :	structure
`v` :	void
`m` :	matrix; This indicates one of the PRISM specific data types for matrices such as the data type `P_R_MATRIX` which is a pointer to a real matrix.
`x` :	This indicates an opaque or unknown data type such as the MPI type `MPI_Status`.

4.1.2 Calling the BiMMeR Matrix Multiply Routine

In the following description, $\mathrm{op}(X)$ represents the choice between either the matrix X or its transpose, X^T. The top level routine in the BiMMeR library is the general matrix multiply routine `prism_v_bimmer()` which computes

$$C = \alpha\,\mathrm{op}(A)\,\mathrm{op}(B) + \beta C$$

where $\mathrm{op}(A)$ is an $M \times K$ matrix, $\mathrm{op}(B)$ is a $K \times N$ matrix, C is an $M \times N$ matrix, and α and β are scalar values.

Except for the omission of $C = \alpha A^T B^T + \beta C$, `prism_v_bimmer()` performs the same computation as the BLAS routine `DGEMM`.

Within each node, the layout of $\mathrm{op}(A)$, $\mathrm{op}(B)$, and C must be conformal. The exact correspondence between the individual rows and columns of the blocks and the original matrices does not need to be known for the purposes of performing the multiplication, but the *number* of rows and columns is needed.

The function prototype for prism_v_bimmer() is:

```
void prism_v_bimmer(
  int i_v_dim, int i_rblk_nw, int i_cblk_nw,
  int i_g_m, int i_g_n, int i_g_k,
  int i_m_panelwidth, int i_n_panelwidth, int i_k_panelwidth,
  int i_m_offset, int i_n_offset, int i_k_offset,
  int i_panel_spc,
  E_BOOLEAN e_transa, E_BOOLEAN e_transb,
  double r_alpha,
  P_R_MATRIX m_r_a, int i_lda,
  P_R_MATRIX m_r_b, int i_ldb,
  double r_beta,
  P_R_MATRIX m_r_c, int i_ldc,
  P_R_MATRIX m_r_bcbuf, int i_bcbuf_sz,
  P_R_MATRIX m_r_rollbuf, int i_rollbuf_sz,
  MPI_Comm comm_row, MPI_Comm comm_col,
  int (*i_lin_dim)(int i_v_dim, int i_g_dim, int i_blk_nw,
                   int i_first_blk, int i_last_blk,
                   int i_panelwidth, int i_offset,
                   int i_panel_spc))
```

where the parameters are defined as follows:

i_v_dim : Virtual dimension used for blocking of matrices.

i_rblk_nw : Row index of block in nw corner of submesh.

i_cblk_nw : Col index of block in nw corner of submesh.

i_g_m : Number of rows of op(A) and of C.

i_g_n : Number of cols of op(B) and of C.

i_g_k : Number of cols of op(A) and rows of op(B).

i_m_panelwidth : Panel width for torus wrap of rows of op(A) and of C.

i_n_panelwidth : Panel width for torus wrap of cols of op(B) and of C.

i_k_panelwidth : Panel width for torus wrap of cols of op(A) and rows of op(B).

i_m_offset : Row offset for size of first logical block of op(A) and of C (has the opposite effect on the last logical block).

i_n_offset : Col offset for size of first logical block of op(B) and of C (has the opposite effect on the last logical block).

i_k_offset : Offset for size of first logical col block of op(A) and row block of op(B) (has the opposite effect on the last logical block).

i_panel_spc : Number of virtual nodes to step to next contiguous panel.

e_transa : This parameter is of type `E_BOOLEAN` which is defined in the `mm.h` include file. This is a C enumeration type with the two defined values of `true` and `false`. If false, $\mathrm{op}(A) \rightarrow A$; if true, $\mathrm{op}(A) \rightarrow A^T$.

e_transb : If false, $\mathrm{op}(B) \rightarrow B$; if true, $\mathrm{op}(B) \rightarrow B^T$.

r_alpha : Constant to multiply $\mathrm{op}(A)$ by in matrix-matrix multiplication.

m_r_a : This parameter is of type `P_R_MATRIX` which is defined in the `mm.h` include file. This is a vector of pointers to columns of a matrix of double-precision elements. In C terms, this data type is a `double **`. The utility routine `prism_m_d_alloc_matrix(num_rows, num_cols)` can be used to create these arrays. `m_r_a` is a pointer to pointer to subblocks of matrix A; buffer for A *must* be a contiguous block in memory.

i_lda : Leading dimension of "arrays" containing blocks of A.

m_r_b : Pointer to pointer to subblocks of matrix B; buffer for B *must* be a contiguous block in memory.

i_ldb : Leading dimension of "arrays" containing blocks of B; must be large enough to accommodate block of B with max number of rows,

r_beta : Constant to multiply C by in matrix-matrix multiplication.

m_r_c : Pointer to pointer to subblocks of matrix C; buffer for C *must* be a contiguous block in memory.

i_ldc : Leading dimension of "array" containing blocks of C; must be large enough to accommodate block of C with max number of rows.

m_r_bcbuf : Pointer to pointer to buffered copy of subblocks of matrix A; if `NULL`, then a temporary buffer will be allocated. This buffer must

- have same leading dimension as A,
- be a contiguous block in memory, and,
- be large enough to accommodate block with most cols of A.

i_bcbuf_sz : Size of buffer for storing broadcast matrix.

m_r_rollbuf : Pointer to pointer to buffered copy of subblocks of rolled matrix; if `NULL`, then a temporary buffer will be allocated. This buffer must

- have same leading dimension as rolled matrix,
- have at least as many cols as rolled matrix, and,
- be a contiguous block in memory.

i_rollbuf_sz : Size of buffer for storing rolled matrix `m_r_rollbuf`.

comm_row : Communicator for row containing calling node.

comm_col : Communicator for column containing calling node.

i_lin_dim : Function which computes the number of cols (rows) in blocks i_first_block through i_last_blk. Both i_first_blk and i_last_blk are calculated within the BiMMeR routine. The utility routine prism_i_ldim is used in the SYISDA code.

Some of the prism_v_bimmer() parameters, such as i_rblk_nw and i_cblk_nw, are used to specify a submatrix (owned by a submesh of the entire processor mesh) on which to operate. This capability is included specifically to support the needs of the SYISDA library.

The routine prism_v_bimmer() chooses which axis to broadcast across and which axis to roll based on the relative sizes of the processor mesh extents and the transpose options (e_transa and e_transb).

There are six routines used by prism_v_bimmer() which perform variations of the Broadcast-Multiply-Roll (BMR) matrix multiply algorithm:

prism_v_bca_rb():	broadcast A roll B
prism_v_bca_rc():	broadcast A roll C
prism_v_bcb_rc():	broadcast B roll C
prism_v_ra_bcb():	roll A broadcast B
prism_v_ra_bcc():	roll A broadcast C
prism_v_rb_bcc():	roll B broadcast C

For a processor mesh of size $P \times Q$, the prism_v_bimmer() routine chooses the appropriate version of BMR based on the table shown below. In this table, the abbreviations for the broadcast and roll matrix are used that match the prism_v_* routine name shown above. The multiplication of $A^T B^T$ is not supported so the corresponding entries in the table are empty.

	$P \geq Q$		$P < Q$	
	A	A^T	A	A^T
B	bca_rb	bca_rb	ra_bcb	ra_ bcc
B^T	rb_bcc		bcb_rc	

See the BiMMeR test code in code/mm_test for an example of using prism_v_bimmer().

4.1.3 Calling the SYISDA Eigensolver Routine

The main interface to the PRISM eigensolver is the routine prism_v_syisda() found in the SYISDA library. This routine computes all the eigenvalues/eigenvectors of a real symmetric matrix. The function prototype for prism_v_syisda() is:

```
void prism_v_syisda(int i_g_dim, int i_panelwidth,
                    P_R_MATRIX m_r_a, int i_lda,
                    P_R_MATRIX m_r_z, int i_ldz,
                    MPI_Comm comm_2d)
```

where the parameters are defined as follows:

i_g_dim : Dimension of matrix to be solved.

i_panelwidth : Panel width for torus wrap.

m_r_a : Pointer to array of pointers to columns of matrix.

i_lda : Leading dimension of "arrays" containing blocks of matrix to be solved.

m_r_z : Pointer to array of pointers to columns of eigenvector matrix.

i_ldz : Leading dimension of eigenvector matrix.

The C type P_R_MATRIX is defined in the include mm.h in the code/include SYISDA directory. See the SYISDA test code in code/syisda_test for an example of using prism_v_syisda().

4.2 Linking and Running

The libraries created by the create_syisda and create_bimmer scripts are placed in the directory code/lib/<ARCH>/<COMM>/ where <ARCH> and <COMM> are obtained from the environment variables as described in Section 3.2. To use the production version of the SYISDA routines, link with the command-line switches:

-L<SYISDA_LIB_DIR> -lrrtrid -lutility -lmmm -lsyisdam

where <SYISDA_LIB_DIR> is the appropriate library directory for your installation, e.g.

~/SYISDA/code/lib/<ARCH>/<COMM>

if the root directory of your SYISDA distribution is ~/SYISDA. If you wish to link with the pc version of the libraries, link with -lmmp and -lsyisdap instead of -lmmm and -lsyisdam.

5 Test Programs

5.1 BiMMeR Test Program

The BiMMeR test program built with create_bimmer mctest is prism_bimmer_test_mc. This program performs a single matrix multiplication

$$C = A \times B.$$

The test case run by this program is specified with the 16 command-line arguments (in order): M, N, K, lambda, vdim, nw_x, nw_y, rows, cols, m_width, n_width, k_width, spc, transa, transb, and file. These arguments are defined as follows:

M : The number of rows of matrix A and the number of rows of the result matrix C.

N : The number of columns of matrix B and the number of columns of the result matrix C.

K : The number of columns of matrix A and rows of matrix B.

lambda : Factor to use when generating data for testing. Each matrix element is assiged a value according to the function

$$A(i,j) = (i+1) + lambda * (j+1)$$

Not very important what you use; 0.5 is fine but 0.0 is a bad choice.

vdim : The number of virtual processors to use. If you are using the 2D MPI topology of size $X \times Y$, then this value is typically the least common multiple of X and Y. For example, on a 2×2 2D topology, the value is 2. On a 3×2 2D topology, the value is 6.

`nw_x` : The x coordinate of the node where the data should begin in the 2D topology. For unshifted matrices this is 0.

`nw_y` : The y coordinate of the node where the data should begin in the 2D topology. For unshifted matrices this is 0.

`rows` : Number of nodes in the row direction. If on an $X \times Y$ 2D topology, this would normally be X.

`cols` : Number of nodes in the column direction. If on an $X \times Y$ 2D topology, this would normally be Y.

`m_width` : The width of the panels in dimension M. This is the size of the blocks. You can use 1.

`n_width` : The width of the panels in dimension N. This is the size of the blocks. You can use 1.

`k_width` : The width of the panels in dimension K. This is the size of the blocks. You can use 1.

`spc` : The spacing between panels. Often this is 1. If not, the value must divide the number of virtual processors, `vdim`.

`transa` : If false, uses A; if true, uses A^{T}.

`transb` : If false, uses B; if true, uses B^{T}.

Note: combinations of `transa` and `transb` can compute

$$\begin{aligned} C &= A \times B \\ C &= A^{\mathrm{T}} \times B \\ C &= A \times B^{\mathrm{T}}. \end{aligned}$$

However, $C = A^{\mathrm{T}} \times B^{\mathrm{T}}$ has not yet been implemented.

`file` : The name of the file in which to place the output. You can use `stderr` or `stdout`, in addition to file names.

To quickly verify your installation, a simple test can be run with the script `run_mm` located in the `code/mm_test` directory. This script, which takes no command-line arguments, uses the `PRISM_ARCH` and `PRISM_COMM` environment variables to determine the proper command-line needed to execute the test program `prism_bimmer_test_mc`. You may need to modify this script to work on your particular system. The test case in this script multiplies two 100×100 matrices on a 2×2 mesh of processors. The output of the script will be in the local directory in the file `run_mm.out`. The contents of the output file when run on a set of SUN workstations is shown in Figure 1. The exact results will vary from machine to machine. The code generates the correct answer if the `max_error` is small.

To simplify the running of tests, a set of tests can be run with a single script provided in the `code/mm_test` directory.

These standard test scripts are named <ARCH>-<COMM>`.run` where the <ARCH> and <COMM> names are the same values used to build the libraries. The scripts define a home directory in the shell variable `PRISM_PROG_DIR` which usually points to `code/mm_test`, the home directory of the `prism_bimmer_test_mc` program. The output files go in the directory `$PRISM_PROG_DIR/outmm`, which will be created for you if one does not already exist. Edit the script if you desire a different location. A number of the <ARCH>'s don't depend on the <COMM> so the different

```
prog = 'MC';
bc_mode = 'BC_REGULAR';
skew_mode = 'SKEW_REGULAR';
mach = 'SUN';
params = [100 100 100 5.000000e-01 -5.000000e-01 2 0 0 2 2 1 1 1 1];
transa = 'false';
transb = 'false';
a_dims = [50 50];
b_dims = [50 50];
c_dims = [50 50];
bcbuf_dims = [50 50];
rollbuf_dims = [50 50];
mesh = [2 2];
max_error = 1.294606e-15;
%mflops:      total         per_node
mflops =    [1.504284e+00 3.760711e-01];
%time:          max           min          avg
tot_time =  [1.329536e+00 6.527570e-01 9.337868e-01];
bc_time =   [4.127680e-01 1.560000e-02 1.715090e-01];
mm_time =   [7.098230e-01 1.466710e-01 2.974025e-01];
roll_time = [8.892860e-01 8.724200e-02 4.284632e-01];
mm_stats
case=case+1;
% All nodes (4) done!
```

Figure 1: Sample output from the script `run_mm`.

<ARCH>-<COMM>`.run` files are sometimes the same. The scripts run tests on both the "mc" and "pc" versions of the BiMMeR library.

For the Sun4 and CS2, the scripts use `mpirun` from ANL/MS. You need to have `$PRISM_MPI_HOME/util` in your path. You also need

`$PRISM_MPI_HOME/util/machines/machines.`<ARCH>

set to the appropriate machines. (This is usually done by the MPI administrator.)

You will get one output file for each test executed with the output file names built from the specific test parameters used. The test cases in the provided scripts are split into 5 different types of tests, denoted m1 through m5. Before running the complete script (which has over 370 cases), try to run just a few of the cases first, to make sure everything is OK.

The m1 tests produce output files named:

`m1_`<C>`_`<P>`_`<LDIM>`_`<LOG>`.m`

where the file names indicate the specific test parameters as follows:

<C> = `mc`, if the test used the "mc" version of BiMMeR.
= `pc`, if the test used the "pc" version of BiMMeR.

<P> = size of square mesh on which the code logically ran. Thus, for <P> = p, the test was run on p^2 processors and treated it as a $p \times p$ mesh.

<LDIM> = the local (on processor) dimension of the matrix run. If it ran on a $p \times p$ mesh, the full matrix size was $N = p \times$ <LDIM>.

<LOG> = ff if multiplication tested was $A \times B$.
= ft if multiplication tested was $A \times B^T$.
= tf if multiplication tested was $A^T \times B$.
(Recall that $A^T \times B^T$ is not an option at this time.)

The m2 and m3 tests produce output files named:

m2_<C>_<P>_<LOG>.m
m3_<C>_<P>_<LOG>.m

where <C>, <P>, and <LOG> mean the same as above. The difference between the m2 and m3 tests is the size of the test matrix. In each case a fixed size matrix is used while varying the size of the processor mesh. The m2 tests use a fixed size matrix of size $N = 2200$. The m3 tests use a fixed size matrix of size $N = 6600$.

Test types m4 and m5 test the code on matrices distributed over non-square processor meshes and with various panel widths. The m4 and m5 tests produce output files named:

m4_<C>_<I>_<W>_<LOG>.m
m5_<C>_<I>_<W>_<LOG>.m

where <C> and <LOG> are the same as for the previous tests and

<I> = counter. As the counter varies, the size of the logical mesh also changes. The code runs on a $16 \times p$ logical mesh for the m4 tests and a $p \times 16$ logical mesh for the m5 tests. For each test type, $p = 4, 6, 8, 16$ when <I> $= 1, 2, 3, 4$.

<W> = the panelwidth (size of local matrix blocks in the torus wrap), set equal to 1, 4, 16.

5.2 SYISDA Test Program

A test program is supplied with the SYISDA distribution in the `code/syisda_test` directory. This test program, `prism_syisda_test_mc`, requires one command-line argument: the name of file containing the specifications for the test case. A sample input file, `in1`, is included with the distribution. The contents of this file are show in Figure 2.

The values specified in this file are:

`input version` : Identifies the version of `prism_st.c` in use.

`l_v_dim` : Extent of the square virtual mesh of processors.

`l_g_dim` : Rank of the matrix.

`l_panelwidth` : Panel width for distributing the matrices.

`l_panel_spc` : Panel spacing for distributing the matrices.

`e_test_case` : Identifies the type of test matrix. Possible values are from 0 to 7. (See `prism_st.c` for a description of the test types.)

```
input version:
0
l_v_dim l_g_dim l_panelwidth    l_panel_spc     e_test_case     l_minrk
2       101     1               1               1               20
seeds for dlagsy:
127 4019 3210 1117
ratio for exponentially distributed eigenvalues:
.5
Wilkinson: size of block glue value
           101                  1.0e-14
print flags
2 15 -10000
```

Figure 2: Sample input file for the `prism_syisda_test_mc` program.

`l_minrk` : Subdivision of the problem stops at matrices smaller than this rank.

`seeds for dlagsy` : Seeds for random number generator, if needed.

`ratio for exponentially distributed eigenvalues` : Used for test case 6 (exponentially distributed eigenvalues) only.

`size of block` : Used in test case 7 (Wilkinson and glued Wilkinson matrices) only.

`glue value` : Used in test case 7 (Wilkinson and glued Wilkinson matrices) only.

`print flags` : Enable 0 or more sets of print statements compiled into the test versions of the libraries. The list of print flags must end with -10000.

These printouts are generally used for debugging the source code.

The `in1` parameter file we have supplied specifies an order 101 matrix to be run through the test code on a 2x2 mesh of processors. To run this test on a Sun using the MPICH mpirun command, enter the command-line

```
mpirun -np 4 prism_syisda_test_mc in1
```

while on the Paragon you would enter

```
prism_syisda_test_mc -sz 4 in1
```

The `prism_syisda_test_mc` program writes its output to `stdout`. The file `out1` supplied with the SYISDA distribution shows the results of this specific test when run on an Intel Paragon. The order of the printouts in the `out1` file may vary between systems. For example, when using the MPICH implementation of MPI on a cluster of Sun workstations the output may be sorted by node rather than in chronological order. Also, you will probably get slightly different answers on other machines. It is very important that the residuals are small. In `out1` they are:

```
orthogonality residual  = 2.416469802035692e-15
max residual            = 1.060734413656425e-15
```

6 Performance

For all of the results shown here, the Intel Paragon machines used were located at the Center for Advanced Computational Research (CACR) at Caltech. The "SP1.5" timings were obtained on the machine at the High-Performance Computing Research Facility (HPCRF) at Argonne National Laboratory. At the time these runs were performed, the system had been upgraded to use the communication hardware of the SP2, but the individual processors were still those of the SP1.

6.1 The Performance of BiMMeR

Timings of `prism_v_bimmer()` for various size problems and various size processor meshes have been obtained for the Intel Paragon. All tests used square processor meshes and square matrices of double precision data.

Figure 3 shows the effect on execution time of varying the size of the processor mesh while holding the problem size constant. Plots for matrices of size 2200×2200 are shown for processor mesh sizes $p \times p$ where $p = 4$, 7, 8, 12, and 16. Plots for matrices of size 6600×6600 are shown for processor mesh sizes where $p = 12$, 13, 14, and 15.

Figure 4 shows the effect on execution time of varying the size of the processor mesh while scaling the problem size with the size of the processor mesh. Plots are given for problems with per-node submatrix sizes of 100×100, 200×200, 400×400, and 550×550. The processor meshes tested are the same as in Figure 3.

Figures 5 and 6 show the MFLOPS/node rates for the same set of tests shown in Figures 3 and 4.

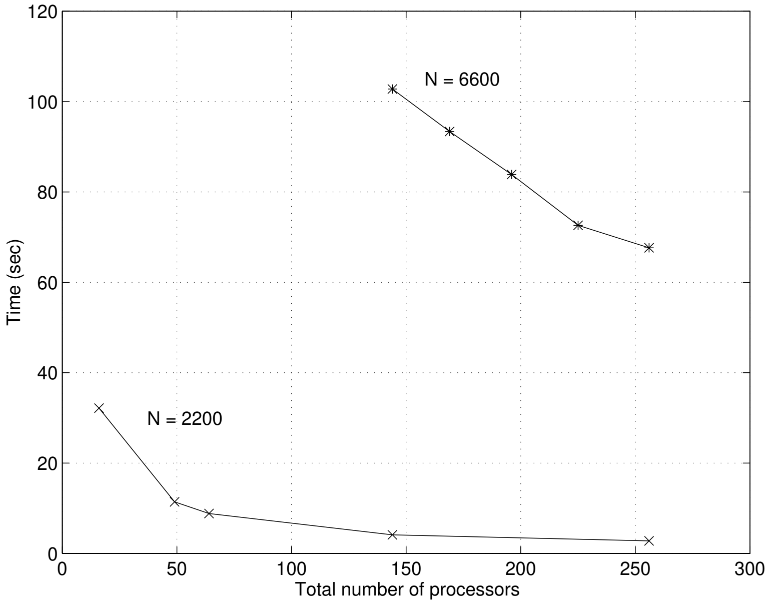

Figure 3: Time vs number of nodes for BiMMeR on constant matrix sizes on the Intel Paragon.

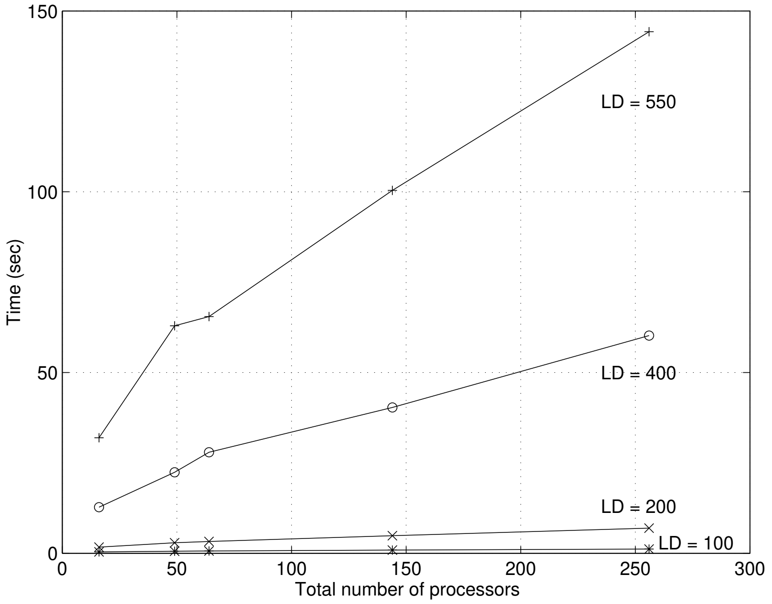

Figure 4: Time vs number of nodes for BiMMeR constant subgrid sizes on the Intel Paragon.

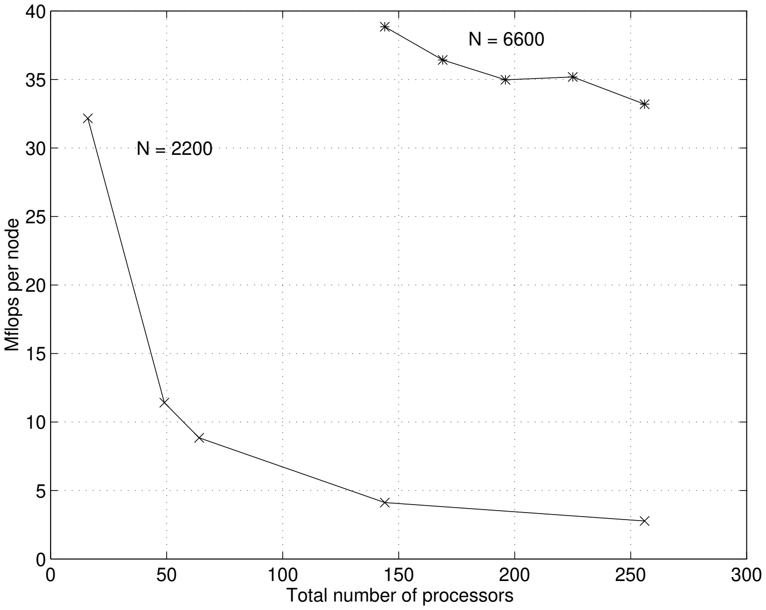

Figure 5: Performance in MFLOPS/node vs number of nodes for constant matrix sizes on the Intel Paragon.

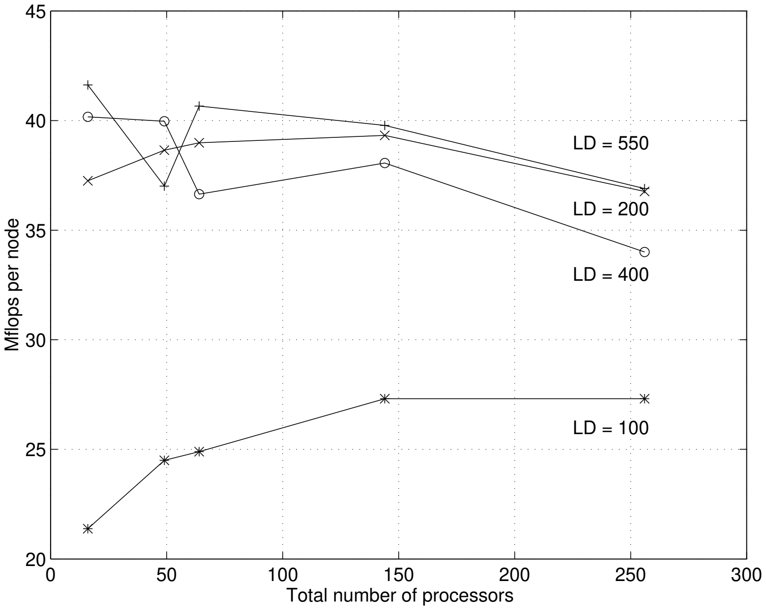

Figure 6: Performance in MFLOPS/node vs number of nodes for constant subgrid sizes on the Intel Paragon.

6.2 The Performance of SYISDA

Timings of `prism_v_syisda()` for various size problems and various size processor meshes have been obtained for both the the Intel Paragon and the IBM "SP1.5". These timing tests were conducted in August and September of 1995 using release 2.0 of the `prism_syisda` library. All code was compiled with the `-O` C compiler option and without the memory checking and debugging print statements included (no significant timings differences were noticed between the *test* and the *production* versions of the library).

Figures 7 and 8 show the effect of increasing the size of the processor mesh while holding the size of the local submatrices constant.

Figures 9 and 10 show the effect of increasing the size of the processor mesh while holding the size of the problem constant.

Figures 11 and 12 show the relative amount of time spent on each of the four major phases of the computation. In these plots, `mm` is the matrix multiply time, `rrtrid` is the time spend in the rank-revealing tridiagonalization, `endgame` is the time spent constructing the final answer from the subproblems, and `overhead` is any other miscellaneous work required. These plots show the dominating cost to be the matrix multiply with the fraction of the time required for matrix multiply increasing as the size of the local submatrices increases.

Figure 13 shows the GFLOPS rate for `prism_v_syisda()` on the Paragon and the SP. The processor mesh sizes tested were $v \times v$ for $v = 2$ through 10 on the SP, and additionally $v = 12$, 14, and 16 on the Paragon. The test matrices were generated to contain a uniform random distribution of eigenvalues. The minimum rank used for these tests was set at $\max(\frac{1}{4}M, 500)$ where M was the global extent of the test matrix A. The minimum rank was limited to a maximum of 500 to ensure adequate memory to complete the problem. One should be careful in interpreting these graphs since SYISDA usually does more operations than other eigensolvers, but is more efficient. When comparing to other eigensolvers it is usually best to use total time.

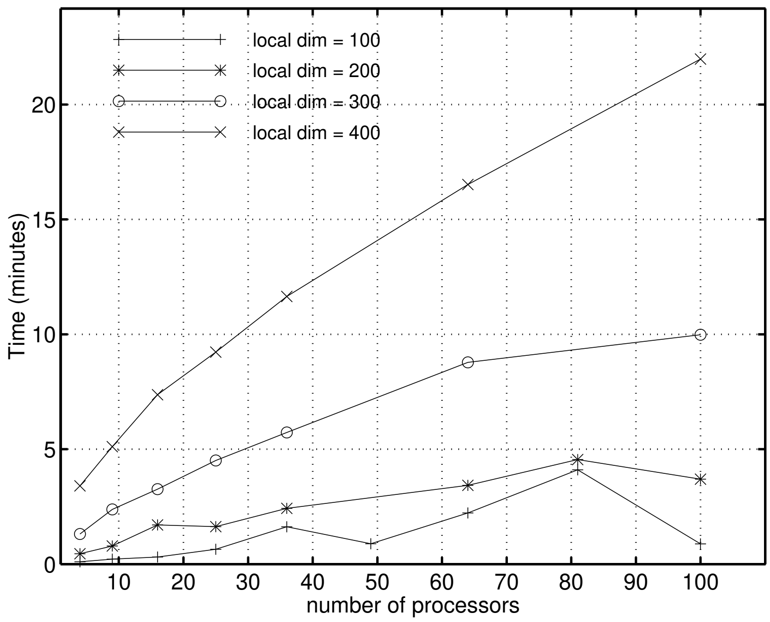

Figure 7: Time vs number of nodes for constant subgrid sizes on the ANL IBM SP1.5.

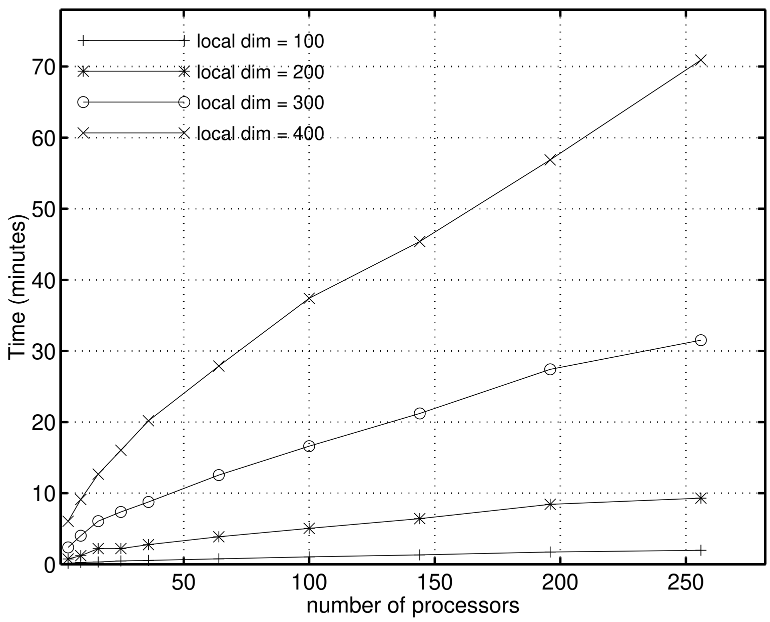

Figure 8: Time vs number of nodes for constant subgrid sizes on the Intel Paragon.

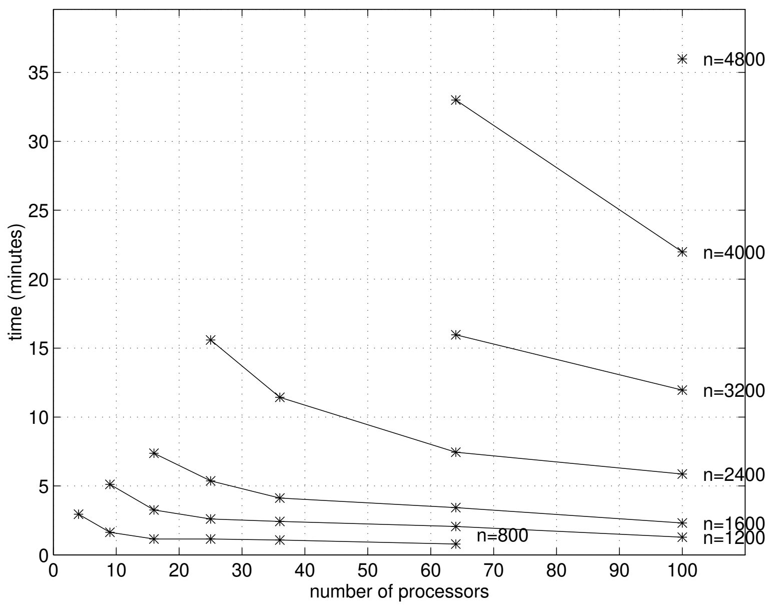

Figure 9: Time vs number of nodes for constant matrix sizes on the ANL IBM SP1.5.

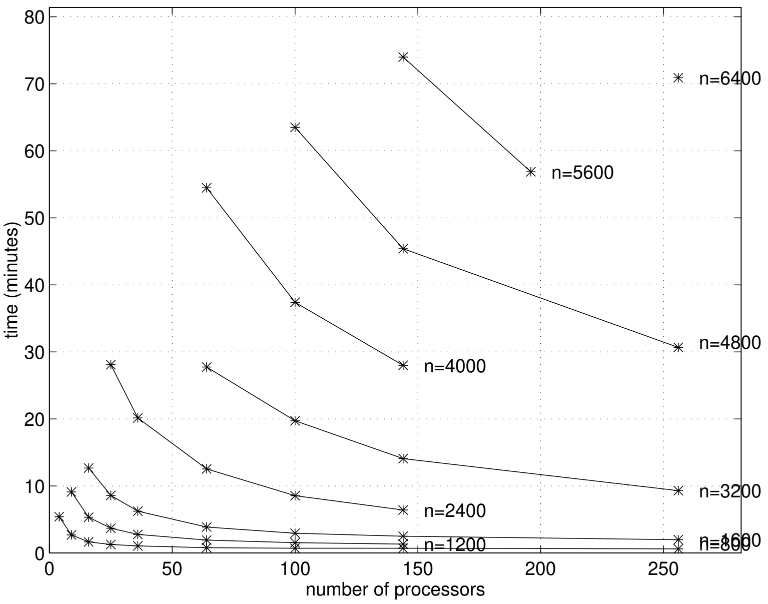

Figure 10: Time vs number of nodes for constant matrix sizes on the Intel Paragon.

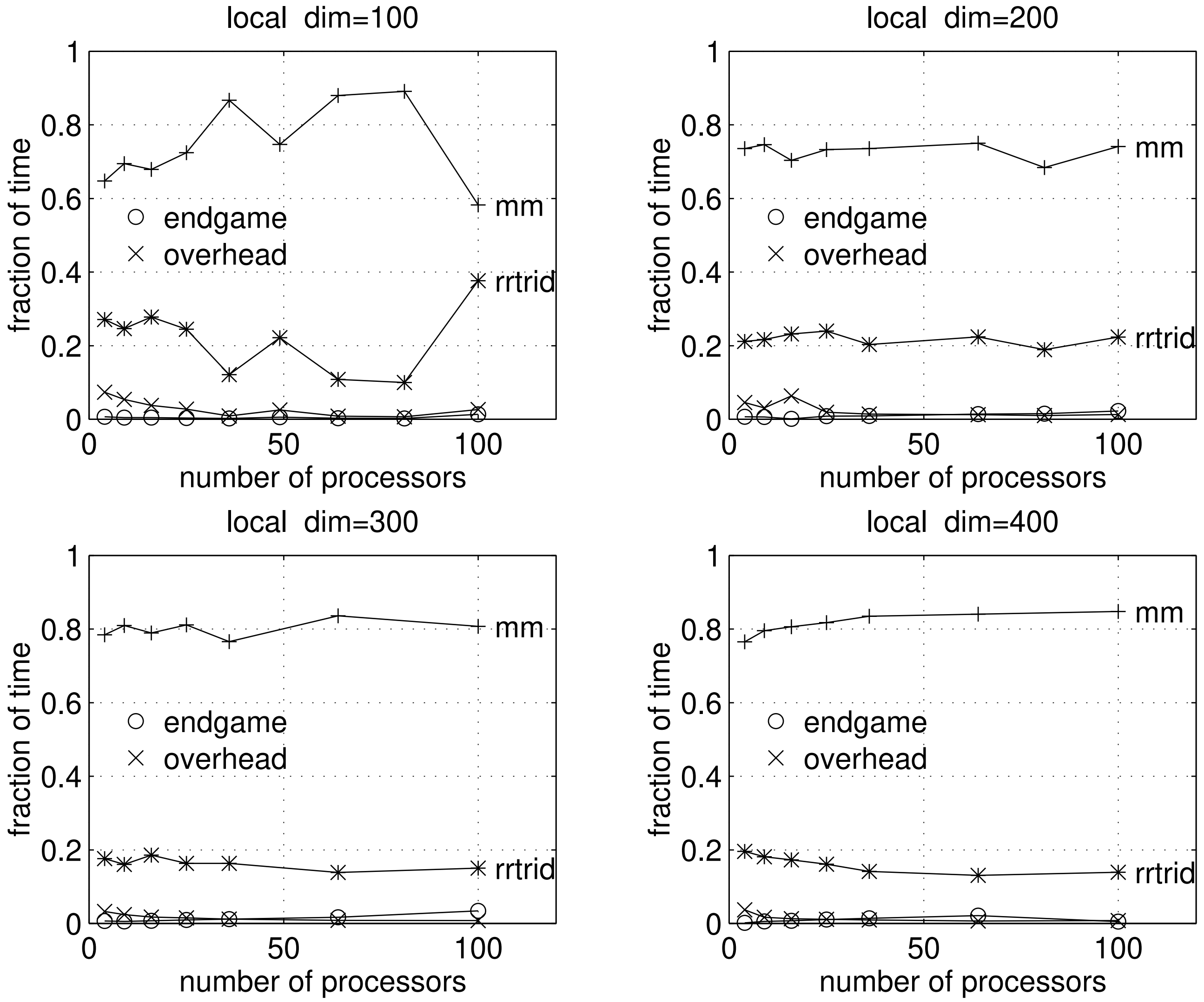

Figure 11: Time breakdown for SYISDA on the ANL IBM SP1.5

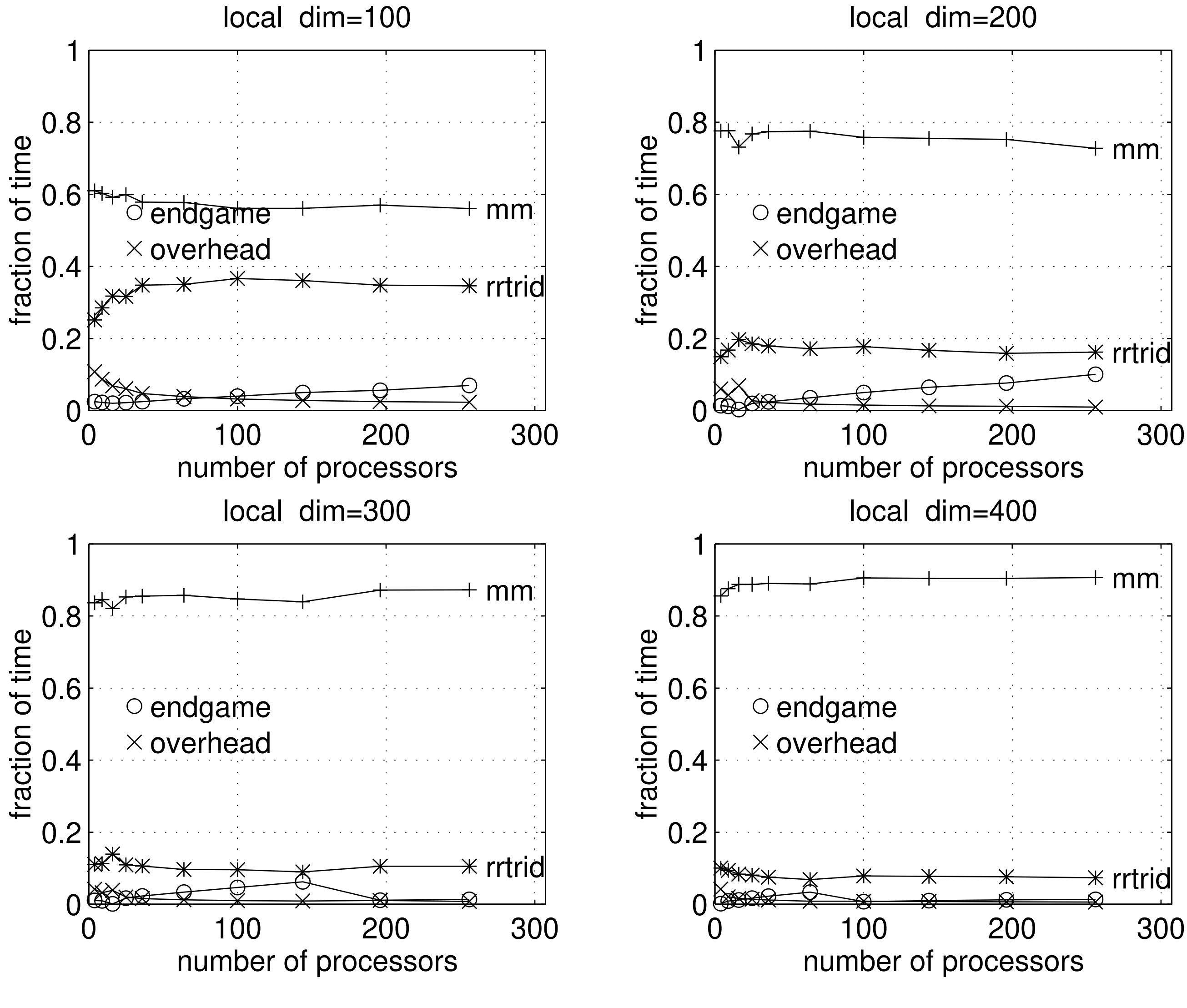

Figure 12: Time breakdown for SYISDA on the Intel Paragon.

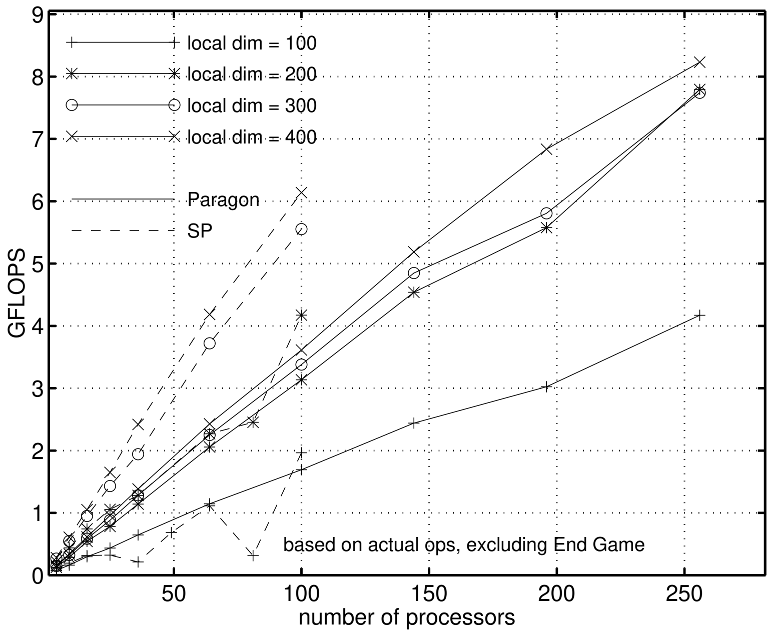

Figure 13: GFLOPS rate for SYISDA on the Intel Paragon and the ANL IBM SP1.5

7 Bugs

At the time of this writing one outstanding bug exists. We have found that the SYISDA eigensolver can be sensitive to the choice of the `l_minrk` (minimum rank) for large matrices; specifically very small values for `l_minrk` can cause problems. In particular, when run on a 16×16 processor mesh on the Intel Paragon, with a rank 4800 input matrix and the `l_minrk` set at 50, the final residuals are too large. In this case we have had residuals on the order of 10^{-6}. However, the same test case has residuals on the order of 10^{-15} when the `l_minrk` is set to 400. Since small `l_minrk` values are not optimal, this bug will hopefully not effect users.

8 Contacts

Comments, questions, bugs and requests should be sent to `prism@super.org`. We welcome all input.

9 PRISM Working Notes

A set of working notes in PostScript format can be obtained from the anonymous ftp server at `ftp.super.org` in the directory `pub/prism`.

The current set of documents in `pub/prism` are:

Papers related to PRISM

`src-tr-91-028.ps`: Auslander, L. & A. Tsao, "On parallelizable eigensolvers," Adv. Appl. Math., vol. 13, pp. 253-261, 1992 (preprint appeared as Technical Report SRC-TR-91-028, Supercomputing Research Center).

`src-tr-91-042.ps`: Lederman, S., A. Tsao, & T. Turnbull, "A parallelizable eigensolver for real diagonalizable matrices with real eigenvalues," Technical Report TR-91-042, Supercomputing Research Center (1991, 22 pages).

PRISM Working Notes

`wn1.ps`: Bischof, C. & X. Sun, "A divide-and-conquer method for computing complementary invariant subspaces of symmetric matrices," Argonne National Lab Technical Report MCS-P286-0192 (Mar. '92, 13 pages)

`wn2.ps`: Bai, Z., "Error analysis of the Lanczos algorithm for the nonsymmetric eigenvalue problem," to appear in Math. Comp. (April '92, 21 pages)

`wn3.ps`: Bischof, C. & X. Sun, "A framework for symmetric band reduction and tridiagonalization," Argonne National Lab Preprint MCS P298-0392 (July '92, 16 pages)

`wn4.ps`: Bai, Z., "Progress in the numerical solution of the nonsymmetric eigenvalue problem," to appear in JNLAA (Oct. '92, 17 pages)

`wn5.ps`: Bai, Z. & J. Demmel, "Design of a Parallel Nonsymmetric Eigenroutine Toolbox (Part I)," Univ. of Kentucky Research report 92-09 (Dec. '92, 30 pages)

`wn6.ps`: Tsao, A. & T. Turnbull, "A Comparison of Algorithms for Banded Matrix Multiplication," Technical Report SRC-TR-093-092, Supercomputing Research Center (1993, 9 pages) `wn7.ps`: Huss-Lederman, S., E. Jacobson, A. Tsao, & G. Zhang, "Matrix Multiplication on the Intel Touchstone DELTA," appeared in proceedings of Sixth SIAM conference on Parallel Processing for Scientific Computing in Norfolk, VA (March '93, 9 pages)

`wn8.ps`: Bischof, C., M. Marques, & X. Sun, "Parallel bandreduction and tridiagonalization," appeared in proceedings of Sixth SIAM conference on Parallel Processing for Scientific Computing in Norfork, VA (March '93, 8 pages)

`wn9.ps`: Huss-Lederman, S., A. Tsao, & G. Zhang, "A Parallel Implementation of the Invariant Subspace Decomposition Algorithm for Dense Symmetric Matrices," appeared in proceedings of Sixth SIAM conference on Parallel Processing for Scientific Computing in Norfork, VA (March '93, 8 pages)

`wn10.ps`: Bischof, C., Huss-Lederman, S., E. Jacobson, X. Sun, & A. Tsao, "On the Impact of HPF Data Layout on the Design of Efficient and Maintainable Parallel Linear Algebra Libraries," a description of virtual 2D torus wrap, submitted as comments to the HPF working group (March '93, 8 pages)

`wn11.ps`: Huss-Lederman, S., E. Jacobson, A. Tsao, & G. Zhang, "Matrix Multiplication on the Intel Touchstone DELTA," Technical Report SRC-TR-93-101, Supercomputing Research Center, (May 1993, 18 pages). This is an expanded version of PRISM Working Note #7 and has been superseded by PRISM Working Note #14.

`wn12.ps`: Bischof, C., S. Huss-Lederman, X. Sun, &A. Tsao, "The PRISM Project: Infrastructure and Algorithms for Parallel Eigensolvers," appeared in proceedings of the Scalable Parallel Libraries conference in Starksville, MS (October 1993, 9 pages)

`wn13.ps`: Huss-Lederman, S., E. Jacobson, A. Tsao, "Comparison of Scalable Parallel Matrix Multiplication Libraries," appeared in proceedings of the Scalable Parallel Libraries conference in Starksville, MS (October 1993, 8 pages)

`wn14.ps`: Huss-Lederman, S., E. Jacobson, A. Tsao, & G. Zhang, "Matrix Multiplication on the Intel Touchstone DELTA," Technical Report SRC-TR-93-101 (revised), Supercomputing Research Center, (Feb. 1994, 21 pages). This is an expanded version of PRISM Working Note #7 and supersedes PRISM Working Note #11. Appeared in Concurrency: Practice and Experience, Vol. 6 (7), Oct. 1994, pp. 571-594.

`wn15.ps`: Bischof, C., S. Huss-Lederman, X. Sun, A. Tsao, T. Turnbull, "Parallel Performance of a Symmetric Eigensolver based on the Invariant Subspace Decomposition Approach," appeared in proceedings of Scalable High Performance Computing Conference 1994 (March 1994, 8 pages).

`wn16.ps`: Bischof, C., X. Sun, A. Tsao, & T. Turnbull, "A Study of the Invariant Subspace Decomposition Algorithm for Banded Symmetric Matrices," appeared in Proceedings of the 5th SIAM Conference on Applied Linear Algebra (March 1994, 6 pages).

`wn17.ps`: Bischof, C., B. Lang, & X. Sun, "Parallel Tridiagonalization through Two-Step Band Reduction," appeared in proceedings of Scalable High Performance Computing Conference 1994 (March 1994, 5 pages).

`wn18.ps`: Bai, Z., J. Demmel, & M. Gu, "Inverse Free Parallel Spectral Divide and Conquer Algorithms for Nonsymmetric Eigenproblems," Research Report 94-01, Department of Mathematics, University of Kentucky (April 1994, 33 pages).

wn19.ps: Sun, X. & C. Bischof, "A Basis-Kernel Representation of Orthogonal Matrices," Argonne National Lab Preprint MCS-P431-0594 (1994, 20 pages).

wn20.ps: Lederman, S., A. Tsao, & T. Turnbull, "A parallelizable eigensolver for real diagonalizable matrices with real eigenvalues," revision of Technical Report TR-91-042, Supercomputing Research Center (1994, 24 pages).

wn21.ps: Bischof, C. & X. Sun, "On orthogonal block elimination," Argonne Preprint MCS-P450-0794 (1994, 27 pages).

wn22.ps: Bischof, C., S. Huss-Lederman, X. Sun, A. Tsao, & T. Turnbull, "A case study of MPI: Portable and efficient libraries," to appear in Proceedings, Seventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 15–17 February, 1995, 6 pages.

wn23.ps: Bischof, C., S. Huss-Lederman, X. Sun, A. Tsao, & T. Turnbull, "Parallel studies of the Invariant Subspace Decomposition Approach for banded symmetric matrices," to appear in Proceedings, Seventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 15–17 February, 1995, 6 pages.

wn24.ps: Bischof, C. H. & X. Sun, "On the design of a tridiagonalization routine for banded matrices," to appear in Proceedings, Seventh SIAM Conference on Parallel Processing for Scientific Computing, San Francisco, CA, 15–17 February, 1995, 6 pages.

References

[1] L. Auslander and A. Tsao. On parallelizable eigensolvers. *Adv. Appl. Math.*, 13:253–261, 1992.

[2] J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM*, 30(5):403–407, July 1987.

[3] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard.* University of Tennessee, June 1995. Available on **netlib**.
URL: http://www.mcs.anl.gov/mpi/mpi-report-1.1/mpi-report.html.

[4] High Performance Fortran Forum. *High Performance Fortran Language Specification, Version 1.1*, 1994. URL: http://www.erc.msstate.edu/hpff/hpf-report/hpf-report/hpf-report.html.

[5] Steven Huss-Lederman, Elaine M. Jacobson, Anna Tsao, and Guodong Zhang. Matrix multiplication on the Intel Touchstone Delta. *Concurrency: Practice and Experience*, 6(7):571–594, October 1994. also released as PRISM working note #14 (`wn14.ps`) at `ftp.super.org:pub/prism`.