ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-234

# Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation

by

*Rajeev Thakur, Ewing Lusk, and William Gropp*

Mathematics and Computer Science Division

Technical Memorandum No. 234

October 1997

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; and by the Scalable I/O Initiative, a multiagency project funded by the Defense Advanced Research Projects Agency (Contract DABT63-94-C-0049), the Department of Energy, the National Aeronautics and Space Administration, and the National Science Foundation.

# Contents

**Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation**

by

*Rajeev Thakur, Ewing Lusk, and William Gropp*

**Abstract**

ROMIO is a high-performance, portable implementation of MPI-IO (the I/O chapter in MPI-2). This document describes how to install and use ROMIO version 1.0.0 on various machines.

# 1 Introduction

ROMIO[1] is a high-performance, portable implementation of MPI-IO (the I/O chapter in MPI-2 [1]). This document describes how to install and use ROMIO version 1.0.0 on various machines.

This version of ROMIO includes everything defined in the MPI-2 I/O chapter except file info (Sec. 9.2.8), shared file pointer functions (Sec. 9.4.4), split collective data access routines (Sec. 9.4.5), support for file interoperability (Sec. 9.5), I/O error handling (Sec. 9.7), and I/O error classes (Sec. 9.8). Since info is not supported, `MPI_INFO_NULL` should be used as the info parameter where needed. Since shared file pointer functions are not supported, the `MPI_MODE_SEQUENTIAL` amode to `MPI_File_open` is also not supported. The subarray and distributed array datatype constructor functions from MPI-2 Chapter 4 (Secs. 4.14.4 and 4.14.5) have been implemented. They are useful for accessing arrays stored in files. The functions `MPI_File_f2c` and `MPI_File_c2f` (Sec. 4.12.4) also have been implemented. C, Fortran, and profiling interfaces are provided for all functions that have been implemented.

ROMIO 1.0.0 runs on the following machines: IBM SP; Intel Paragon; HP/Convex Exemplar; SGI Origin 2000, Challenge, and Power Challenge; and networks of workstations (Sun4, Solaris, IBM, DEC, SGI, HP, FreeBSD, and Linux). Supported file systems are IBM PIOFS, Intel PFS, NFS, and any Unix file system (e.g., SGI's XFS and the HP/Convex Exemplar file system).

ROMIO works with MPICH 1.1.0 (or higher) on any machine. (You can get the latest version of MPICH from `http://www.mcs.anl.gov/mpi/mpich`.) On SGI machines, ROMIO works with SGI's MPI 3.0 (or higher), and we recommend that you use it with SGI's MPI instead of MPICH. On the HP/Convex Exemplar, ROMIO works with HP MPI 1.3 (or higher), and we recommend that you use it with HP MPI instead of MPICH.

ROMIO requires that the file name passed to `MPI_File_open` be prefixed with a string to indicate the type of file system. The strings corresponding to PIOFS, PFS, NFS, and UFS file systems are `piofs:`, `pfs:`, `nfs:`, and `ufs:`. An example file name is `nfs:/home/thakur/foo`. You can open files on multiple file systems in the same program by specifying the type of file system in the file name. The user is responsible for ensuring that the directory where the file is to be opened is accessible from the process opening the file. For example, a process running on one workstation may not be able to access a directory on the local disk of another workstation, and therefore ROMIO will not be able to open a file in such a directory. Note that if you are creating files on an NFS-mounted

[1] `http://www.mcs.anl.gov/home/thakur/romio`

file system, you must specify `nfs:` in the file name; `ufs:` may not work, particularly if multiple processes write to a common file.

An MPI-IO file created by ROMIO is no different from any other file created by the underlying file system. Therefore, you may use any of the commands provided by the file system to access the file, for example, `ls`, `mv`, `cp`, `rm`, `ftp`.

Please read the limitations of this version of ROMIO that are listed in Section 5 of this document (e.g., `MPIO_Request` object, file size less than 2 Gbytes, restriction to homogeneous environments).

## 2 General Installation Instructions

Untar the tar file as

```
gunzip -c romio.tar.gz | tar xvf -
```

OR

```
zcat romio.tar.Z | tar xvf -
```

THEN

```
cd romio

./configure -file_system=nfs -mpiincdir=/usr/local/mpi/include \
            -mpilib=/usr/local/mpi/lib/sun4/ch_p4/libmpi.a
  (ONLY AN EXAMPLE. SPECIFIC configures FOR VARIOUS MACHINES ARE GIVEN BELOW.)

make

cd test

make
```

Run the examples as you would run any MPI program. Each program takes the filename as a command-line argument "`-fname filename`". The filename must be prefixed with a string to indicate the type of file system (`nfs:`, `ufs:`, `pfs:`, `piofs:`). An example filename is `nfs:test`.

## 3 Configuring and Building ROMIO on Various Machines

Here we discuss how ROMIO is commonly configured and built on various machines. For your particular machine environment, you may need to specify some other options to `configure`. For the entire list of options, do

```
./configure -h | more
```

You can configure and build ROMIO for multiple file systems by specifying the names and using '+' as a separator, for example, `./configure -file_system=ufs+nfs` ...

### 3.1 IBM SP

On an IBM SP using MPICH 1.1.0 (or higher) and PIOFS file system (specify appropriate `mpiincdir` and `mpilib` for your system):

```
./configure -file_system=piofs -mpiincdir=/usr/local/mpi/include \
            -mpilib=/usr/local/mpi/lib/rs6000/ch_mpl/libmpich.a
make
```

### 3.2 Intel Paragon

On an Intel Paragon using MPICH 1.1.0 (or higher) and PFS file system (specify appropriate `mpiincdir` and `mpilib` for your system):

```
./configure -arch=paragon -file_system=pfs -mpiincdir=/usr/local/mpi/include \
            -mpilib=/usr/local/mpi/lib/paragon/ch_nx/libmpi.a
make
```

### 3.3 HP/Convex Exemplar

On an HP/Convex Exemplar using HP MPI 1.3 (or higher) and the Exemplar file system:

```
./configure -file_system=ufs -mpi=hp
make
```

On an HP/Convex Exemplar using MPICH 1.1.0 (or higher) and the Exemplar file system (specify appropriate `mpiincdir` and `mpilib` for your system):

```
./configure -file_system=ufs -mpi=mpich -mpiincdir=/usr/local/mpi/include \
            -mpilib=/usr/local/mpi/lib/hpux/ch_shmem/libmpi.a
make
```

### 3.4 SGI Origin 2000, Power Challenge, and Challenge

On an SGI Origin 2000, Power Challenge, or Challenge using SGI's MPI 3.0 (or higher) and XFS file system:

```
./configure -file_system=ufs
make
```

If you need to generate a particular version that corresponds to the -64, -n32, or -32 compiler/linker options, you can specify the options to be passed to the compiler as

```
./configure -file_system=ufs -cc="cc -64" -fc="f77 -64"
```

or

```
./configure -file_system=ufs -cc="cc -n32" -fc="f77 -n32"
```

or

```
./configure -file_system=ufs -cc="cc -32" -fc="f77 -32"
```

To configure for networks of SGI workstations, see Section 3.5.

### 3.5 Network of Workstations

On a network of Sun4, Solaris, IBM, DEC, FreeBSD, or Linux workstations using MPICH 1.1.0 (or higher) and NFS file system (specify appropriate `mpiincdir` and `mpilib` for your system):

```
./configure -file_system=nfs -mpiincdir=/usr/local/mpi/include \
            -mpilib=/usr/local/mpi/lib/sun4/ch_p4/libmpi.a
make
```

On a network of SGI or HP workstations using MPICH 1.1.0 (or higher) and NFS file system (specify appropriate `mpiincdir` and `mpilib` for your system):

```
./configure -file_system=nfs -mpi=mpich -mpiincdir=/usr/local/mpi/include \
            -mpilib=/usr/local/mpi/lib/IRIX64/ch_p4/libmpi.a
make
```

### 3.6 Miscellaneous Instructions

If any error occurs during the build, try

```
./configure -h
```

for further options to configure, or send e-mail to `romio-maint@mcs.anl.gov` with a detailed description of the error.

After building a specific version, you can install it in a particular directory with

```
make install PREFIX=/usr/local/romio (or whatever directory you like)
```

or just

```
make install (if you used -prefix at configure time)
```

If you intend to leave ROMIO where you built it, you should *not* install it; `make install` is used only to move the necessary parts of a built ROMIO to another location. The installed copy will have the include files, libraries, man pages, and a few other odds and ends, but not the whole source tree. It will have a `test` directory for testing the installation and a location-independent Makefile built during installation, which users can copy and modify to compile and link against the installed copy.

To rebuild ROMIO with a different set of configure options, do

```
make cleanall
```

to clean up everything, including the Makefiles created by `configure`. Then run `configure` again with the new options.

## 4 Compiling and Running MPI-IO Programs

Following are instructions for compiling an MPI-IO program on various machines using ROMIO. The Makefile in the `romio/test` directory also illustrates how to compile and link to ROMIO on the particular machine you are using.

You need to include the file `mpio.h` for C or `mpiof.h` for Fortran in your MPI-IO program. If your Fortran compiler does not accept the `-I` option that specifies the include directory, you will need to copy (or soft link) the files `mpif.h` (from the MPI implementation) and `$(ROMIO_HOME)/include/mpiof.h` to the directory where you are compiling your program.

```
Assume
   CC = C compiler
   F77 = Fortran compiler
   MPI_LIB = full path to MPI library
   MPI_INCDIR = directory where mpi.h and mpif.h files are located
   ROMIO_HOME = top-level directory where ROMIO is installed
   ARCH = type of machine
   INCLUDE_DIR = -I$(ROMIO_HOME)/include -I$(MPI_INCDIR)
   LIBS = $(ROMIO_HOME)/lib/$(ARCH)/libmpio.a $(MPI_LIB)

You can compile MPI-IO programs on various machines as follows:

On an IBM SP for PIOFS file system:
   $(CC) -O $(INCLUDE_DIR) -bI:/usr/include/piofs/piofs.exp test.c $(LIBS)
   $(F77) -O $(INCLUDE_DIR) -bI:/usr/include/piofs/piofs.exp test.f $(LIBS)

On an Intel Paragon:
   $(CC) -O $(INCLUDE_DIR) test.c $(LIBS) -nx
   $(F77) -O $(INCLUDE_DIR) test.f $(LIBS) -nx

On an HP/Convex Exemplar using HP MPI:
   mpicc -O -I$(ROMIO_HOME)/include test.c $(ROMIO_HOME)/lib/$(ARCH)/libmpio.a
   mpif77 -O -I$(ROMIO_HOME)/include test.f $(ROMIO_HOME)/lib/$(ARCH)/libmpio.a

On HP workstations or Exemplar machines using MPICH:
   $(CC) -O $(INCLUDE_DIR) -Aa -D_POSIX_SOURCE test.c $(LIBS) -lV3
   $(F77) -O $(INCLUDE_DIR) test.f $(LIBS) -lV3

On SGI machines using SGI's MPI:
   $(CC) -O -I$(ROMIO_HOME)/include test.c $(ROMIO_HOME)/lib/$(ARCH)/libmpio.a -lmpi
   $(F77) -O -I$(ROMIO_HOME)/include test.f $(ROMIO_HOME)/lib/$(ARCH)/libmpio.a -lmpi

On SGI machines using MPICH:
   $(CC) -O $(INCLUDE_DIR) test.c $(LIBS)
   $(F77) -O $(INCLUDE_DIR) test.f $(LIBS)

On Sun 4, IBM RS6000, FreeBSD, and Linux workstations:
   $(CC) -O $(INCLUDE_DIR) test.c $(LIBS)
   $(F77) -O $(INCLUDE_DIR) test.f $(LIBS)

On Solaris workstations:
   $(CC) -O $(INCLUDE_DIR) test.c $(LIBS) -lsocket -lnsl -laio -lthread
   $(F77) -O $(INCLUDE_DIR) test.f $(LIBS) -lsocket -lnsl -laio -lthread
```

```
On DEC Alpha workstations:
   $(CC) -O $(INCLUDE_DIR) test.c $(LIBS) -laio
   $(F77) -O $(INCLUDE_DIR) test.f $(LIBS) -laio
```

Run the program as you would run any MPI program on the machine. If you use `mpirun`, make sure you use the correct `mpirun` for the MPI implementation you are using. For example, if you are using MPICH on an SGI machine, make sure that you use MPICH's `mpirun` and not SGI's `mpirun`.

# 5 Limitations of This Version of ROMIO

- The `status` argument is not filled in any function. Consequently, `MPI_Get_count` and `MPI_Get_elements` will not work when passed the `status` object from an MPI-IO operation.
- All nonblocking I/O functions use a ROMIO-defined `MPIO_Request` object instead of the usual `MPI_Request` object. Accordingly, two functions, `MPIO_Test` and `MPIO_Wait`, are provided to test and wait on these `MPIO_Request` objects. They have the same semantics as `MPI_Test` and `MPI_Wait`.

  ```
  int MPIO_Test(MPIO_Request *request, int *flag, MPI_Status *status);
  int MPIO_Wait(MPIO_Request *request, MPI_Status *status);
  ```

  The usual functions `MPI_Test`, `MPI_Wait`, `MPI_Testany`, and so forth, will not work for nonblocking I/O.
- This version works only on a homogeneous cluster of machines, and only the "native" file data representation is supported.
- This version works only for files of size less than 2 Gbytes. Accordingly, `MPI_Offset` is of type `integer`, and Fortran programs must use file offsets, file displacements, and so on, of type `integer` (not `integer*8`).
- All functions return only two possible error codes—`MPI_SUCCESS` on success and `MPI_ERR_UNKNOWN` on failure.

# 6 Usage Tips

- When using IBM's PIOFS file system, open the file with the `MPI_MODE_UNIQUE_OPEN` amode whenever possible. Certain collective I/O optimizations cannot be done if this amode is not used.
- When using ROMIO with SGI's MPI implementation, you may sometimes get an error message from SGI's MPI: "MPI has run out of internal datatype entries. Please set the environment variable `MPI_TYPE_MAX` for additional space." If you get this error message, add the following line to your .cshrc file:

  ```
  setenv MPI_TYPE_MAX 65536
  ```

  Use a larger number if you still get the error message.
- If a Fortran program uses a file handle created using ROMIO's C interface, or vice versa, you must use the functions `MPI_File_c2f` or `MPI_File_f2c` (see Sec. 4.12.4 in [1]). Such a

situation occurs, for example, if a Fortran program uses an I/O library written in C with MPI-IO calls. Similar functions `MPIO_Request_f2c` and `MPIO_Request_c2f` are also provided.

- For Fortran programs on the Intel Paragon, you may need to provide the complete path to `mpif.h` in the `include` statement, for example,

  ```
  include '/home/mpich-1.1.0/include/mpif.h'
  ```

  instead of

  ```
  include 'mpif.h'
  ```

  The reason is that the `-I` option doesn't work on the Paragon Fortran compiler `if77`. It always looks in the default directories first and, therefore, may pick up Intel's `mpif.h`, which is actually the `mpif.h` of an older version of MPICH.

# 7 ROMIO Users Mailing List

Please register your copy of ROMIO with us by sending e-mail to `majordomo@mcs.anl.gov` with the message

```
subscribe romio-users
```

This will enable us to notify you of new releases of ROMIO as well as bug fixes.

# 8 Reporting Bugs

If you have trouble, first check the users guide. Then check the on-line list of known bugs and patches at `http://www.mcs.anl.gov/home/thakur/romio`. Finally, if you still have problems, send a detailed message containing:

- the type of system (often `uname -a`),
- the output of `configure`,
- the output of `make`, and
- any programs or tests

to `romio-maint@mcs.anl.gov`.

# 9 ROMIO Internals

A key component of ROMIO that enables such a portable MPI-IO implementation is an internal abstract I/O device layer called ADIO [2]. Most users of ROMIO will not need to deal with the ADIO layer at all. However, ADIO is useful to those who want to port ROMIO to some other file system. The ROMIO source code and the ADIO paper [2] will help you get started.

# References


[1] Message-Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*. July 1997. On the World-Wide Web at `http://www.mpi-forum.org/docs/docs.html`.

[2] R. Thakur, W. Gropp, and E. Lusk. An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. In *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation*, pages 180–187, October 1996.