Using MPI on the Cray T3D

Kenneth Cameron, Lyndon J. Clarke, A. Gordon Smith

30th November 1995

Contents

1 Introduction

This document describes how to compile and run applications using the implementation of MPI [1] for Cray T3D by Edinburgh Parallel Computing Centre (EPCC) and Cray Research (CRI). For the purposes of this document it is assumed that the reader has an understanding of MPI. There is an increasing amount of general information on using the MPI interface including a book [2], a world-wide web page [3] and a Frequently Asked Questions (FAQ) list [4].

The information in this document is applicable to the Cray T3D system `darwin.edinburgh.ac.uk` *at The University of Edinburgh at the time of writing. Differences in installed compilers and the system configuration may invalidate this information. Please refer to your local user support team for assistance with any system variations.*

2 Compiling MPI Programs

This section describes compilation procedure and covers language binding issues for MPI programs written in C and Fortran on Cray T3D.

2.1 Compiling MPI Programs written in C

For MPI applications written in C, one must cross-compile for the T3D on the front-end (a Cray PVP system, eg. Y-MP), using the `-T` option to specify the required architecture, `cray-t3d`. The user is required to link with the MPI (`libmpi.a`) library.

The MPI implementation has been enhanced to optimise transfer, packing and unpacking of word-strided data. This makes use of Fortran BLAS [9] routines and so requires explicit linking with the BLAS library `libsci.a`.

It is possible to reduce the size of executables by fixing the number of processes making up the application at compile time by specifying the number of processes using the `-X` option (passed through to the MPP link loader). If fixed in this way the number of processes to be run cannot be overridden on execution, as is possible for executables that do not use MPI (including PVM and shared memory applications).

An example of compiling a single C file into a T3D executable to be run on 4 PEs follows. These examples assume that the MPI headers and library are held in locations which are searched by the compilers by default; the standard directories are `/usr/include/mpp` for headers and `/mpp/lib` for libraries. If the headers and libraries are held elsewhere it may be necessary to specify the location of each using compiler options: `-I` for headers and `-L` for libraries. In the example, the executable is written in the current working directory as file `mytest`.

```
cc -Tcray-t3d -X4 mytest.c -o mytest -lmpi -lsci
```

It is possible to use the C bindings to MPI from C++ programs: the C linkage directive is asserted within the MPI C header `mpi.h`. Please refer to the C++ compiler manual pages (`man 1 CC`) for details.

2.2 Compiling MPI Programs written in Fortran

Compilation of MPI applications written in Fortran is similar to that for C, except that the Fortran compiler requires the `-C` option to specify the cross-compilation architecture. In addition, unless the Fortran source uses the preprocessor (via `#include`) to access the MPI header, it is necessary to specify the location of the MPI header file using the `-I` option. The Fortran analogue of the above example would be:

```
cf77 -Ccray-t3d -I/usr/include/mpp mytest.f -X4 -o mytest -lmpi
```

The reader should refer to the Cray C [5] and Fortran [6] documentation for other useful compiler options. In particular, it is possible to compile MPI programs to allow use of the TotalView[1] debugging tool and the Apprentice[1] profiling tool by adding appropriate information to the generated executable. The `-g` compiler option includes debugging information with C and Fortran executables. Profiling information can be added to an executable using options `-happrentice` and `-Wf"-Ta"` for C and Fortran respectively; in addition it is necessary to link with the Apprentice library `libapp.a`. The compile lines to allow use of the debugger and profiling tool with the C and Fortran examples above would be:

```
cc -Tcray-t3d -g -happrentice mytest.c -X4 -o mytest \
     -lmpi -lsci -lapp

cf77 -Ccray-t3d -I/usr/include/mpp -g -Wf"-Ta" mytest.f \
     -X4 -o mytest -lmpi -lapp
```

2.3 Fortran MPI Issues

There are some subtleties in using MPI from Cray Fortran programs which should be highlighted. Cray Fortran does not provide the `DOUBLE PRECISION` language type, but allows compilation of Fortran where this type is interpreted as `REAL*8` using the `-dp` compiler option. The MPI basic datatype `MPI_DOUBLE_PRECISION` is defined regardless and correctly matches use of the language type when validated by the `-dp` option. (The optional MPI basic datatype `MPI_REAL8` is also provided.)

It is currently not possible to pass Fortran `CHARACTER` arrays and strings as the "choice" buffer argument to MPI calls (eg. buffer argument to `MPI_SEND`). It is possible to circumvent this problem using the Fortran `EQUIVALENCE` mechanism. The following Fortran code fragment illustrates this workaround, using an equivalent `INTEGER` array:

```
C ...
      character*24 msg
      equivalence (msg, ibogus)
      integer ibogus(4)
C ...
      msg = "global greetings"
      call MPI_SEND(ibogus, 24, MPI_CHARACTER, 0, 99,
     +              MPI_COMM_WORLD, ierr)
C ...
```

The equivalent array must be large enough to accommodate the `CHARACTER` data; the sizing of the arrays in the example above relies on implicit knowledge that an `INTEGER` comprises eight `CHARACTER` elements. Compiler errors are generated if misaligned data is equivalenced; this can occur for `CHARACTER` arrays within common blocks. In this case it will be necessary to pad common block `CHARACTER` arrays to multiples of eight. Warnings when equivalencing aligned common block data can be ignored safely.

It is possible for the MPI for T3D implementation to correct this problem, but only at significant performance penalty to all MPI calls from Fortran. The current situation is preferred, where the given workaround can be employed where it is necessary that `CHARACTER` data be manipulated using MPI.

[1] Trade Marks, Cray Research Inc.

2.4 Using the MPI Profiling Interface

The MPI Profiling Interface provides a name-shifted clone of every interface call, prefixed by `PMPI_` instead of `MPI_`. This allows replacement of a subset of MPI interface calls with user routines that can access the standard functionality through the corresponding name-shifted call. This is useful, in particular, for implementing MPI profiling by *instrumenting* MPI calls transparently.

The standard MPI for T3D library does not contain the name-shifted calls, as there is a small per-call penalty for uninstrumented calls. To use the Profiling Interface it is necessary to link against an extended library, `libpmpi.a`. System administrators may choose not to install this library in the standard location; in this case it is necessary to specify the link path using `-L` compiler option.

Fortran compilation generates `CAUTION` messages indicating duplicate symbols for any MPI calls that have been instrumented. These messages can be safely ignored, though it is possible to disable them by passing a directive through to the MPP link loader from the Fortran compile line (see example below). The following example shows compilation of C and Fortran programs with the Profiling Interface MPI library, where the library is held in directory `/home/mpi`.

```
cc -Tcray-t3d -X4 mytest.c -o mytest -L/home/mpi -lpmpi -lsci

cf77 -Ccray-t3d -I/usr/include/mpp mytest.f -X4 \
     -Wl"-Ddupentry=NOTE" -o mytest -L/home/mpi -lpmpi
```

2.5 Using MPI 2 One-Sided Communications

MPI for Cray T3D has been extended with a prototype implementation of of the current MPI 2 One-sided Communications chapter – a proposed standard for remote memory read and write capability within MPI; a separate document [10] discusses the use of this interface.

The addition of these services incurs a very small overhead on standard MPI operations. Nevertheless, the new services are provided by a separate library to avoid any penalty on standard MPI use. To access the prototype MPI 2 features, one must link against a separate library, `libmpi2.a`. System administrators may choose not to install this library in the standard location; in this case it is necessary to specify the link path using `-L` compiler option. Examples of compilation for C and Fortran test programs are given:

```
cc -Tcray-t3d -I/home/mpi mytest.c -o mytest -L/home/mpi -lmpi2 -lsci
cf77 -Ccray-t3d -I/home/mpi mytest.f -o mytest -L/home/mpi -lmpi2
```

3 Running MPI Applications

The Cray C and Fortran compilers, when cross-compiling for the T3D, add a directive to have the executable load itself onto T3D PEs. Running the executable on the front-end invokes the `mppexec` command as the first step. If the number of processes to be run was fixed on compilation, `mppexec` will run the executable on that number of PEs. It is also possible to invoke the `mppexec` command directly on an executable.

Following the example above, the executable `mytest` would be run on a partition of 4 PEs in either of the following ways.

```
./mytest
mppexec mytest
```

If the executable to be run was not compiled for a specified number of PEs, the number of PEs to run can be given to `mppexec` using the `-npes` argument. The following commands will run the executable on a partition of 8 PEs (assuming that it has not been compiled to run on a different number of PEs).

```
./mytest -npes 8
mppexec -npes 8 mytest
```

Each of the above forms of invocation can appear in a script submitted to NQS, allowing execution of MPI applications under batch operation; please refer to NQS documentation [7] and local queue configuration information for details of submitting, deleting, and querying jobs. An example session, where the above executable is submitted as a job (called `myjob`) to the `day` queue to be run on a 4 PE partition for a maximum of ten minutes, follows.

```
% cat day.qsub
#QSUB -r myjob
#QSUB -l mpp_t=0:10:00
#QSUB -l mpp_p=4
#QSUB -q day
echo Test running
cd ~kulidna/test
./mytest -npes 4
echo Test done

% qsub day.qsub
% qstat -a
...
```

4 Runtime User Control

A set of environmental variables allow runtime user control over various properties of MPI operation. Primarily, control over the amount of memory used by MPI is provided to facilitate tight user memory allocation in memory-bound applications.

`MPI_BUFFER_MAX`: specifies the maximum message size (in bytes) that will be successfully buffered in MPI Standard, Buffered and Ready send modes. By default, there is no limit imposed by the MPI implementation.

`MPI_BUFFER_TOTAL`: specifies the total amount of memory, separate from user memory donated through use of `MPI_Buffer_attach()`, available for message buffering in MPI Standard, Buffered and Ready send modes. By default, there is no limit imposed by the MPI implementation.

`MPI_SM_POOL`: specifies the maximum number of communications, involving a target process and any other process, that can be pending at any time. Communications are pending when the target process is not engaged in MPI services – for example, during lengthy computation. The default value for this quantity is 1024.

`MPI_SM_TRANSFER`: specifies the number of `SM_POOL` "slots" that may be occupied by a single send at the destination process. That is, a send operation will use up to this number of `SM_POOL` resources on the destination process while it is not performing MPI operations. Each "slot" comprises 4 Cray T3D words: that is 4 integers, 8 short integers, or 32 characters. For example, a send of 32 integer values will occupy 8 "slots". This relates to the function of the `MPI_SM_POOL` control, which originally assumed a single "slot" used for each outstanding communication. A sending process will limit itself to using a single "slot" if `(MPI_SM_POOL/2)` "slots" are in use at the destination; this guarantees that half of the SM_POOL “slots” are available for individual outstanding communications. The default value of this quantity is 128; setting the value to zero is allowed and prevents sends from using more than one remote “slot”.

MPI *send modes* define the completion conditions for an MPI send operation. Standard, Buffered and Ready send modes will complete after some period that is not dependent on any other process, ie. they are *local* operations, if there is a sufficient amount of memory available for message buffering. The quantities given by MPI_BUFFER_MAX and MPI_BUFFER_TOTAL define limits on this memory. The behaviour of an MPI send operation if this limit is exceeded depends on the send mode: Standard and Ready mode sends will complete only when a matching receive operation occurs at the destination process; Buffered mode sends will generate an error (of class MPI_ERR_BUFFER) and return immediately. Note that the nonblocking send operations always return immediately; completion (as discussed above) is indicated by success of a subsequent wait or test operation.

5 Debugging and Profiling MPI Applications

If an MPI application run terminates prematurely with a fault, a core file should be generated in the current working directory as file mppcore. If debugging information has been added to the executable (see § 2), it is possible to debug the application *post mortem* using the TotalView debugger tool. For example, if the executable mytest terminated due to a fault it would be possible to get a snapshot of the failure using the following TotalView command line:

```
totalview ./mppcore
```

If the number of processes to be run was specified at compile time, it is also possible to interactively debug an MPI application using TotalView by specifying the -debug option to mppexec when executing the application in the background. This delays the application until a debugger process attaches and assumes control of execution. A process identifier is displayed (<xxxx> below) and must be specified to Totalview via the -p option to attach the debugger to the delayed application run. If mppexec -debug is not run in the background it is necessary to kill the appropriate processes manually from another login as it is not possible to interrupt or suspend it. A sample session based on the running example follows:

```
% mppexec -debug ./mytest &
[<job>] <xxxx>
%
% totalview -p <xxxx> ./mytest
```

To profile MPI applications, compile with the Apprentice options and library. Execution of the application should generate a file called app.rif in the working directory. MPI calls will appear as uninstrumented function calls; there is no intrinsic overhead analysis of MPI communications time as there is for PVM.

Both tools have an X windows interface, requiring that display information and authorities are set up correctly. Refer to the TotalView documentation [8] for more information on use of the debugger facilities.

6 Further Information

Dynamically changing information, such as lists of known problems for releases of the T3D MPI software, will be made available via the World-Wide Web (WWW), Electronic mail, or surface mail (in order of convenience and preference).

URL:	`http://www.epcc.ed.ac.uk/t3dmpi/Product`
E-mail:	`t3dmpi@epcc.ed.ac.uk`
Postal:	**T3D MPI Support,** Edinburgh Parallel Computing Centre, The University of Edinburgh, JCM Building, King's Buildings, Mayfield Road, Edinburgh EH9 3JZ. United Kingdom.

References

[1] MPI: A Message-Passing Interface Standard. Message Passing Interface Forum. v1.1 June 1995.

[2] Using MPI, Portable Parallel Programming with the Message Passing Interface. William Gropp, Ewing Lusk, Anthony Skjellum. October 1994. MIT Press ISBN 0-262-57104-8.

[3] World-wide Web (WWW) URL: `http://www.mcs.anl.gov/mpi`. Mathematics and Computer Science Division, Argonne National Laboratory.

[4] Message Passing Interface (MPI) FAQ. Monthly posting to USENET newsgroup `comp.parallel.mpi`. Nathan Doss. Mississippi State University NSF Engineering Research Center for Computational Field Simulation.

[5] Cray Standard C Reference Manual, publication SR-2074. Cray Research Inc.

[6] CF77 Fortran Language Reference Manual, publication SR-3772. Cray Research Inc.

[7] Network Queuing System (NQS) User's Guide, publication SG-2105. Cray Research Inc.

[8] Introducing the Cray TotalView Debugger, publication IN-2502. Cray Research Inc.

[9] Scientific Libraries Reference Manual. publication SR-2081. Cray Research Inc.

[10] Using MPI 2 One Sided Communications on Cray T3D. A. Gordon Smith. November 1995. Edinburgh Parallel Computing Centre.