MPI Regression Testing and Migration to Origin2000

Terry Nelson
Sterling Software, Inc.
NASA Ames Research Center
Moffett Field, CA. 94035 USA
tnelson@nas.nasa.gov

Abstract: The computing industry, for economic and technical reasons, is moving inexorably towards an increasingly parallel environment. One of the major paradigms to accomplish this is message passing, and one of the major tools in this area is MPI. This paper will describe a set of Fortran and C regression tests which test MPI functionality and performance. A series of runs on a C90, J90, and Origin 2000, as well as tests with production sized jobs, will be described. Particular attention will be given to the migration of programs from the C90 and J90 environments to the Origins.

Numerical Aerospace Simulation (NAS) Systems Environment

To get a sense of the environment in which various tests were run and conclusions reached, the following is the NAS hardware configuration and software levels.

NAS Systems:
C90:
Von Neumann - 16 CPUs, 1Gw - Unicos 10.0.0.1
Eagle - 8 CPUs, 256Mw - Unicos 9.0.2.5 (no MPT module)

J90:
Newton1 - 16 CPUS, 512 Mw - Unicos 9.0.2.6
Newton2 - 12 CPUs, 512 Mw - Unicos 9.0.2.6
Newton3 - 4 CPUs, 128 Mw - Unicos 9.0.2.6
Newton4 - 4 CPUs, 128 Mw - Unicos 9.0.2.6

Origin2000:
Turing - 64 CPUs - IRIX 6.4
Evelyn/Hopper - 8 CPU frontend / 64 CPUs - IRIX 6.4
Jimpf1/Jimpf2 - 64 CPUs / 64 CPUs - IRIX 6.4
Heisenberg/Steger - 128 CPUs / 128 CPUs - IRIX 6.5

Job Scheduler On All Systems - Portable Batch System (PBS)

MPI Overview

This paper discusses functional regression tests for the Message Passing Interface (MPI) , so a brief overview of what MPI is may be in order.

The Message Passing Interface (MPI) is a portable standardized library of routines which deal with message passing, primarily on parallel computer systems.

It was established in 1992 initially by the "Workshop on Standards for Message Passing", and consists of about 129 calls which perform a variety of message passing and context creation functions. There are bindings for both Fortran and C. On SGI/Cray systems the standard release version is packaged in the MPT module, along with PVM and shmem.

MPI Functional Regression Tests

This paper describes a set of 160 functional regression tests, about 80 each written in Fortran and C, which test every MPI call. This set of tests, initially developed on an IBM SP2, was created for several reasons:

1) It was a learning tool for MPI.

2) MPI texts never give enough examples, or examples of the more complicated or 'obscure' calls, or often even discuss them.

3) Text examples are more often in C than Fortran, whereas the users at NAS run mainly large Fortran codes..

4) Short tests are handy to provide as examples to users or colleagues.

5) They are useful for more general testing, e.g. testing job scheduling software, an active area at NAS.

6) They have been very useful in testing compliance to the MPI standard.

There is a common format for all of the tests. For example, all Fortran tests start like the following:

```
     program t1
CCCCCCCCCCCCC
C
C   MPI_SEND
C   MPI_RECV
C    blocking send
C    standard communication mode
C   2 nodes
C
CCCCCCCCCCCC
      include 'mpif.h'
      integer status (MPI_STATUS_SIZE)
       ...
```

Inside the comment block are found the names of the MPI calls which this program tests, some pertinent comments about blocking, modes, or other relevant information, and the number of nodes the test uses.

Similiarly, all of the Fortran tests end like the following:

```
  if(myid.eq.0) then
      print *,'Test t1 at end'
   endif
   call MPI_FINALIZE(ierr)
   end
```

Currently, node 0 writes an end-of-test message. The real sense of the success of the test arises not only from node 0's message, but from examination of all of the output files. Even though normally tests in error are quickly perceived, some further actions on termination are being considered.

PBS - At NASA Ames (and at about 70 other sites in the USA) the job management program is not the vendor

supplied or supported one, e.g. NQE, or LSF, but is the locally developed Portable Batch System (PBS). This has implications for run scripts, but should not effect the test set per se.

The common test format also applies to all C tests. For example, all C tests start like the following:

```
#include <stdio.h>
#include "mpi.h"
main(int argc, char **argv)
{
/*                                                      */
/*  test t1                                             */
/*  MPI_SEND                                            */
/*  MPI_RECV                                            */
/*   blocking send                                      */
/*   standard communication mode                        */
/*  2 nodes                                             */
/*                                                      */
       MPI_Status status;
```

Again, as with Fortran, inside the comment block are found the names of the MPI calls which this program tests, some pertinent comments about blocking, modes, or other relevant information, and the number of nodes the test uses.
Similiarly, all of the C tests end like the following:

```
       if(myid == 0)
         {
         printf(" Test t1 at end\n");
         }
       ierr = MPI_Finalize();
}
```

Regression Test Crosstable

A crosstable is provided which lists all of the standard MPI calls, and in the following line the names of at least some of the jobs which use, and therefore test that particular call. In front of the name of the call is a 2 digit code, which contains references to the text

"MPI:The Complete Reference", M. Snir, S. Otto, S. Huss-Lederman,
D. Walker, J. Dongarra

The first number refers to the chapter in which the call is found, which thereby provides a clue to the functional category of the call. After a dash the second number is the page number of the description. This provides a quick guide to further study of the MPI call of interest.

The names of the tests currently consist of a letter, t-z, a number, and a .f or .c. While the test name is not an immediate intuitive guide to its function, there are a number of organizing files and tables to track the tests and the results of test runs.

The following is a sample from the point-to-point functional area:

2-18 MPI_SEND
t1,t111

2-22 MPI_RECV
t1,t111,t2,t3,t33,t4,t44
2-24 MPI_GET_COUNT
t10
2-90 MPI_BSEND
t2
2-90 MPI_SSEND
t3,t33
2-91 MPI_RSEND
t4,t44
2-95 MPI_BUFFER_ATTACH
t2,t6,t7
2-96 MPI_BUFFER_DETACH
t7
2-51 MPI_ISEND
t5,t55,t555
2-92 MPI_IBSEND
t6
2-92 MPI_ISSEND
t7,t77,z7

Regession Test Package

It is planned to make this regression test set available on the web in the late summer of 1998. That package would include:

- Fortran source code

- C source code

-make files for compilation, using modules

- run scripts, setup for PBS, and hopefully NQE.

- a function call/routine crosstable

- a readme with instructions and status

Sites already running MPI should have no trouble compiling the programs, and converting the scripts, if necessary.

Possible Future Test Directions

There are several ideas in mind which could increase the utility of this software.

- The syntax of the Fortran tests is currently in f77. It could be translated into more specific Fortran 90 syntax.

- All of the attention so fall has been paid to MPI-1. As the upgraded MPI-2 becomes increasingly available, these tests could be updated to handle those features as well.

- The current error checking methods are fast and straightforward, but they are visual. There may be opportunities to automate some of this task.

- These tests deal with the functionality of MPI calls, not their bandwidth or latency. Such extensions are being considered.

- The MPI_PCONTROL and MPI_ERRHANDLER calls depend to a large extent on user provided code outside the scope of MPI. This code for these tests is not as extensive is it could be.

Larger MPI Tests - C90/J90

The small regression tests can often zip in and out of the system without showing some of the throughput problems which can arise with large production codes using MPI. Also, MPI applications seem to have never been especially popular on C90/J90 platforms. To learn more about the reasons why that might be, several tests, including a job which partitions, maps, and increments values among the nodes, were run at production sizes.

Conclusions:

1) On dedicated systems, mpirun -nt (shared memory/HW communication) runs up to 40 times faster than mpirun -np (TCP/sockets).

2) On busy systems,
-nt will spin wait on messages and CPU timeout.
-np will wait on system calls and wall timeout.

When these were completed the case seemed quite closed. But since then, other users are trying to use MPI on Von Neumann, and other analysts are searching for scheduling parameters, or other factors that may clarify or improve MPI performance. So this issue remains at the moment a question and an area still being investigated.

Cross Platform Running

One of the more interesting variations on these themes is the ability to run an MPI job using processors on distinct mainframes. This involves specific parameters in the array_services files, which were ported from SGI platforms to the C90/J90s in MPT.1.2. This only involves the -np option of mpirun. The following example shows the mpirun command of a job, launched from the J90 newton4, and running on both newton4 and the C90 vn. The binary `ary` was already in the appropriate directory on vn.

```
...
mpirun -a n4vn -d ~/mpidir/tdir -p "<%g:%G>" newton4 2 ./ary : vn 4 ./ary

newton4.tnelson 55> ./test2
<0:6> I am node  0  out of  6  nodes.
<0:6> Name is newton4
<1:6> I am node  1  out of  6  nodes.
<1:6> Name is newton4
<0:6>unicos newton4 9.0.2.6 nas.14 CRAY J90
<0:6>  ==> Test ary is done.
<2:6> I am node  2  out of  6  nodes.
<2:6> Name is vn
<1:6>unicos newton4 9.0.2.6 nas.14 CRAY J90
<3:6> I am node  3  out of  6  nodes.
<3:6> Name is vn
<4:6> I am node  4  out of  6  nodes.
<4:6> Name is vn
```

```
<5:6> I am node  5  out of  6  nodes.
<5:6> Name is vn
<2:6>unicos vn 9.0.2.6 nas.34 CRAY C90
<3:6>unicos vn 9.0.2.6 nas.34 CRAY C90
<4:6>unicos vn 9.0.2.6 nas.34 CRAY C90
<5:6>unicos vn 9.0.2.6 nas.34 CRAY C90
```

Origin Conversion Issues

Most of the problems involved with conversion and performance on Origins seem to concern memory, in one way or another.

Memory locality:

Being a cache based system, and with memory distributed across distant hubs, locality of one's program and data becomes of paramount concern. Jim Taft, of MRJ, showed a nice way to illustrate this a while back:

Compute at the speed of light philosophy

- Running from 1st cache == speed of light compute = SOL
- Running from 2nd cache == 10 clock hit = 10% SOL
- Running from memor == 100 clock hit = 1% SOL
- Running from disk == speed of sound = SOS (help)

In general, vector friendly code should be cache friendly, but not always.

On a C90/J90, the inner loop should be largest, to take advantage of the vector registers.

```
DO J=1,5
DO I=1,IMAX
....
```

On an Origin, it may be possible get pipelining of separate variables along separate cache lines by reversing the order.

```
DO I=1,IMAX
DO J=1,5
....
```

However, this requires a detailed understanding of the Origin hardware as well as the way your problem and data are laid out.

CPU scheduling:

This might appear to be unrelated to memory issues, but that is definitely not the case. Distant assigned memory leads to the above indicated problems, as well as increasing the chances of distant CPU conflicts with competing jobs or system activities.
There have been several approaches to this issue, such as 'nodemask' and the earlier 'psets'. Static nodemasks can leave some CPUs underutilized, depending on the job mix, and in any event is purely advisory to the system. NAS is currently exploring a 'dynamic nodemask' approach, to adjust more quickly to the pool of available CPUs. An SGI product to handle these functions which is new in IRIX 6.5 is called 'MISER'.

Buffering:

Some conversion problems involving MPI_BSEND/MPI_IBSEND were resolved by requesting much larger buffer space.
But even with other kinds of MPI message passing commands, the system buffers all messages of a certain size. For jobs which are quite large and very asynchronous, that can create memory problems. One approach to this problem involves changing environment variable settings. They can be found with the `'man mpi'` command, which in IRIX 6.5 includes their default values.

New approaches:

Traditionally, one would not mix the older SGI MP_ style of Parallel Fortran shared memory commands, such as $DO_ACROSS with the distributed memory paradigm of MPI calls. However,analysts at SGI have run tests of this method, and one analyst in our group at NAS, Dr. Samson Cheung, is co-authoring a paper this summer on a technique to mix the two systems, and thereby try to profit from the advantages of both.The technique involves combining both threads and process parameters.

Web Addresses and References

The following is an excellent starting point for MPI on the web, with link in many different directions.

http://www.mcs.anl.gov/mpi/index.html

There is an MPI course in German from the University of Tuebingen at

http://alphamor.zdv.uni-tuebingen.de/PARALLEL/uni-tue/ZDV/lectures/mpi.ps.gz

The text used as page references for MPI commands is:

"MPI:The Complete Reference", M. Snir, S. Otto, S. Huss-Lederman,
D. Walker, J. Dongarra

The quote from Jim Taft is from a lecture:

"Early Experience with the SGI Origin 2000"

Summary and Conclusions

- Test sets, such as the regression tests discussed here, can help to ensure consistent functionality and performance for MPI and other products, over upgrades and new releases.

- Getting the most performance out of Origins requires paying attention to cache and memory issues. Some SGI tools, such as perfex, dplace, ssrun, or proprietary tools such as Xforge may be of assistance.

- New combinations, whether between platforms, or mixing products within one platform, can lead to exciting possibilities.