MPI Quick Reference: Compiling/Running

M. D. Jones, Ph.D.

Center for Computational Research
University at Buffalo
State University of New York

High Performance Computing I, 2014

Background

This document covers the essentials of compiling and running MPI applications on the CCR platforms. It does not cover MPI programming itself, nor debugging, etc. (covered more thoroughly in separate presentations).

Modules Software Management System

There are a large number of available software packages on the CCR systems, particularly the Linux clusters. To help maintain this often confusing environment, the **modules** package is used to add and remove these packages from your default environment (many of the packages conflict in terms of their names, libraries, etc., so the default is a minimally populated environment).

The module Command

module command syntax:

```
1  -bash-2.05b$ module help
2
3    Modules Release 3.1.6 (Copyright GNU GPL v2 1991):
4    Available Commands and Usage:
5          + add|load              modulefile [modulefile ...]
6          + rm|unload             modulefile [modulefile ...]
7          + switch|swap           modulefile1 modulefile2
8          + display|show          modulefile [modulefile ...]
9          + avail                 [modulefile [modulefile ...]]
10         + use [-a|--append]     dir [dir ...]
11         + unuse                 dir [dir ...]
12         + update
13         + purge
14         + list
15         + clear
16         + help                  [modulefile [modulefile ...]]
17         + whatis                [modulefile [modulefile ...]]
18         + apropos|keyword       string
19         + initadd               modulefile [modulefile ...]
20         + initprepend           modulefile [modulefile ...]
21         + initrm                modulefile [modulefile ...]
22         + initswitch            modulefile1 modulefile2
23         + initlist
24         + initclear
```

Using module in Batch

If you change shells in your batch script you may need to explicitly load the modules environment:

tcsh :

```
1 source $MODULESHOME/init/tcsh
```

bash :

```
1 . ${MODULESHOME}/init/bash
```

but generally you should not need to worry about this step (do a "module list" and if it works ok your environment should already be properly initialized).

Objective: Construct a very elementary MPI program to do the usual "Hello World" problem, i.e. have each process print out its rank in the communicator.

in C

```
1  #include <stdio.h>
2  #include "mpi.h"
3
4  int main(int argc,char **argv)
5  {
6      int myid, nprocs;
7      int  namelen,mpiv,mpisubv;
8      char processor_name[MPI_MAX_PROCESSOR_NAME];
9
10     MPI_Init(&argc,&argv);
11     MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
12     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
13     MPI_Get_processor_name(processor_name,&namelen);
14
15     printf("Process %d of %d on %s\n",myid, nprocs, processor_name);
16     if (myid == 0) {
17        MPI_Get_version(&mpiv,&mpisubv);
18        printf("MPI Version: %d.%d\n",mpiv,mpisubv);
19     }
20     MPI_Finalize();
21     return 0;
22 }
```


CCR: Intel MPI

There are several commercial implementations of MPI, `Intel` and `IBM` currently being the most prominent. CCR has a license for `Intel` MPI, and it has some nice features:

- Support for multiple networks (Infiniband, Myrinet, TCP/IP)
- Part of the ScaLAPACK support in the Intel MKL
- MPI-2 features (one-sided, dynamic tasks, I/O with parallel filesystems support)
- CPU pinning/process affinity options (extensive)

Build the code with the appropriate wrappers:

```
[rush:~/d_mpi-samples]$ module load intel-mpi intel
[rush:~/d_mpi-samples]$ module list
Currently Loaded Modulefiles:
  1) null                2) modules             3) use.own
  4) intel-mpi/4.1.1   5) intel/13.1
[rush:~/d_mpi-samples]$ mpiicc -o hello.impi hello.c     # icc version
[rush:~/d_mpi-samples]$ mpiicc -show
icc -I/util/intel/impi/4.1.1.036/intel64/include -L/util/intel/impi/4.1.1.036/intel64/lib \
    -Xlinker --enable-new-dtags -Xlinker -rpath -Xlinker /util/intel/impi/4.1.1.036/intel64/lib \
    -Xlinker -rpath -Xlinker /opt/intel/mpi-rt/4.1 -lmpigf -lmpi -lmpigi -ldl -lrt -lpthread
[rush:~/d_mpi-samples]$ mpicc -o hello.impi.gcc hello.c  # gcc version
[rush:~/d_mpi-samples]$ mpicc -show
gcc -I/util/intel/impi/4.1.1.036/intel64/include -L/util/intel/impi/4.1.1.036/intel64/lib \
    -Xlinker --enable-new-dtags -Xlinker -rpath -Xlinker /util/intel/impi/4.1.1.036/intel64/lib \
    -Xlinker -rpath -Xlinker /opt/intel/mpi-rt/4.1 -lmpigf -lmpi -lmpigi -ldl -lrt -lpthread
```

Unfortunately Intel MPI can have somewhat flaky integration with Slurm, so you have a range of options when it comes to launching MPI codes.

Simple Example - Interactive

```
[rush:~/d_mpi-samples]$ export I_MPI_DEBUG=4
[rush:~/d_mpi-samples]$ mpirun -np 4 ./hello.impi
[0] MPI startup(): shm data transfer mode
[1] MPI startup(): shm data transfer mode
[2] MPI startup(): shm data transfer mode
[3] MPI startup(): shm data transfer mode
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       66649    k07n14     {0,4,8,12,16,20,24,28}
[0] MPI startup(): 1       66650    k07n14     {1,5,9,13,17,21,25,29}
[0] MPI startup(): 2       66651    k07n14     {2,6,10,14,18,22,26,30}
[0] MPI startup(): 3       66652    k07n14     {3,7,11,15,19,23,27,31}
Process 0 of 4 on k07n14
MPI Version: 2.2
Process 1 of 4 on k07n14
Process 2 of 4 on k07n14
Process 3 of 4 on k07n14
```


Simple Example - Slurm/srun

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --partition=debug
#SBATCH --time=00:10:00
#SBATCH --mail-type=END
#SBATCH --mail-user=jonesm@buffalo.edu
#SBATCH --output=slurmQ.out
#SBATCH --job-name=mpi-test
#
# Note the above directives can be commented out using an
# additional "#"
#
module load intel-mpi intel
#
# Intel MPI has flaky tight integration with Slurm,
# generally it has been safer to use Slurm's srun rather than
# rely on mpirun/mpiexec.
# You can find a description of all Intel MPI parameters in the
# Intel MPI Reference Manual,
# see <intel mpi installdir>/doc/Reference_manual.pdf
#
export I_MPI_DEBUG=4    # nice debug level, spits out useful info
#
# mpirun wrapper:
mpirun ./hello.impi
# srun:
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun ./hello.impi
```


Output from the `mpirun` launch (note that the pinning is not what we want by default, you can likely force better behavior with `I_MPI_PIN_MODE` and its associated variables):

```
[7] MPI startup(): shm data transfer mode
[2] MPI startup(): shm data transfer mode
[4] MPI startup(): shm data transfer mode
[6] MPI startup(): shm data transfer mode
[3] MPI startup(): shm data transfer mode
[1] MPI startup(): shm data transfer mode
[5] MPI startup(): shm data transfer mode
[0] MPI startup(): shm data transfer mode
Process 1 of 8 on d09n29s02
Process 7 of 8 on d09n29s02
Process 5 of 8 on d09n29s02
Process 3 of 8 on d09n29s02
Process 6 of 8 on d09n29s02
Process 2 of 8 on d09n29s02
Process 4 of 8 on d09n29s02
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       15265    d09n29s02  0
[0] MPI startup(): 1       15266    d09n29s02  0
[0] MPI startup(): 2       15267    d09n29s02  0
[0] MPI startup(): 3       15268    d09n29s02  0
[0] MPI startup(): 4       15269    d09n29s02  0
[0] MPI startup(): 5       15270    d09n29s02  0
[0] MPI startup(): 6       15271    d09n29s02  0
[0] MPI startup(): 7       15272    d09n29s02  0
Process 0 of 8 on d09n29s02
MPI Version: 2.2
```

Output from the `srun` launch (in this case +1 indicates that pinning is turned off):

```
[2] MPI startup(): shm data transfer mode
[3] MPI startup(): shm data transfer mode
[5] MPI startup(): shm data transfer mode
[6] MPI startup(): shm data transfer mode
[7] MPI startup(): shm data transfer mode
[0] MPI startup(): shm data transfer mode
[1] MPI startup(): shm data transfer mode
[4] MPI startup(): shm data transfer mode
Process 1 of 8 on d09n29s02
Process 3 of 8 on d09n29s02
Process 5 of 8 on d09n29s02
Process 7 of 8 on d09n29s02
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       15311    d09n29s02  +1
[0] MPI startup(): 1       15312    d09n29s02  +1
[0] MPI startup(): 2       15313    d09n29s02  +1
[0] MPI startup(): 3       15314    d09n29s02  +1
[0] MPI startup(): 4       15315    d09n29s02  +1
[0] MPI startup(): 5       15316    d09n29s02  +1
[0] MPI startup(): 6       15317    d09n29s02  +1
Process 2 of 8 on d09n29s02
Process 4 of 8 on d09n29s02
Process 6 of 8 on d09n29s02
[0] MPI startup(): 7       15318    d09n29s02  +1
Process 0 of 8 on d09n29s02
MPI Version: 2.2
```


Intel MPI on Infiniband

CCR nodes generally have Infiniband (IB) as the optimal interconnect for message-passing. Intel MPI should automatically find and use IB on those machines (they have 8, 12, 16, or 32 cores each, so adjust your script accordingly). Here we change the node count request to two, and just run with `srun`:

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --constraint=CPU-L5520
#SBATCH --partition=debug
```

```
[rush:~/d_mpi-samples]$ cat slurmQ.out
[1] MPI startup(): shm and ofa data transfer modes
[2] MPI startup(): shm and ofa data transfer modes
[9] MPI startup(): shm and ofa data transfer modes
[3] MPI startup(): shm and ofa data transfer modes
[10] MPI startup(): shm and ofa data transfer modes
[4] MPI startup(): shm and ofa data transfer modes
[11] MPI startup(): shm and ofa data transfer modes
[5] MPI startup(): shm and ofa data transfer modes
[12] MPI startup(): shm and ofa data transfer modes
[6] MPI startup(): shm and ofa data transfer modes
[13] MPI startup(): shm and ofa data transfer modes
[7] MPI startup(): shm and ofa data transfer modes
[0] MPI startup(): shm and ofa data transfer modes
```



```
Process 5 of 16 on d16n02
Process 10 of 16 on d16n03
Process 6 of 16 on d16n02
Process 11 of 16 on d16n03
Process 7 of 16 on d16n02
Process 13 of 16 on d16n03
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       7898     d16n02     +1
[0] MPI startup(): 1       7899     d16n02     +1
[0] MPI startup(): 2       7900     d16n02     +1
[0] MPI startup(): 3       7901     d16n02     +1
[0] MPI startup(): 4       7902     d16n02     +1
[0] MPI startup(): 5       7903     d16n02     +1
[0] MPI startup(): 6       7904     d16n02     +1
[0] MPI startup(): 7       7905     d16n02     +1
[0] MPI startup(): 8       29934    d16n03     +1
[0] MPI startup(): 9       29935    d16n03     +1
[0] MPI startup(): 10      29936    d16n03     +1
[0] MPI startup(): 11      29937    d16n03     +1
[0] MPI startup(): 12      29938    d16n03     +1
[0] MPI startup(): 13      29939    d16n03     +1
[0] MPI startup(): 14      29940    d16n03     +1
[0] MPI startup(): 15      29941    d16n03     +1
Process 14 of 16 on d16n03
Process 4 of 16 on d16n02
Process 15 of 16 on d16n03
Process 0 of 16 on d16n02
MPI Version: 2.2
Process 8 of 16 on d16n03
Process 12 of 16 on d16n03
```


Intel MPI on TCP/IP

You can force Intel MPI to run using TCP/IP (or a combination of tcp/ip and shared memory as in the example below) by setting the `I_MPI_DEVICE` variable, or equivalently `I_MPI_FABRICS`):

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --constraint=CPU-L5520
#SBATCH --partition=debug
...
export I_MPI_DEBUG=4
export I_MPI_FABRICS="ssm:tcp" # tcp/ip between nodes, shared memory within
```

```
Process 0 of 2 on d16n02
Process 1 of 2 on d16n03
[0] MPI startup(): shm and tcp data transfer modes
[1] MPI startup(): shm and tcp data transfer modes
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       9072     d16n02     +1
[0] MPI startup(): 1       30436    d16n03     +1
MPI Version: 2.2
Process 1 of 2 on d16n03
```

Intel MPI Summary

Intel MPI has some real advantages:

- Multi-protocol support with the same build, by default gives you the "best" network, but also gives you the flexibility to choose your protocol
- CPU/memory affinity settings
- Multiple compiler support (wrappers for GNU compilers, **mpicc, mpicxx, mpif90**, as well as Intel compilers, **mpiicc, mpicpc, mpiifort**)
- (Relatively) simple integration with Intel MKL, including ScaLAPACK
- Reference manual - on the CCR systems look at `$INTEL_MPI/doc/Reference_Manual.pdf` for a copy of the reference manual (after loading the module)

Whither Goest Thou, MPI?

MPI processes - things to keep in mind:

- You can over-subscribe the processors if you want, but that is going to under-perform (but it is often useful for debugging). Note that batch queuing systems (like those in CCR) may not let you easily over-subscribe the number of available processors
- Better MPI implementations will give you more options for the placement of MPI tasks (often through so-called "affinity" options, either for CPU or memory)
- Typically want a 1-to-1 mapping of MPI processes with available processors (cores), but there are times when that may not be desirable

Affinity

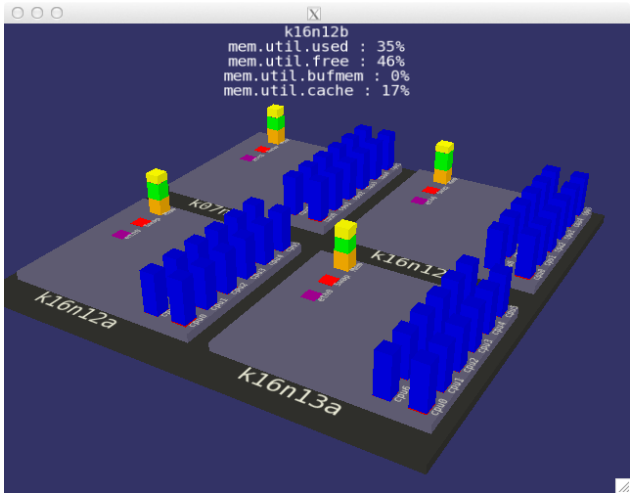
Intel MPI has options for associating MPI tasks to cores - better known as CPU-process affinity

- `I_MPI_PIN`, `I_MPI_PIN_MODE`, `I_MPI_PIN_PROCESSOR_LIST`, `I_MPI_PIN_DOMAIN` in the current version of Intel MPI (it never hurts to check the documentation for the version that you are using, these options have a tendency to change)
- Can specify core list on which to run MPI tasks, also domains of cores for hybrid MPI-OpenMP applications

Summary - MPI at CCR

- Use `modules` environment manager to choose your MPI flavor
- I recommend `Intel MPI` on the clusters, unless you need access to the source code for the implementation itself. It has a lot of nice features and is quite flexible.
- Be careful with task launching - use `srun` in Slurm jobs, or treat pinning with care
- Ensure that your MPI processes end up where you want - use `ps` and `top` to check (also use MPI_Get_processor_name in your code).
- Also use the CCR `slurmjobvis` job visualizer utility to quickly scan for expected task placement and performance issues.

`slurmjobvis` Example



Sample `slurmjobvis` from a 4-node job using all 12 cores/node and roughly 50% of available memory/node.