

Using the mbind Tool for Pinning

Category: Process Pinning

Summary: The `mbind` utility is a "one-stop" tool for binding processes and threads for MPI and OpenMP, and hybrid applications.

The `mbind` utility, developed at NAS, is used for binding processes and threads to CPUs. It works for MPI, OpenMP, or MPI+OpenMP hybrid applications, and is available under `/u/scicon/tools/bin` on Pleiades.

One of the benefits of `mbind` is that it relieves users from the burden of learning the complexity of each individual pinning approach for associated MPI or OpenMP libraries. It provides a uniform usage model that works for multiple MPI and OpenMP environments.

Currently supported MPI and OpenMP libraries are listed below.

MPI:

- SGI-MPT
- MVAPICH2
- INTEL-MPI
- OPEN-MPI
- MPICH2

OpenMP:

- Intel OpenMP runtime library
- GNU OpenMP library
- PGI runtime library
- Pathscale OpenMP runtime library

Use of `mbind` is limited to cases where the same set of CPU lists is used for all processor nodes, and the same number of threads is used for all processes.

WARNING: Be aware that `mbind` may have issues when used together with other performance tools, such as PerfSuite.

Syntax

```
#For OpenMP:  
mbind.x [-options] program [args]
```

```
#For MPI or MPI+OpenMP hybrid which supports mpiexec:  
mpiexec -np nrank mbind.x [-options] program [args]
```

Information about all available options can be found in the text file `/u/scicon/tools/doc/mbind.txt` on Pleiades.

Here are a few recommended `mbind` options:

`-cs`, `-cp`, `-cc`; or `-ccpulist`

`-cs` for spread (default), `-cp` for compact, `-cc` for cyclic; `-ccpulist` for process ranks (for example, `-c0, 3, 6, 9`). CPU numbers in the `cpulist` are relative within a cpuset if present.

Note that the `-cs` option will spread the processes and threads among the physical cores to minimize various resource contentions, and is usually the best choice for placement.

`-nn`

Number of processes per node.

`-tn`

Number of threads per process. The default value is given by the `OMP_NUM_THREADS` environment variable.

`-vn`

Verbose flag; print some useful information. `[n]` controls the level of details. Default is `n=0` (OFF).

Examples

For Pure OpenMP Codes Using Intel OpenMP Library

Sample PBS script:

```
#PBS -l select=1:ncpus=12:model=wes
#PBS -l walltime=0:5:0

module load comp-intel/11.1.072

cd $PBS_O_WORKDIR

mbind.x -cs -t4 -v ./a.out
```

The 4 OpenMP threads are spread (with the `-cs` option) among 4 physical cores in a node, as shown in the application's `stdout`:

host: r211i0n5, ncpus 24, nthreads: 4, bound to cpus: {0,3,6,9}
The proper placement is further demonstrated in the output of the `ps` command below:

```
r211i0n5% ps -C a.out -L -opsr,comm,time,pid,ppid,lwp
PSR COMMAND          TIME  PID  PPID  LWP
 9 a.out             00:02:06  849   711   849
 3 a.out             00:00:00  849   711   850
 3 a.out             00:02:34  849   711   851
 0 a.out             00:01:47  849   711   852
```

Note that Intel OpenMP creates an extra thread, which is unknown to the user and does not need to be placed. In this example, this extra thread (thread id 850) is running on the same core (core 3) as thread 851. Since this extra thread does not do any work, it will not interfere with the other threads.

For Pure MPI Codes Using SGI MPT

WARNING: `mbind.x` overwrites the placement initially performed by MPT's `mpiexec`. The placement output from `MPI_DSM_VERBOSE` (if set) most likely is incorrect and should be ignored.

Sample PBS script:

```
#PBS -l select=1:ncpus=12:model=wes

module load comp-intel/11.1.072
module load mpi-sgi/mpt.2.04.10789

#setenv MPI_DSM_VERBOSE

cd $PBS_O_WORKDIR

mpiexec -np 8 mbind.x -cs -n4 -v ./a.out
```

On each of the two nodes, 4 MPI processes are spread among 4 physical cores (0,3,6,9), as shown in the application's `stdout`:

```
host: r213i2n12, ncpus 24, process-rank: 0, bound to cpu: 0
host: r213i2n12, ncpus 24, process-rank: 1, bound to cpu: 3
host: r213i2n12, ncpus 24, process-rank: 3, bound to cpu: 9
host: r213i2n12, ncpus 24, process-rank: 2, bound to cpu: 6
host: r213i2n13, ncpus 24, process-rank: 4, bound to cpu: 0
host: r213i2n13, ncpus 24, process-rank: 5, bound to cpu: 3
host: r213i2n13, ncpus 24, process-rank: 6, bound to cpu: 6
host: r213i2n13, ncpus 24, process-rank: 7, bound to cpu: 9
```

For MPI+OpenMP Hybrid Codes Using SGI MPT and Intel OpenMP

Sample PBS script:

```
#PBS -l select=2:ncpus=12:mpiprocs=4:model=wes

module load comp-intel/11.1.072
module load mpi-sgi/mpt.2.04.10789

#setenv MPI_DSM_VERBOSE

cd $PBS_O_WORKDIR
```

```
mpiexec -np 8 mbind.x -cs -n4 -t2 -v ./a.out
```

On each of the two nodes, the 4 MPI processes are spread among the physical cores. The 2 OpenMP threads of each MPI process run on adjacent physical cores as seen in the application's **stdout**:

```
host: r215i2n12, ncpus 24, process-rank: 4, nthreads: 2, bound to cpus: {0-1}
host: r215i2n12, ncpus 24, process-rank: 6, nthreads: 2, bound to cpus: {6-7}
host: r215i2n12, ncpus 24, process-rank: 5, nthreads: 2, bound to cpus: {2-3}
host: r215i2n12, ncpus 24, process-rank: 7, nthreads: 2, bound to cpus: {8-9}
host: r215i2n11, ncpus 24, process-rank: 0, nthreads: 2, bound to cpus: {0-1}
host: r215i2n11, ncpus 24, process-rank: 2, nthreads: 2, bound to cpus: {6-7}
host: r215i2n11, ncpus 24, process-rank: 3, nthreads: 2, bound to cpus: {8-9}
host: r215i2n11, ncpus 24, process-rank: 1, nthreads: 2, bound to cpus: {2-3}
```

For MPI+OpenMP Hybrid Codes Using MVAPICH2 and Intel OpenMP

Sample PBS script:

```
#PBS -l select=2:ncpus=12:mpiprocs=4:model=wes

module load comp-intel/11.1.072
module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

mpiexec -np 8 mbind.x -cs -n4 -t2 -v ./a.out

#If you use mpirun_rsh instead of mpiexec
#use the following

mpirun_rsh -np 8 -hostfile $PBS_NODEFILE \
mbind.x -cs -n4 -t2 -v2 ./a.out
```

The application's **stdout** in this example is very similar to that in the previous MPT/Intel OpenMP example.

For MPI+OpenMP Hybrid Codes Using Intel MPI and Intel OpenMP

The Intel MPI library automatically pins processes to CPUs to prevent unwanted process migration. If you find that the placement done by the Intel MPI library is not optimal, you can use **mbind** to do the pinning instead.

WARNING: Note that in order for **mbind** to work with the Intel MPI library, the internal pinning mode of the library must be turned off explicitly by setting the environment variable **I_MPI_PIN** to *0*.

Sample PBS script:

```
#PBS -l select=2:ncpus=12:mpiprocs=4:model=wes

module load comp-intel/11.1.072
module load mpi-intel/4.0.2.003

setenv I_MPI_PIN 0

cd $PBS_O_WORKDIR

mpdboot --file=$PBS_NODEFILE --ncpus=1 --totalnum=`cat $PBS_NODEFILE | \
  sort -u | wc -l` --ifhn=`head -1 $PBS_NODEFILE` --rsh=ssh \
  --mpd=`which mpd` --ordered

mpiexec -ppn 4 -np 8 mbind.x -cs -n4 -t2 -v ./a.out

mpdallexit
```

For the above job, the following placement is seen in the application's **stdout**:

```
host: r215i2n11, ncpus 24, process-rank: 0, nthreads: 2, bound to cpus: {0-1}
host: r215i2n11, ncpus 24, process-rank: 1, nthreads: 2, bound to cpus: {2-3}
host: r215i2n11, ncpus 24, process-rank: 3, nthreads: 2, bound to cpus: {8-9}
host: r215i2n11, ncpus 24, process-rank: 2, nthreads: 2, bound to cpus: {6-7}
host: r215i2n12, ncpus 24, process-rank: 5, nthreads: 2, bound to cpus: {2-3}
host: r215i2n12, ncpus 24, process-rank: 4, nthreads: 2, bound to cpus: {0-1}
host: r215i2n12, ncpus 24, process-rank: 7, nthreads: 2, bound to cpus: {8-9}
host: r215i2n12, ncpus 24, process-rank: 6, nthreads: 2, bound to cpus: {6-7}
```

This can be confirmed by running the following **ps** command on the running nodes. For clarity, the extra OpenMP threads created by the Intel OpenMP (which don't do any work) are removed from the output.

```
r215i2n11% ps -C hybrid_intelmpi -L -opsr,comm,time,pid,ppid,lwp
PSR COMMAND      TIME    PID  PPID  LWP
  6 a.out         00:00:12 44698 44696 44698
  7 a.out         00:00:12 44698 44696 44715
  2 a.out         00:00:12 44699 44695 44699
  3 a.out         00:00:12 44699 44695 44711
  8 a.out         00:00:12 44700 44697 44700
  9 a.out         00:00:12 44700 44697 44713
  0 a.out         00:00:12 44701 44694 44701
  1 a.out         00:00:12 44701 44694 44717
```

If **I_MPI_PIN** is not set to **0** in the PBS script, then **mbind.x** prints out identical placement results, as in the case where **I_MPI_PIN** is set to **0** but the **ps** command shows that some OpenMP threads "step on" one another.

```
r215i2n11% ps -C hybrid_intelmpi -L -opsr,comm,time,pid,ppid,lwp
PSR COMMAND      TIME    PID  PPID  LWP
  3 a.out         00:00:12 44185 44182 44185
  3 a.out         00:00:12 44185 44182 44198
  6 a.out         00:00:19 44186 44183 44186
  7 a.out         00:00:12 44186 44183 44202
```

```
9 a.out    00:00:12 44187 44184 44187
9 a.out    00:00:12 44187 44184 44200
0 a.out    00:00:19 44188 44181 44188
1 a.out    00:00:12 44188 44181 44204
```

The **mbind** utility was created by NAS staff member Henry Jin.

Article ID: 288

Last updated: 08 Aug, 2012

Computing at NAS -> Best Practices -> Process Pinning -> Using the mbind Tool for Pinning

<http://www.nas.nasa.gov/hecc/support/kb/entry/288/?ajax=1>