ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

High Performance Computing for Science and Engineering II
Spring semester 2015

M. Troyer
ETH Zürich, HIT G 31.8
CH-8093 Zürich

Hybrid MPI on Euler

Load the MPI module

For hybrid MPI jobs we cannot use the default Open MPI installation, but we have to switch to the MPICH implementation. To load the corresponding model you need to:

```
module load new
module load mvapich2/2.0
```

Compile

Once the modules are loaded, compilation works as usual with the `mpicxx` compiler wrapper (remember that for OpenMP support the `-fopenmp` flag is required).

```
mpicxx -fopenmp main.cpp -o prog
```

Environment variables

You already know the OpenMP environment variable for the total number of threads `OMP_NUM_THREADS`. For threading support to be enabled, the MPICH library requires CPU affinity to be turned off (see section 6.16 in the MPICH user manual), which is done by setting the variable `MV2_ENABLE_AFFINITY`.

Altogether your environment should look something like:

```
export MV2_ENABLE_AFFINITY=0
export OMP_NUM_THREADS=<number of threads>
```

where `<number of threads>` is the number of threads that you want for each process.

Job launching

Assume you want to launch the program `./prog` using 4 MPI processes having 2 OpenMP threads each, then you have to submit a request for $2 \times 4 = 8$ cores:

```
export OMP_NUM_THREADS=2
bsub -n 8 mpirun -np 4 ./prog
```

When you want to operate on two or more nodes, you have to specify additional options to both the scheduling system and the MPI launcher `mpirun`.

Suppose you want to run 2 MPI processes on each node, then you need to specify

1. Scheduling system: `-n 8 -R 'span[ptile=4]'`, meaning that you want a total of 8 cores split on multiples nodes having $2 \times 2 = 4$ cores each.
2. MPI launcher: `-ppn=2`, meaning 2 MPI processes per node.

Altogether the launch command will look like:

```
export OMP_NUM_THREADS=2
bsub -n 8 -R 'span[ptile=4]' mpirun -np 4 -ppn 2 ./prog
```

CPU affinity

Unfortunately the previous launch command will not show any speed up for your application. The reason is that the MPICH library has some problem with CPU binding, i.e. all processes will bind their threads to the same physical cores.

By using a debugging *hello world* program we obtain:

```
Process 0 on e2110 out of 4. Thread 0 of 2 running on CPU18.
Process 0 on e2110 out of 4. Thread 1 of 2 running on CPU19.
Process 1 on e2110 out of 4. Thread 0 of 2 running on CPU18.
Process 1 on e2110 out of 4. Thread 1 of 2 running on CPU19.
Process 2 on e2112 out of 4. Thread 0 of 2 running on CPU0.
Process 2 on e2112 out of 4. Thread 1 of 2 running on CPU1.
Process 3 on e2112 out of 4. Thread 0 of 2 running on CPU0.
Process 3 on e2112 out of 4. Thread 1 of 2 running on CPU1.
```

Attached to this document you find a binding script `myrun` which will fix the problem. The script reads the CPU ids of the assigned resources and set a different CPU affinity for each MPI process, in order to avoid overlaps.

You just have to wrap your program with it, i.e.

```
export OMP_NUM_THREADS=2
bsub -n 8 -R 'span[ptile=4]' mpirun -np 4 -ppn 2 ./myrun ./
  prog
```

By running again the debugging program we obtain the correct binding:

```
Process 0 on e2074 out of 4. Thread 0 of 2 running on CPU0.
Process 0 on e2074 out of 4. Thread 1 of 2 running on CPU5.
Process 1 on e2074 out of 4. Thread 0 of 2 running on CPU12.
Process 1 on e2074 out of 4. Thread 1 of 2 running on CPU13.
Process 2 on e2076 out of 4. Thread 0 of 2 running on CPU0.
Process 2 on e2076 out of 4. Thread 1 of 2 running on CPU1.
Process 3 on e2076 out of 4. Thread 0 of 2 running on CPU2.
Process 3 on e2076 out of 4. Thread 1 of 2 running on CPU3.
```