Symmetric Computing

John Cazes

Texas Advanced Computing Center

Symmetric Computing

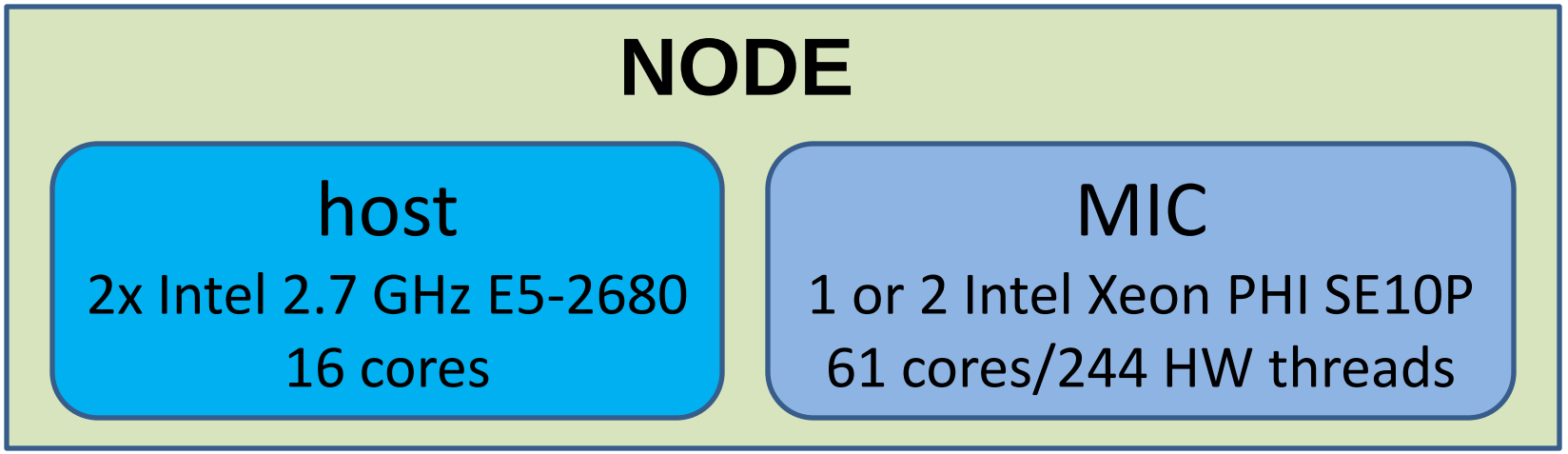
Run MPI tasks on both MIC and host and across nodes

- Also called “heterogeneous computing”
- Two executables are required:
  - CPU
  - MIC
- Currently only works with Intel MPI
- MVAPICH2 support coming

Definition of a Node

A “node” contains a host component and a MIC component

- host – refers to the Sandy Bridge component
- MIC – refers to one or two Intel Xeon Phi co-processor cards



Environment variables for MIC

By default, environment variables are “inherited” by all MPI tasks

Since the MIC has a different architecture, several environment variables must be modified

- OMP_NUM_THREADS – # of threads on MIC
- LD_LIBRARY_PATH – must point to MIC libraries
- I_MPI_PIN_MODE – controls the placement of tasks
- KMP_AFFINITY – controls thread binding

Symmetric run on 1 Node

16 tasks on host

```
mpiexec.hydra \
      –n 16 –host localhost ./host.exe \
    : –env OMP_NUM_THREADS 30 \
      –env LD_LIBRARY_PATH $MIC_LD_LIBRARY_PATH \
      –env I_MPI_PIN_MODE mpd \
      –env KMP_AFFINITY balanced \
      –n   4 –host mic0 ./mic.exe
```

4 tasks on mic0

Environment variables for MIC tasks

Steps to create a symmetric run

1. Compile a host executable and a MIC executable:
   - mpicc –openmp –o my_exe.cpu my_code.c
   - mpicc –openmp –mmic –o my_exe.mic my_code.c
2. Determine the appropriate number of tasks and threads for both MIC and host:
   - 16 tasks/host – 1 thread/MPI task
   - 4 tasks/MIC – 30 threads/MPI task

Steps to create a symmetric run

3. Create a batch script to distribute the job

```
#!/bin/bash
#----------------------------------------------------
# symmetric_example.slurm
# Generic symmetric script - MPI + OpenMP
#----------------------------------------------------
#SBATCH -J myjob           # Job name
#SBATCH -o myjob.%j.out    # stdout; %j expands to jobid
#SBATCH -e myjob.%j.err    # stderr; skip to combine stdout and stderr
#SBATCH -p development     # queue
#SBATCH -N 2               # Number of nodes, not cores (16 cores/node)
#SBATCH -n 32              # Total number of MPI tasks (if omitted, n=N)
#SBATCH -t 00:30:00        # max time
#SBATCH -A TG-01234        # necessary if you have multiple project accounts

export MIC_PPN=4
export MIC_OMP_NUM_THREADS=30

ibrun.symm -m ./my_exe.mic -c ./my_exe.cpu
```

Steps to create a symmetric run

1. Compile a host executable and a MIC executable
2. Determine the appropriate number of tasks and threads for both MIC and host
3. Create the batch script
4. Submit the batch script
   - sbatch symmetric.slurm

Symmetric launcher – ibrun.symm

Usage:

ibrun.symm –m ./<mic_executable> -c ./<cpu_executable>

- Analog of ibrun for symmetric execution
- # of MIC tasks and threads are controlled by env variables

MIC_PPN = <# of MPI tasks/MIC card>

MIC_OMP_NUM_THREADS = <# of OMP threads/MIC MPI task>

MIC_MY_NSLOTS = < Total # of MIC MPI tasks >

Symmetric launcher

- # of host tasks determined by batch script (same as regular ibrun)
- ibrun.symm does not support “-o” and “-n” flags
- Command line arguments may be passed with quotes

```
ibrun.symm –m “./my_exe.mic args” –c “./my_exe.cpu args”
```

Symmetric launcher

- If the executables require redirection or complicated command lines, a simple shell script may be used:

run_mic.sh	run_cpu.sh
`#!/bin/sh` `a.out.mic <args> < inputfile`	`#!/bin/sh` `a.out.host <args> < inputfile`

```
ibrun.symm –m ./run_mic.sh –c ./run_cpu.sh
```

Note: The bash, csh, and tcsh shells are not available on MIC. So, the MIC script must begin with "`#!/bin/sh`"

Symmetric Launcher Example

```
…
#SLURM -N 4 -n 32
export OMP_NUM_THREADS=2
export MIC_OMP_NUM_THREADS=60
export MIC_PPN=2
ibrun.symm -m a.out.mic -c a.out.cpu
```

The MPI tasks will be allocated in consecutive order by node (CPU tasks first, then MIC tasks). For example, the task allocation described by the above script snippet will be:

NODE 1	8 host tasks (0 - 7)	2 MIC tasks (8 - 9)
NODE 2	8 host tasks (10 - 17)	2 MIC tasks (18 - 19)
NODE 3	8 host tasks (20 - 27)	2 MIC tasks (28 - 39)
NODE 4	8 host tasks (30 - 37)	2 MIC tasks (38 - 49)

Task Binding

When using IMPI, the process binding mechanism may be controlled with the following environment variable:

- I_MPI_PIN_MODE=<pinmode>

mpd	mpd daemon pins MPI processes at startup (Best performance for MIC)
pm	Hydra launcher pins MPI processes at startup (Doesn't appear to work on MIC)
lib	MPI library pins processes BUT this does not guarantee colocation of CPU and memory (Default)

I_MPI_PIN_MODE=mpd (Default for ibrun.symm)

Task Binding

You can also lay out tasks across the local cores

- Explicitly: I_MPI_PIN_PROCESSOR_LIST=<proclist>
  - export I_MPI_PIN_PROCESSOR_LIST=1-7,9-15
- Grouped: I_MPI_PIN_PROCESSOR_LIST=<map>

bunch	The processes are mapped as closely as possible on the socket
scatter	The processes are mapped as remotely as possible to avoid sharing common resources: caches, cores
spread	The processes are mapped consecutively with the possibility to not share common resources

Task Binding

Be careful when using MIC and host

- MIC – 244 H/W threads and 1 socket
- Host – 16 cores and 2 sockets

To set I_MPI_PROCESOR_LIST for MIC only use the MIC_ prefix, e.g.

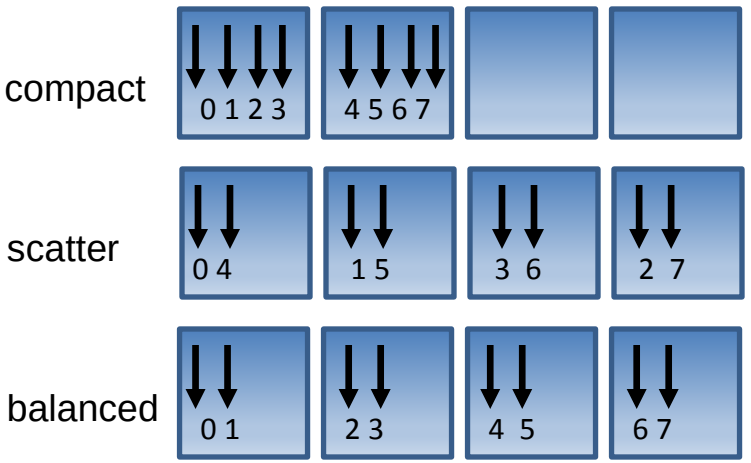
```
export MIC_I_MPI_PROCESSOR_LIST=1,61,121,181
```

Thread Placement

Thread placement may be controlled with the following environment variable

- KMP_AFFINITY=<type>

compact	pack threads close to each other
scatter	Round-Robin threads to cores
balanced	keep OMP thread ids consecutive (MIC only)
explicit	use the proclist modifier to pin threads
none	does not pin threads



KMP_AFFINITY=balanced (Default for ibrun.symm)

Balance

- How to balance the code?

	Sandy Bridge	**Xeon Phi**
Memory	32 GB	8 GB
Cores	16	61
Clock Speed	2.7 GHz	1.1 GHz
Memory Bandwidth	51.2 GB/s(x2)	352 GB/s
Vector Length	4 DP words	8 DP words

Balance

Example: Memory balance

Balance memory use and performance by using a different # of tasks/threads on host and MIC

Host
16 tasks/1 thread/task
2GB/task

Xeon PHI
4 tasks/60 threads/task
2GB/task

Balance

Example: Performance balance

Balance performance by tuning the # of tasks and threads on host and MIC

Host
16 tasks/1 thread/task
2GB/task

Xeon PHI
? tasks/? threads/task
?GB/task

MPI with Offload Sections

ADVANTAGES

- Offload Sections may easily be added to MPI/OpenMP codes with directives
- Intel compiler will automatically detect and compile offloaded sections

CAVEATS

- However, there may be no MPI calls within offload sections
- Each host task will spawn an offload section

Exercises

- Exercise 1
  - Run natively on the MIC using mpiexec.hydra
- Exercise 2
  - Run in a symmetric mode using MIC and host
- Exercise 3
  - Run an MPI code with offload