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

**NODE**

**host**

2x Intel 2.7 GHz E5-2680
16 cores

**MIC**

1 or 2 Intel Xeon PHI SE10P
61 cores/244 HW threads
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

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

- 16 tasks on host
- 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.slurm
# Generic symmetric script – MPI + OpenMP
#----------------------------------------------------
#SBATCH -J symmetric # Job name
#SBATCH -o symmetric.%j.out # stdout; %j expands to jobid
#SBATCH -e symmetric.%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 projects

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:

<table>
<thead>
<tr>
<th>run_mic.sh</th>
<th>run_cpu.sh</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/sh</code></td>
<td><code>#!/bin/sh</code></td>
</tr>
<tr>
<td><code>a.out.mic &lt;args&gt; &lt; inputfile</code></td>
<td><code>a.out.host &lt;args&gt; &lt; inputfile</code></td>
</tr>
</tbody>
</table>

```bash
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:

<table>
<thead>
<tr>
<th>NODE</th>
<th>8 host tasks</th>
<th>2 MIC tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE 1</td>
<td>0 - 7</td>
<td>8 - 9</td>
</tr>
<tr>
<td>NODE 2</td>
<td>10 - 17</td>
<td>18 - 19</td>
</tr>
<tr>
<td>NODE 3</td>
<td>20 - 27</td>
<td>28 - 29</td>
</tr>
<tr>
<td>NODE 4</td>
<td>30 - 37</td>
<td>38 - 39</td>
</tr>
</tbody>
</table>
## Task Binding

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

- **I_MPI_PIN_MODE=<pinmode>**

<table>
<thead>
<tr>
<th>pinmode</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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>`

<table>
<thead>
<tr>
<th>bunch</th>
<th>The processes are mapped as closely as possible on the socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>scatter</td>
<td>The processes are mapped as remotely as possible to avoid sharing common resources: caches, cores</td>
</tr>
<tr>
<td>spread</td>
<td>The processes are mapped consecutively with the possibility to not share common resources</td>
</tr>
</tbody>
</table>
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_PROCESSOR_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>**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>compact</td>
<td>Pack threads close to each other</td>
<td>![compact example]</td>
</tr>
<tr>
<td>scatter</td>
<td>Round-Robin threads to cores</td>
<td>![scatter example]</td>
</tr>
<tr>
<td>balanced</td>
<td>Keep OMP thread ids consecutive (MIC only)</td>
<td>![balanced example]</td>
</tr>
<tr>
<td>explicit</td>
<td>Use the proclist modifier to pin threads</td>
<td>![explicit example]</td>
</tr>
<tr>
<td>none</td>
<td>Does not pin threads</td>
<td>![none example]</td>
</tr>
</tbody>
</table>

**KMP_AFFINITY=balanced** (Default for ibrun.symm)
Balance

- How to balance the code?

<table>
<thead>
<tr>
<th></th>
<th>Sandy Bridge</th>
<th>Xeon Phi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>32 GB</td>
<td>8 GB</td>
</tr>
<tr>
<td>Cores</td>
<td>16</td>
<td>61</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>2.7 GHz</td>
<td>1.1 GHz</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>51.2 GB/s(x2)</td>
<td>352 GB/s</td>
</tr>
<tr>
<td>Vector Length</td>
<td>4 DP words</td>
<td>8 DP words</td>
</tr>
</tbody>
</table>
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
? tasks/? threads/task
?GB/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