Programming on the Intel MIC (Many Integrated Core) Architecture:
Part 2 - How to run MPI applications

Chongxun (Doris) Pan
doris.pan@nasa.gov
June 13, 2013
Agenda of the previous MIC talk (Part 1)

- Discover SCU8 augmentation
- What is MIC?
- MIC Programming Considerations
- Offload vs. Native
- Demo

http://www.nccs.nasa.gov/list_brown_bags.html
Click “An introduction to MIC programming models”
Today’s Agenda – MIC Talk Part 2

- First -- a few words about running jobs on the Sandy Bridge nodes
- Intel MPI on MIC
  - MPI+Offload Model
  - Native Model
  - Symmetric Model
- User Environment Variables
- General Performance Guidelines
- Demo
Discover SCU8/9 Sandy Bridge Nodes

- The SNB processor family features -
  Intel Advanced Vector eXtensions
- AVX is a **256-bit** instruction set extension
- SSE (Streaming 128-bit SIMD Extensions) on Westmere processors
- While MIC has 512-bit instructions (hence higher peak FLOPS with better power efficiency)
- Some SNB nodes are available in the “general” queue
- To request SNB nodes:
  #PBS –l select=N:ncpus=16:mpiprocs=m (recommended)
  Or
  #PBS –l select=N:ncpus=12:mpiprocs=m:proc=sand
Discover SCU8 “Sandy Bridge” User Changes

Compiler flags to take advantage of Intel AVX (for Intel compilers 11.1 and up ONLY)

-xavx:
- Generate an optimized executable that runs on the Sandy Bridge processors ONLY

-axavx –xsse4.2:
- Generate an executable that runs on any SSE4.2 compatible processors but with additional specialized code path optimized for AVX compatible processors (i.e., run on all Discover processors)
- Application performance is affected slightly compared to with “-xavx” due to the run-time checks needed to determine which code path to use
Spectrum of Programming Models

- **Multi-Core Centric**
  - Xeon
  - Multi-Core Hosted
    - General purpose serial and parallel computing
  - Offload
    - Codes with highly-parallel phases
- **Symmetric**
  - Codes with balanced needs
- **Many Core Hosted**
  - MIC
  - Many Core Hosted
    - Highly-parallel codes

Range of models to meet application needs

NASA Center for Climate Simulation
Intel MPI Support for Xeon Phi Coprocessors

Three Programming Models:

- **Using MPI and Offload together**
  - MPI ranks on Xeon only

- **Native**
  - MPI ranks on Xeon Phi only

- **Symmetric**
  - MPI ranks on both Xeon and Xeon Phi
Prerequisites on Discover

- Have to use Intel 13 compiler and Intel MPI 4.1, e.g.
  - `module load comp/intel-13.1.2.183`
  - `module load mpi/impi-4.1.0.024-test`

- MPI libraries accessible also from Phi
  - `/usr/local/intel/mpi/4.1.0.024/intel64/{bin,etc,include,lib}`
  - `/usr/local/intel/mpi/4.1.0.024/mic/{bin,etc,include,lib}`

- Set the Intel MPI environment
  - `source /opt/intel/impi/4.1.0.024/intel64/bin/mpivars.sh`

- Only one user environment setup required for $PATH and $LD_LIBRARY_PATH, serves both architectures
  - `setenv PATH ${{PATH}:~/opt/intel/mic/bin`}

Refer to `/home/cpan2/.login_scu8`
NFS Mounting on Host and MIC

- NFS File Mounts configured on the MIC cards. It is useful for handling input/output of large data sets.
- There is no permanent file system on the cards. **Files you copied to non-NFS mounted directories on the cards will be cleaned up once you exit the PBS job.**
MPI + Offload

- All MPI communications occur between Xeons only

- Offload used to accelerate MPI ranks. Programmers designates (OpenMP, pthreads, TBB, or Cilk Plus) code sessions to run on Phi using offload directives.

- Calling MPI functions within an offload region is NOT allowed

- No direct file system access needed on Xeon Phi
Compile the code with offload directives

- The same as with MPI-OpenMP hybrid applications
- offload build is the default compiler option with Intel13 compiler

```bash
mpiifort --openmp test.f -o test.offload
```
To request explicitly no offloading:

```bash
mpiifort --openmp --no-offload test.f -o test.nooffload
```

Launch your application:

```bash
mpiexec.hydra -f $PBS_NODEFILE -perhost 1 -np 2 ./test.offload
```

“mpirun” is usually a script written to integrate with the PBS. For module mpi/impi-4.1.0.024-test, we have not created the “mpirun” script. Simply use mpiexec.hydra instead.
MPI Offload in a Script Example

```bash
#!/usr/bin/csh

# xsub -l -V -l select=2:ncpus=16,walltime=1:00:00 -q test
module purge
module load comp/intel-13.1.2.183
module load mpi/impi-4.1.0.024-test
source /usr/local/intel/Composer/composer_xe_2013.4.183/bin compilervars.csh intel64
source /usr/local/intel/mpi/4.1.0.024/intel64/bin/mpivars.csh
setenv MIC_ENV_PREFIX MIC
setenv MIC_OMP_NUM_THREADS 236
setenv OMP_NUM_THREADS 16
setenv MIC_OMP_STACKSIZE 2M
setenv MIC_USE_2MB_BUFFERS 64K
setenv OFFLOAD_INIT on_start
setenv I_MPI_PIN_DOMAIN auto
setenv KMP_AFFINITY compact
setenv MIC_KMP_AFFINITY "granularity=thread,balanced"
setenv OFFLOAD_REPORT 2
mpiifort -openmp -align array64byte -offload-option,mic,compiler,"-O3 -vec-report3" -o test.offload
test.f90
mpiexec.hydra -f $PBS_NODEFILE -perhost 1 -np 2 ../test.offload
```
Avoid Offload Resource Conflicts

Coordinate coprocessor resource usage among the MPI ranks. Three methods:

1. Only running one MPI rank per host, so there is no chance of multiple ranks offload to the same Phi coprocessor
2. Heterogeneous: Running multiple MPI ranks per host but arranging processors to allow only single rank offload to the same Phi
3. Explicit Pinning: Setting the pinning on a per-process basis to allow control of where each thread is offloaded.

borg01x045 $ cat $PBS_NODEFILE
borg01x045.prv.cube
borg01x046.prv.cube
borg01x045 $ mpiexec.hydra -f $PBS_NODEFILE -perhost 2 \
    -genv MIC_OMP_NUM_THREADS 118 \
    -env MIC_KMP_AFFINITY "granularity=fine,proclist=[1-118],explicit" -n 1 ./jacobi : \
    -env MIC_KMP_AFFINITY "granularity=fine,proclist=[119-236],explicit" -n 1 ./jacobi : \
    -env MIC_KMP_AFFINITY "granularity=fine,proclist=[1-118],explicit" -n 1 ./jacobi : \
    -env MIC_KMP_AFFINITY "granularity=fine,proclist=[119-236],explicit" -n 1 ./jacobi
Create a configuration file for convenience

Borg01x045 $ mpiexec.hydra -f $PBS_NODEFILE -perhost 2 \
   -genv MIC_OMP_NUM_THREADS 118 \
   -env MIC_KMP_AFFINITY "granularity=fine,proclist=[1-118],explicit" -n 1 ./jacobi : \
   -env MIC_KMP_AFFINITY "granularity=fine,proclist=[119-236],explicit" -n 1 ./jacobi : \
   -env MIC_KMP_AFFINITY "granularity=fine,proclist=[1-118],explicit" -n 1 ./jacobi : \
   -env MIC_KMP_AFFINITY "granularity=fine,proclist=[119-236],explicit" -n 1 ./jacobi

Borg01x045 $ cat conf_file
   -genv MIC_OMP_NUM_THREADS 118 -env MIC_KMP_AFFINITY "granularity=fine,proclist=[1-118],explicit" -n 1 ./jacobi -env MIC_KMP_AFFINITY "granularity=fine,proclist=[119-236],explicit" -n 1 ./jacobi -env MIC_KMP_AFFINITY "granularity=fine,proclist=[1-118],explicit" -n 1 ./jacobi -env MIC_KMP_AFFINITY "granularity=fine,proclist=[119-236],explicit" -n 1 ./jacobi

Borg01x045 $ mpiexec.hydra -f $PBS_NODEFILE -perhost 2 -configfile conf_file

Using `\' as line continuation is NOT supported by the configure file
Native Model -- How to Compile and Run

- MPI ranks on the Phi coprocessors only
- MPI messages into/out of the Phi
- Threading possible

Compile the code for Phi:
```bash
mpiifort -mmic test.f -o test.mic
```

Copy the executable to the Phi, Or, make sure the executable located in the NFS-shared directory

```
borg01x045 $ cd /mnt/micfs/discover/home/cpan2/MIC
borg01x045 $ mpiifort -mmic test.f -o test.mic
borg01x045 $ ssh borg01x045-mic0
borg01x045-mic0 $ cd /discover/home/cpan2/MIC
borg01x045-mic0 $ ls
test.mic ....
```

If password is asked when you ssh to the host-mic0, your key is not set up correctly. Contact support@nccs.nasa.gov.
Two Ways to Launch Application Natively

1. From the Phi:
   - ssh to the Phi
   - Set appropriate environment on the Phi:
     
     ```bash
     export PATH=$PATH:/usr/local/intel/mpi/4.1.0.024/mic/bin
     export LD_LIBRARY_PATH=/usr/local/intel/Composer/composer_xe_2013.4.183/compiler/lib/mic
     export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/intel/impi/4.1.0.024/mic/lib:/discover/home/cpan2/lib
     ```

     Note: other libraries may be needed for your applications but are not mounted on the cards, in which case you can copy them over and point the LD_LIBRARY_PATH to it.

   - Create a hostfile
     ```bash
     cd /discover/home/cpan2/MIC; cat hostfile.mic
     borg01x045-mic0
     borg01x046-mic0
     ```

   - Run
     ```bash
     mpiexec.hydra –f hostfile –perhost 60 –np 120 ./test.mic
     ```
Two Ways to Launch your application natively

From the Host:

- Create a hostfile
  ```bash
cd /mnt/micfs/discover/home/cpan2/MIC; cat hostfile.mic
borg01x045-mic0
borg01x046-mic0
```
- Let the library know you are using coprocessor(s) for your MPI job
  ```bash
  setenv I_MPI_MIC 1
  ```
- Run
  ```bash
  mpiexec.hydra –f hostfile –perhost 60 –np 120 /discover/home/cpan2/MIC/test.mic
  ```

**Note the PATH of the executable is accessible by Phi!**

Or you can set I_MPI_MIC_PREFIX (be careful) if all Phi executables are located in one path.

```bash
setenv I_MPI_MIC_PREFIX /discover/home/cpan2/MIC/
mpiexec.hydra –f hostfile –perhost 60 –np 120 test.mic
```
Symmetric Model – How to Compile and Run

- MPI ranks on both the hosts and Phi(s)
- MPI messages into/out of the hosts and Phi(s)
- Threading possible

- Compile the code for host
  mpiifort test.f –o test

- Compile the code for Phi
  mpiifort –mmic test.f –o test.mic

- Copy the executable to the Phi, Or, make sure the executable located in the NFS-shared directory

- Tell the MPI library to add a postfix to the Phi executable
  setenv I_MPI_MIC_POSTFIX .mic

- Tell the MPI library to add a prefix to the Phi executable
  setenv I_MPI_MIC_PREFIX /discover/home/cpan2/MIC/
Tell the library you are using coprocessors for MPI job

```
setenv I_MPI_MIC 1
```

Create a hostfile

```
cat hostfile
borg01w001
borg01w001-mic0
borg01w002
borg01w002-mic0
```

Launch the application from the host:

```
mpiexec.hydra -f hostfile -perhost 10 -np 40 ./test
```

When the test executable is run on the Phi, the mpirun script will automatically add the postfix and prefix for the Phi executable

```
mpiexec.hydra --env I_MPI_DEBUG 3 -host borg01w001 -n 16 -env OMP_NUM_THREADS 1 ./test : -host borg01w001-mic0 -n 59 -env MIC_OMP_NUM_THREADS 4 /discover/home/cpan2/MIC/test.mic
```
Load Balancing with Symmetric Model

- **Situation**
  - Host and MIC computation performance are different
  - Host and MIC internal communication speed is different

- **MPI in symmetric mode is like running on a heterogeneous cluster**

- **Solutions**
  - **Approach 1**: Adapt MPI mapping of the hybrid code
    - Example: m1 processes and m2 threads per host, n1 process and n2 threads per MIC card
  - **Approach 2**: Change code internal mapping of workload to MPI processes
    - Example: uneven split of calculation grid for MPI processes on host vs. MIC

- **Analyze and improving MPI/thread load balance of application with Trace Analyzer and Collector (ITAC)**
Define `I_MPI_PIN_DOMAIN` to map cpus into non-overlapping domains

- Mapping rule: 1 MPI process per domain
- Pin OpenMP threads inside the domain with `KMP_AFFINITY`

`I_MPI_PIN_DOMAIN = <size>[:<layout>]`

- `<size>` =
  - `omp`: Adjust to OMP_NUM_THREADS
  - `auto`: `#total CPUs / #MPI procs`
  - `<n>`: Number

- `<layout>` =
  - `platform`: According to BIOS numbering
  - `compact`: Close to each other -- Default
  - `scatter`: Far away from each other
Avoid using the last physical core of the Phi, which is for kernel & low level housekeeping.

Example: on a 60-core Phi coprocessor, max threads usable is 236.

```bash
setenv MIC_KMP_AFFINITY "explicit,granularity=fine,
proclist=[1-236:1]"
```

Easier to use "compact", "scatter", or "balanced" *(new, coprocessor only)*. "balanced" uses all cores like "scatter", but keeps adjacent threads on the same core.

Different env-variables on host and Phi:

```bash
setenv MIC_ENV_PREFIX MIC
setenv OMP_NUM_THREADS 16
setenv MIC_OMP_NUM_THREADS 236
setenv KMP_AFFINITY "granularity=fine,compact"
setenv MIC_KMP_AFFINITY "granularity=fine,balanced"
```
Selecting Network Fabrics

- The Intel MPI dynamically select the most appropriate fabric for communications
- Use `I_MPI_FABRICS` (replacing `I_MPI_DEVICE`) to select a different communication device explicitly
- The best fabric is usually based on Infiniband (`dapl` and `ofa`) for inter node communication and shared memory for intra node

```
```

Available for Phi (`shm:dapl is the default`):

- `shm`, `tcp`, `ofa`, `dapl` (RMDA-enabled device)
- Availability checked in the order `shm:dapl`, `shm:ofa`, `shm:tcp` (intra:inter)

Recommend

```
setenv I_MPI_FABRICS shm:ofa
```
For **master thread** for the offload: the default stack limit is 12MB

- In offloaded functions, stack is used for local or automatic arrays and compiler temporaries
- To increase limit:
  
  ```
  setenv MIC_STACKSIZE=100M  
  ```
  (no need to set MIC_ENV_PREFIX)

For **other threads**: the default stack limit is 4MB

- Space is only needed for those local or automatic arrays for which each thread keeps private for thread safety
- To increase limit:
  
  ```
  setenv MIC_OMP_STACKSIZE=10M  
  ```
  
  ```
  setenv MIC_ENV_PREFIX=MIC  
  ```

**Typical error message if stacksize limit is reached:**

- offload error: process on the device 0 was terminated by SEGFAULT
- offload error: EventWait failed with error COI_PROCESS_DIED
Environment Variables to Control Offload

- **OFFLOAD_REPORT=1 | 2**
  Turn on/off offload reporting at runtime

- **OFFLOAD_INIT=on_start | on_offload**
  Specify on offload runtime when it should initialize MIC devices

- **MIC_STACKSIZE=100M**
  Stack size for the master thread in offload region. Change the default if allocating large arrays on the stack. Default 12M.

- **MIC_USE_2MB_BUFFERS=100K**
  Offloaded data larger than the “size” will use the 2MB pages to maximize data transfer rate. Default is not to use the 2MB pages at all

- **MIC_LD_LIBRARY_PATH**
  The path where shared libraries needed by the MIC offloaded code

The setting of **MIC_ENV_PREFIX** has no effect on the fixed MIC_* env variables, MIC_USE_2MB_BUFFERS, MIC_STACKSIZE and MIC_LD_LIBRARY_PATH. Those names are fixed.
How to involve Phi in your applications?

<table>
<thead>
<tr>
<th>Offload Model</th>
<th>Native / Symmetric Models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros:</strong></td>
<td><strong>Pros:</strong></td>
</tr>
<tr>
<td>• Better serial processing</td>
<td>• Easy to run. Almost no code change</td>
</tr>
<tr>
<td>• More Memory</td>
<td>• Better for code that does not have well-identified hot spots than can be offloaded without substantial data transfer overhead</td>
</tr>
<tr>
<td>• Better file access and I/O</td>
<td><strong>Cons:</strong></td>
</tr>
<tr>
<td><strong>Cons:</strong></td>
<td><strong>Cons:</strong></td>
</tr>
<tr>
<td>• More program effort to add offload directives and to tune offload performance</td>
<td>• Constraints in memory footprint and I/O</td>
</tr>
<tr>
<td>• Less long-term benefit for those targeting future MIC processors</td>
<td>• workload imbalance</td>
</tr>
</tbody>
</table>
Development Options

- **Multi-Core Hosted**
  General serial and parallel computing

- **Offload**
  Code with highly-parallel phases

- **Symmetric**
  Codes with balanced needs

- **Many Core Hosted**
  Highly-parallel codes

**Threading Options**
- Intel® Math Kernel Library
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
- OpenMP*
- Pthreads*

**Vector Options**
- Intel® Math Kernel Library
- Array Notation: Intel® Cilk™ Plus
- Auto vectorization
- Semi-auto vectorization: #pragma (vector, ivdep, simd)
- OpenCL*
- C/C++ Vector Classes (F32vec16, F64vec8)

**Ease of use**

**Depth**

**Fine control**
Regardless of programming model of your choice, **FOUR things** you must do to attain maximum performance on MIC

1. **Analyze and Characterize your applications**
   - Hot spots – focus tuning efforts on hot spots
   - Load balance, serialization, and overhead
   - Vectorization – whether hot loops are vectorized

2. **Optimize for SIMD**
   - Choose SIMD-friendly algorithms
     - Loop interchange: remove dependencies between loop iterations
     - Minimize gather/scatter and branch misprediction
   - Vectorize inner loops with `!dir$ simd` or the like
   - Vectorize outer loops using Intel Cilk Plus array notation or transform outer loops
3. Exploit thread and task parallelism
   - Balance MPI and OMP thread parallelism for Host and Phi
   - !$omp do collapse (n) to increase thread parallelism
   - Set KMP_AFFINITY to avoid resource conflicts

4. Optimize for memory access
   - Data alignment to 64 byte boundary
   - !$dir$ unroll and !$dir$ unroll_fuse to minimize the required number of loop iterations, while reducing the frequency of cache misses
   - “Blocking” data structure (loop tiling) to maximize time data spends in cache
   - Minimize gather/scatter operations by converting arrays of structures (AOS) to structures of arrays (SOA)
   - Prefetching
   - Large Page Support
Lots of documentations and tutorial videos are offered by Intel

http://software.intel.com/mic-developer

More brownbag tutorials to come …
- Intel MPI on MIC
- Running WRF on MIC
- Language Extensions for Offload
- Maximize Vectorization
- Performance analysis with VTune Amplifier and Tracer Analyzer
- Performance tuning topics for MIC