Logging in and Running Applications

Gaining Access to Beacon

Begin by following the instructions in "login_instructions_beacon.pdf" to configure your One Time password token (OTP) and gain access to Beacon. When choosing a PIN, make sure to choose something memorable, since you will need it every time that you log in to the system.

Launching Applications

Interactive Submission

For the duration of today's tutorial session, NICS have granted us a reservation of the entire Beacon system, enabling each attendee to reserve their own node in an interactive session.

Each of our hands-on sessions today should last about an hour. To request an interactive job for each session, pass the following PBS options to qsub:

```
qsub -I -A UT-NTNLEDU -l nodes=1,walltime=01:00:00
```

Batch Submission

Your user account will remain active for the rest of the week, but our system reservation will not. If you want to continue with the labs in your own time, then we recommend that you switch to batch submission.

To submit a non-interactive job, you must supply a job script of the form:

```
#!/bin/bash
#PBS -A UT-NTNLEDU
#PBS -l nodes=1,walltime=00:15:00
cd $PBS_O_WORKDIR
…
```

Those interested in learning more about Beacon and its queues are referred to the NICS website (https://www.nics.tennessee.edu/beacon).

Building Applications

We'll start by looking at a simple "Hello World!" program, which does nothing but identify whether it has been launched on a processor or coprocessor.

```
#include <stdio.h>

int main(int argc, char* argv[])
{
        #ifndef __MIC__
        printf("Hello world! This is the host.\n");
        #else
        printf("Hello world! This is the MIC.\n");
        #endif
}
```

To build for the processor:

```
icc -o hello hello.c
```

To build for the coprocessor:

```
icc -o hello_mic -mmic hello.c
```

Note that the only difference between the two build commands is the change of executable name and the addition of the "-mmic" flag.

Running Applications

Please make note of the hostname of the host node you are assigned by the scheduler (e.g. beacon002), since the hostname of your assigned mic card will be based on this (e.g. beacon002-mic0).

Running on the Processor

There are no "tricks" to running the application we just compiled on the processor:

```
[user@beaconXXX ~]$ ./hello
Hello world! This is the host.
```

Running on the Coprocessor

Since an Intel® Xeon Phi™ coprocessor runs its own OS, and is separated from the main system by a PCI-Express* bus, running the application on the coprocessor requires a little more work.

One option is to copy the binary across to the card using scp, and then launch it using ssh. Beacon supplies custom scripts ("micscp" and "micssh") to enable users to access the card without entering a password. Do this now:

```
[user@beaconXXX ~]$ micscp hello_mic beaconXXX-mic0:/User/hello_mic
hello_mic                                     100% 10KB 10.2KB/s 00:00
[user@beaconXXX ~]$ micssh beaconXXX-mic0 /User/hello_mic
Hello world! This is the MIC.
```

Intel also provide a utility called micnativeloadex, which will handle the copying and launching of the executable for you:

```
[user@beaconXXX ~]$ micnativeloadex hello_mic
Hello world! This is the MIC.
```

This utility is not recommended for use in production runs of native applications, as the micnativeloadex utility reserves one of the coprocessor cores for itself. Native applications that do not account for this (i.e. applications that assume all of the coprocessor cores will be available) are likely to see better performance when launched via SSH.

The example codes in these labs make no assumptions about the number of cores available, and therefore it is safe to use micnativeloadex.