Hands-on with Intel Xeon Phi

Intro to Stampede nodes:
Sandy Bridge CPUs + Xeon Phi Coprocessors (MICs)

Bill Barth
Kent Milfeld
Dan Stanzione

Lab I

- What you will learn
  - The lab introduces you to Stampede and the Xeon Phi coprocessor built into each Stampede node.
- What you will do:
  - Compile for Xeon Sandy Bridge (host) and Xeon Phi (MIC)
  - Submit a job
  - Inspect the queue
  - Submit an interactive job
  - Execute on the host and on the Phi

Lab 1

Stampede Orientation

Part 0 – Grab the Lab Files

- Login to Stampede

```
$ ssh <username>@stampede.tacc.utexas.edu
```

- We use modules (list the compiler and MPI versions with the list option)

```
$ module list
```

- Untar the file mic_intro.tar file (in ~train00) into your directory:

```
$ tar xvf ~train00/mic_intro.tar
```

- Move into the newly created lab1 directory:

```
$ cd intro   # LAB1 directory
```

Part 1 – Host Batch Job (sbatch)

- Compile the mpipi program:

```
$ mpicc mpipi.c -o mpipi
```

- Look at the SLURM batch script (2 nodes, 5 minutes, development queue):

```
$ cat job  # or vi, or emacs
```

- Launch the batch job

```
$ sbatch job
```

- Monitor the job's status (when done, command will return nothing):

```
$ squeue -u <username>
$ showq  | more
$ squeue | more      # details: node lists on jobs
```

- When job completes, take a look at results:

```
$ cat mpipi.oxxxxx  # "xxxxx" is the jobid
```

Part 2 – Host Interactive Session (idev)

- Use idev to launch a 1-node interactive session in the development queue

```
$ idev
```

- When session begins, compile hello.F90* from compute node:

```
c558-001$ ifort -openmp hello.F90 -o hello.cpu
```

- Run the code:

```
c558-001 $ ./hello.cpu     # you're on a compute node, not a login node
```

- Set OpenMP threads and try again

```
c558-001 $ export OMP_NUM_THREADS=4
c558-001 $ ./hello.cpu
```

*Note: the capital "F" in the suffix instructs the compiler to interpret the macros in the source code. We use "#ifdef __MIC__" in hello.F90. If the suffix were "f90" the compilation would require a "-cpp" flag. Peek in the code to see how __MIC__ is used.

Part 3 – Run MIC App from the Host

NOTE: When you try to execute a MIC binary on the host, it will automatically determine that it is a MIC binary, and execute it on the MIC.

- While on the compute node, recompile to produce "native MIC" code (compilers are not visible from the MIC):

```
c558-001 $ ifort -mmic -openmp hello.F90 -o hello.mic
```

- Launch the MIC code from the host:

```
c558-001 $ ./hello.mic    #OS knows to launch MIC binary on Phi
```

Note: the program reports 244 "processors" and 244 threads, because each MIC core has four hardware threads (61x4).

It may be efficient to run fewer threads.

- From the host, setup MIC Environment variables for the thread count, & try again:

```
c558-001 $ export MIC_ENV_PREFIX=MIC        #MIC environment vars
                                            #will have MIC_ prefix
c558-001 $ export MIC_OMP_NUM_THREADS=60    #Set OMP_NUM_THREADS
                                            #for MIC
c558-001 $ ./hello.mic
```

Part 4 – Visit the MIC

- Go to the MIC using ssh:

```
c558-001 $ ssh mic0     # the "zero" identifies the MIC card
                        # You will get a new prompt <dir> $.
```

- Move into the Lab 1 directory with explicit cd :

```
~ $ cd intro
```

- Run your MIC code:

```
~intro $ ./hello.mic
```

- Change the MIC's thread count and run code again (don't use "MIC" prefix):

```
~intro $ export OMP_NUM_THREADS=25
~intro $ ./hello.mic
```

- Return to host (compute node), and then end srun/idev session:

```
~intro $ exit    # or ^C to return to host
~intro $ exit    # or ^C to end idev session
```

Throughout the rest of the training we will use "$" for the prompt on the host and the MIC.