MIC Lab
Parallel Computing on Stampede

Aaron Birkland and Steve Lantz
Cornell Center for Advanced Computing

June 11 & 18, 2013

1 Interactive Launching

This exercise will walk through interactively launching applications on a single coprocessor on Stampede through SSH and the `micrun` launcher.

1. Log into Stampede and unpack the lab materials into your home directory if you haven't done so already.

   ```
   login$ cd
   login$ tar xvf ~tg459572/LABS/mic.tar
   ```

2. Change into the `mic/exercise1` directory

3. Start an Interactive session with one node:

   ```
   login$  srun -A TG-TRA120006 -p development -t 0:30:00  -n 1 --pty \
   /bin/bash -l
   ```

   - If you have only one Stampede account, you do not need the `-A` argument. It doesn't hurt to keep it, though
   - Please note there are two dashes before pty
   - The last option is the letter "l", not the number "1"

4. In the `exercise1` directory, you will find a C file `where.c`. It prints out the current hostname, as well as the content of the `MY_ENV` environment variable. Compile it for the host CPU, and the MIC coprocessor. You will end up with two executables. Note the `.mic` and `.cpu` convention. This is useful to avoid confusing the two.

   ```
   c557-804$ icc where.c -o where.cpu
   c557-804$ icc -mmic where.c -o where.mic
   ```

5. Run the host executable

```
c557-804$ ./where.cpu
Host c557-804.stampede.tacc.utexas.edu; MY_ENV=
```

6. As you can see, we did not set the `MY_ENV` environment variable. Set it and re-run.

```
c557-804$ export MY_ENV=42
c557-804$ ./where.cpu
Host c557-804.stampede.tacc.utexas.edu; MY_ENV=42
```

7. Now run the MIC executable. When we directly execute a MIC executable from the host, the `micrun` launcher is implicitly invoked to remotely run the executable on the coprocessor rather than the host.

```
c557-804$ ./where.mic
Host c557-804-mic0.stampede.tacc.utexas.edu; MY_ENV=
```

   Notice the hostname. The `mic0` part indicates that our program did truly run on the coprocessor. However, the environment variable is not defined there! This is because the `micrun` launcher only passes along variables that begin with the `MIC_` prefix.

8. Set the `MIC_MY_ENV` variable and re-run

```
c557-804$ export MIC_MY_ENV=mic42
c557-804$ ./where.mic
Host c557-804-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
```

9. Now, try to ssh directly into the MIC coprocessor and run the executable:

```
  c557-804$ ssh mic0
TACC Stampede System - MIC Co-Processor

[Access enabled for user apb18]
~ $ ./where.mic
-sh: ./where.mic: not found
```

10. Oops! When we ssh into the coprocessor, we end up in the top level of our home directory. `cd` into our lab directory and re-run.

```
~$ cd mic/exercise1
~/mic/exercise1 $ ./where.mic
Host c557-804-mic0.stampede.tacc.utexas.edu; MY_ENV=
```

Notice that the environment variable is undefined. Using pure ssh, environment variables are not propagated between the host and the coprocessor. We need to define it manually.

11. Set the `MY_ENV` environment variable and re-run.

```
~/mic/example1 $ export MY_ENV=mic42
~/mic/example1 $ ./where.mic
Host c557-804-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
```

12. When finished `exit` the coprocessor, then `exit` the compute node to terminate the interactive session and return to the login node.

2 Simple Symmetric MPI example

This exercise will explore symmetric execution using MPI processes on host and MIC coprocessors. We will use an MPI-aware version of the `where` program from exercise 1. Each process will announce its rank, and the value of `MY_ENV` environment variable. We will verify that we can control MPI process launching on both hosts and MIC coprocessors.

1. Unpack the lab materials into your home directory. If you've done the first two examples, there is no need to do this again.

```
login$ cd
login$ tar xvf ~tg459572/LABS/mic.tar
```

2. Change into the `mic/exercise2` directory.

3. Create two executables: one for the host CPU (`where_mpi.cpu`), and one for the MIC coprocessor (`where_mpi.mic`). Make sure to use the Intel MPI implementation (`impi`)

```
login$ module swap mvapich2 impi
login$ mpicc where_mpi.c -o where_mpi.cpu
login$ mpicc -mmic where_mpi.c -o where_mpi.mic
```

   - If `impi` is already loaded, then you may omit the `module swap` line. Check with `module list` if you are not sure.

4. Take a look at the `where.sh` batch submission script. As it stands, it will use `ibrun.symm` to launch two MPI processes on one MIC coprocessor, and no processes on the host.

```
login$  cat where.sh
#!/bin/bash

#SBATCH -t 00:05:00
```

```
#SBATCH -n 1
#SBATCH -p development
#SBATCH -A TG-TRA120006

echo "Running on MIC"
export MIC_PPN=2
export MY_ENV=42
ibrun.symm -m where_mpi.mic
```

Note that `ibrun.symm` takes `-m` and `-c` arguments to specify MIC and CPU executables, respectively. Since we just want to run on the MIC coprocessors initially, no `-c` argument is specified.

5. Run the `where.sh` script, and examine the output/ Remember, your slurm output file will be named differently:

```
login$ sbatch where.sh
login$ cat slurm-920435.out
  Running on MIC
TACC: Starting up job 920435
TACC: Starting parallel tasks...
  /opt/apps/intel13/impi/4.1.0.030/intel64/bin/mpiexec.hydra -iface br0
  -configfile /home1/01871/apb18/.slurm/config_file.920435.h5eb2YgW
Rank 1 of 2, host c559-301-mic0.stampede.tacc.utexas.edu; MY_ENV=42
Rank 0 of 2, host c559-301-mic0.stampede.tacc.utexas.edu; MY_ENV=42

TACC: Shutdown complete. Exiting
```

Notice that, unlike the previous example, when we define define a `MY_ENV` environment variable, it *is* visible to the processes running on the MIC. However, specifying a `MIC_` prefixed version of a variable will override the non-prefixed one for processes running on the coprocessor. Let's verify this.

6. Edit the `where.sh` script to *add* the line `export MIC_MY_ENV=mic42`, and re-execute.

```
#!/bin/bash

#SBATCH -t 00:05:00
#SBATCH -n 1
#SBATCH -p development
#SBATCH -A TG-TRA120006

echo "Running on MIC"
export MIC_PPN=2
export MIC_MY_ENV=mic42
export MY_ENV=42
ibrun.symm -m where_mpi.mic
```

```
login$ sbatch where.sh
login$ cat slurm-920488.out
  Running on MIC
TACC: Starting up job 920488
TACC: Starting parallel tasks...
  /opt/apps/intel13/impi/4.1.0.030/intel64/bin/mpiexec.hydra -iface br0
  -configfile /home1/01871/apb18/.slurm/config_file.920488.Vghpo0QI
Rank 1 of 2, host c558-202-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
Rank 0 of 2, host c558-202-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42

TACC: Shutdown complete. Exiting.
```

7. Now let's involve the hosts. Edit the script so that `ibrun.symm` runs both MIC and CPU executables, and specify that three processes should run on the host via the SLURM parameter `-n`.

```
#!/bin/bash
#SBATCH -t 00:05:00
#SBATCH -n 3
#SBATCH -p development
#SBATCH -A TG-TRA120006

echo "Running on MIC and CPU"
export MIC_PPN=2
export MY_ENV=42
export MIC_MY_ENV=mic42
ibrun.symm -m where_mpi.mic -c where_mpi.cpu
```

8. Submit the batch job, and verify the results

```
login$ sbatch where.sh
login$ cat slurm-920358.out
Running on MIC and CPU
TACC: Starting up job 920358
TACC: Starting parallel tasks...
  /opt/apps/intel13/impi/4.1.0.030/intel64/bin/mpiexec.hydra -iface br0
  -configfile /home1/01871/apb18/.slurm/config_file.920358.vqGVT4kT
Rank 2 of 5, host c560-702.stampede.tacc.utexas.edu; MY_ENV=42
Rank 1 of 5, host c560-702.stampede.tacc.utexas.edu; MY_ENV=42
Rank 0 of 5, host c560-702.stampede.tacc.utexas.edu; MY_ENV=42
Rank 3 of 5, host c560-702-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
Rank 4 of 5, host c560-702-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42

TACC: Shutdown complete. Exiting.
```

9. Now edit `where.sh` so that it executes on three nodes, with the same number of processes on each host or coprocessor (3 processes per host, 2 per MIC). You will have

to use `-N` to specify the number of nodes, and `-n` to define the total number of host processes.

```
#!/bin/bash

#SBATCH -t 00:05:00
#SBATCH -n 9
#SBATCH -N 3
#SBATCH -p development
#SBATCH -A TG-TRA120006

echo "Running on MIC"
export MIC_PPN=2
export MY_ENV=42
export MIC_MY_ENV=mic42
ibrun.symm -m where_mpi.mic -c where_mpi.cpu
```

10. Submit the batch job, and verify

```
login$ sbatch where.sh
login$ cat slurm-920411.out
Running on MIC
TACC: Starting up job 920411
TACC: Starting parallel tasks...
  /opt/apps/intel13/impi/4.1.0.030/intel64/bin/mpiexec.hydra -iface br0
  -configfile /home1/01871/apb18/.slurm/config_file.920411.eJRO2NVZ
Rank 2 of 15, host c557-204.stampede.tacc.utexas.edu; MY_ENV=42
Rank 1 of 15, host c557-204.stampede.tacc.utexas.edu; MY_ENV=42
Rank 5 of 15, host c557-701.stampede.tacc.utexas.edu; MY_ENV=42
Rank 10 of 15, host c557-703.stampede.tacc.utexas.edu; MY_ENV=42
Rank 0 of 15, host c557-204.stampede.tacc.utexas.edu; MY_ENV=42
Rank 6 of 15, host c557-701.stampede.tacc.utexas.edu; MY_ENV=42
Rank 11 of 15, host c557-703.stampede.tacc.utexas.edu; MY_ENV=42
Rank 7 of 15, host c557-701.stampede.tacc.utexas.edu; MY_ENV=42
Rank 12 of 15, host c557-703.stampede.tacc.utexas.edu; MY_ENV=42
Rank 13 of 15, host c557-703-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
Rank 3 of 15, host c557-204-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
Rank 8 of 15, host c557-701-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
Rank 4 of 15, host c557-204-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
Rank 9 of 15, host c557-701-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42
Rank 14 of 15, host c557-703-mic0.stampede.tacc.utexas.edu; MY_ENV=mic42

TACC: Shutdown complete. Exiting
```

We see three distinct hosts, three CPU ranks on each host, and two MIC ranks on each host. Success!

3 Non-trivial Symmetric hybrid example

In this example, we explore running a hybrid application symmetrically on host CPUs and coprocessors. We will use a hybrid (MPI+OpenMP) application for calculating pi via numeric integration. The argument to the program is the number of integration points - the larger the number, the more closely our number approaches the true value of pi.

The workload is divided evenly among the MPI ranks, and each process uses a number of OpenMP threads to process the work in a parallel for loop. It will become apparent that this is a flexible paradigm that allows us to decouple computing resources (number of threads) from workload (number of processes). This flexibility is especially appreciated on the heterogeneous computing environment formed by the hosts and coprocessors.

1. Unpack the lab materials into your home directory. If you've done the first two examples, there is no need to do this again.

   ```
   login$ cd
   login$ tar xvf ~tg459572/LABS/mic.tar
   ```

2. Change into the `mic/exercise3` directory. You will see a file named `pi_hybrid.c`. Feel free to take a look at it. It features a parallel `for` loop for computing the integral on a particular interval, as well as code to evenly divide the intervals among MPI processes.

3. Create two executables: one for the host CPU (`pi_hybrid.cpu`), and one for the MIC coprocessor (`pi_hybrid.mic`). Make sure to use the Intel MPI implementation (`impi`). If `impi` is already loaded, then you may omit the `module swap` line. Check with `module list` if you are not sure.

   ```
   login$ module swap mvapich2 impi
   login$ mpicc -openmp -O2 pi_hybrid.c -o pi_hybrid.cpu
   login$ mpicc -openmp -mmic -O2 pi_hybrid.c -o pi_hybrid.mic
   ```

4. Look at the batch script `pi_hybrid_mic.sh`. It uses `ibrun.symm` to launch our hybrid application on the MIC coprocessors only (none on host). It launches one process on the coprocessor, with 240 threads. With an almost entirely CPU-bound workload without any memory dependencies between the threads, we can assume that performance will be best when every core is completely saturated with threads. For more complicated memory-bound workloads, this number should be determined empirically.

5. Run the `pi_hybrid_mic.sh` script. It should take about a minute to finish, once it starts running.

   ```
   login$ sbatch pi_hybrid_mic.sh
   login$ cat slurm-919904.out
   Running on CPU
   TACC: Starting up job 919942
   TACC: Starting parallel tasks...
   /opt/apps/intel13/impi/4.1.0.030/intel64/bin/mpiexec.hydra -iface br0
   ```

```
-configfile /home1/01871/apb18/.slurm/config_file.919942.8Pc9rusv

My rank is: 0/1 host c559-503-mic0.stampede.tacc.utexas.edu

0: Integrating samples 1 to 501000000000
0@c559-503-mic0.stampede.tacc.utexas.edu DONE 42s
The value of pi is:  3.14159265

TACC: Shutdown complete. Exiting.

real    0m42.918s
user    0m0.037s
sys     0m0.025s
```

It took almost 43 seconds to calculate pi in this scenario. As you can see, only one MIC coprocessor was utilized. Can you modify the script to run on on two coprocessors?

6. Look at the batch script `pi_hybrid_symm.sh`. It uses `ibrun.symm` to launch our hybrid MPI application on the hosts and MIC coprocessors. Notice the `OMP_NUM_THREADS` and `MIC_OMP_NUM_THREADS` environment variables used to set different numbers of threads depending on where a process is running. Here, we launch one process on the CPU and one on the MIC. We use the maximum number of threads for each process.

7. Run the script, and view the results when it is finished. It should take about a minute once the job is running. Note: your slurm output file will have a different name.

```
login$ sbatch pi_hybrid_symm.sh
login$ cat slurm-873851.out
Running on CPU and MIC
TACC: Starting up job 873851
TACC: Starting parallel tasks...
  /opt/apps/intel13/impi/4.1.0.030/intel64/bin/mpiexec.hydra -iface br0
  -configfile /home1/01871/apb18/.slurm/config_file.873851.sv5jtzvi
My rank is: 0/2 host c558-504.stampede.tacc.utexas.edu

0: Integrating samples 1 to 250500000000
My rank is: 1/2 host c558-504-mic0.stampede.tacc.utexas.edu
1: Integrating samples 250500000001 to 501000000000
1@c558-504-mic0.stampede.tacc.utexas.edu DONE 21s
0@c558-504.stampede.tacc.utexas.edu DONE 55s
The value of pi is:  3.14159265

TACC: Shutdown complete. Exiting.

real    0m57.854s
user    14m51.654s
sys 0m0.088s
```

In the above example, we launched ONE process per device (host, coprocessor), and specified the maximum number of threads for each. The workload was split evenly. However, notice that it took almost 58 seconds to complete. When we ran the same code on the coprocessor only, it took 43 seconds. How can we add additional computing power (the host CPUs), yet have the same computational workload execute *slower*?

Take a look at the computation times printed out by each MPI rank. Each rank received half the workload. The MIC rank finished in 21s, while the CPU rank finished in 55 seconds. Because the MIC is so much faster than the CPU, it finished its work quickly and sat idle while the CPU struggled. Because computing the end result requires a barrier, the total computation time is constrained by the slowest worker.

Clearly, the work division is sub-optimal. We should assign more work to the MIC coprocessor, and less to the CPU..

Because our workload is CPU intensive, not memory bound, and does not involve a complicated communication topology, balancing the workload between the CPU and coprocessor is straightforward. Ideally, we would want to pick a ratio of work for the CPU and MIC that is roughly balanced. Because our example code evenly divides the work among MPI processes, the work ratio is equal the ratio of number of processes. Since the MIC appears to be 2.5 times as fast as the host CPUs for this particular application, we want to solve $x+2.5x = 1$ to find the best ratio x. In this case, that is .29.

We'd like to use a simple proportion of processes that is close to .29. A 1:3 ratio of CPU to MIC processes (.33) looks close enough. Let's try it and see if the results are any more balanced.

Because we want to utilize 100% of the CPU and 100% of the MIC, we want the ideal number of threads to be concurrently executing on each device. For the host, that is 16 threads in this case. Empirical investigation shows that 240 threads results in optimal performance on the MIC for our pi program.

At a 1:3 ratio of work, the CPU would get one process and the coprocessor would get 3. For the CPU, then, we want that process to have 16 threads. For the MIC, we want the sum of the three processes to total 240. Therefore, each process should be allowed $240/3 = 80$ threads.

8. Edit the `pi_hybrid_symm.sh` batch file to use our desired parameters. Set `MIC_PPN` to 3 (3 processes per MIC), and `MIC_OMP_NUM_THREADS` to 80. Re-submit modified batch file, and check the results.

```
login$ sbatch pi_hybrid_symm.sh
cat slurm-920136.out
Running on CPU and MIC
TACC: Starting up job 920136
TACC: Starting parallel tasks...
  /opt/apps/intel13/impi/4.1.0.030/intel64/bin/mpiexec.hydra -iface br0
  -configfile /home1/01871/apb18/.slurm/config_file.920136.mL8b4EHT
My rank is: 0/4 host c557-402.stampede.tacc.utexas.edu
```

```
0: Integrating samples 1 to 125250000000
My rank is: 1/4 host c557-402-mic0.stampede.tacc.utexas.edu
My rank is: 2/4 host c557-402-mic0.stampede.tacc.utexas.edu
2: Integrating samples 250500000001 to 375750000000
1: Integrating samples 125250000001 to 250500000000
My rank is: 3/4 host c557-402-mic0.stampede.tacc.utexas.edu
3: Integrating samples 375750000001 to 501000000000
0@c557-402.stampede.tacc.utexas.edu DONE 28s
2@c557-402-mic0.stampede.tacc.utexas.edu DONE 31s
1@c557-402-mic0.stampede.tacc.utexas.edu DONE 31s
3@c557-402-mic0.stampede.tacc.utexas.edu DONE 31s
The value of pi is:  3.14159265

TACC: Shutdown complete. Exiting.

real    0m33.129s
user    7m33.571s
sys     0m0.373s
```

Much better! For comparison, running on MIC only took 43 seconds, and a naively balanced symmetric job took 57. We see almost a twofold improvement in symmetric performance simply by intelligently balancing the workload so that the coprocessors and CPUs are both utilized 100% for the duration of the computation. Now that we've found parameters that make effective use of a single node, we can apply the same parameters when running our job on multiple nodes.

9. Lastly, modify the script to run on three nodes, and re-run. Verify that work is assigned to three separate nodes by looking at the output. For convenience, you can use the following command to sift through the results and produce an easily interpretable output. It will print all unique hostnames.

   ```
   cat slurm-920610.out | grep rank | awk '{print $6}' | sort | uniq
   ```

   - Remember, SLURM parameters determine the number of nodes and the distribution of CPU processes. You will need to pick appropriate values for `-n` and `-N` in order to allocate the correct number of nodes, and make sure that only one CPU process is running on each node.