ME964
High Performance Computing for Engineering Applications
Assignment 10
Date Assigned: March 31, 2012
Date Due: April 12, 2012 – 11:59 PM

The goals of this assignment are as follows:
- Getting more versed with using MPI by implementing code to evaluate an integral (Problem 1)
- Understanding the potential of MPI relative to that of CUDA. To this end, you'll perform a vector reduction using the native MPI reduction support in OpenMPI and then compare its efficiency with that of handcrafted MPI and then CUDA implementations (Problem 2)
- Gauge the overhead associated with passing messages between compute nodes when using OpenMPI over a 40 Gbs Infiniband interconnect (Problem 3)

Problem 1. Drawing on the integral calculation example presented in class, write a program that uses the MPI parallel programing paradigm to evaluate the integral

$$I = \int_0^{100} e^{\sin x} \cos(\frac{x}{40}) dx$$

Note that the value provided by MATLAB for this integral is I = 32.121040688226245. To approximate the value of I use the following extended Simpson's rule:

$$\int_0^{100} f(x)dx \approx \frac{h}{48}\left[17f(x_0) + 59f(x_1) + 43f(x_2) + 49f(x_3) + 48\sum_{i=4}^{n-4} f(x_i) + 49f(x_{n-3}) + 43f(x_{n-2}) + 59f(x_{n-1}) + 17f(x_n)\right]$$

In the approximation above, $x_0 = 0$, $x_n = 100$, $h = 10^{-4}$, and $n = \frac{100-0}{h} = 10^6$. This value of n goes to say that you divide the interval $[0, 100]$ in 10^6 subintervals when evaluating I.

After implementing the code, you will have to run the code on Euler using
- one node and one core
- one node and four cores
- one node and eight cores (Euler has on each compute node two quad-core Intel Xeon 5520)
- two nodes and four cores on each node
- four nodes and two cores on each node

Your report should include
- A "results table" that summarizes your findings
- A discussion of the results you obtained for the five scenarios above
- The execution configuration; i.e., number of compute nodes and number of cores per node that produces the value of I in the shortest amount of time. Report this combination along with the corresponding timing result on the forum.

Problem 2. Go back to your favorite implementation of the reduction operation on the GPU. The array that you handle for this problem is of random integers between -5 and 5. First, figure out the largest array size that the CUDA implementation that you can handle. With this array:

- Perform a reduction operation using the $+$ operator on the GPU
- Write your own implementation of the reduction task by combining a straight sequential summation by each process, collecting results to a root process and producing the final result therein. Use as many processes NP as you find advantageous.
- Use the MPI_Reduce function. Use as many processes NP as you find advantageous.

Report on the forum your results in milliseconds using the following format:

```
Max array size handled: blah
CUDA Implementation: blahblah
Handcrafted implementation (NP=…): blahblahblah
MPI_Reduce (NP=…): blahblah
```

Problem 3. This problem is the sister of Assignment's 7 Problem 1.

i) Run an analysis to gauge how much time it takes to move data from a process A to a process B using Euler's OpenMPI implementation of the MPI standard. To this end, transfer from a process A to a process B 2^0 bytes; 2^1 bytes; 2^2 bytes; …; 2^{30} bytes. Generate a **`png`** plot that shows the amount of time required by each of these transfers. Do not register the amount of time necessary to allocate memory. You might want to allocate memory once, for the most demanding case (2^{30} bytes), and then use it for all the other data transfer cases. Make sure you run your code several times to get a good idea about the average amount of time you can expect in a real-life application.

ii) Do the same thing as before but now time a data transfer from A to B followed immediately by a data transfer from B to A. This is called Ping-Pong. To this end, allocate 2^{30} bytes on A. Then allocate the same amount on B. Time the following sequence of two operations: copy N bytes from A into B, then from B back into A. Run this analysis for $N = 2^0,\ldots,2^{30}$ bytes and generate a **`png`** plot with timing results for each value of N.

For this problem, upload the FOUR **`png`** plots to the forum (under topic "Assignment 10: Data transfer overhead plots"). The plots you provide should be identified as follows and capture four different scenarios:

- "PLOT 1" – i) above & processes A and B live on the same node
- "PLOT 2" – i) above & processes A and B live on different nodes
- "PLOT 3" – ii) above & processes A and B live on the same node
- "PLOT 4" – ii) above & processes A and B live on different nodes.

Answer these two questions in a separate document that needs to be submitted through **`Mercurial`**:

- How are your results correlating when A and B are on the same node as opposed to two nodes?
- How are your results correlating when one way vs. two way (Ping-Pong) data movement operations are considered?

NOTE: You should use the `MPI_Ssend` flavor of the send operation to endure synchronization of the send/receive operation. This is just to make sure things are fair for small data transactions, when the data might be buffered by OpenMPI for you. In other words, we are enforcing a "rendezvous" always policy and not get tricked by "eager" mode transfers.