ME759
High Performance Computing for Engineering Applications
Assignment 11
Date Assigned: November 26, 2013
Date Due: December 4, 2013 – 11:59 PM

Pick at least two out of the five problems below and provide their solution. Return by Wednesday,
December 4 at 11:59 PM.
The goals of this assignment are as follows:
- Understand why a solution based on collective communication, specifically on the MPI_Bcast service, is superior to ad-hoc solutions drawing on point-to-point communication (Problem 1)
- Exercise the use of MPI_Send and MPI_Isend in MPI parallel programming and understand the benefits/drawbacks associated with each one of them (Problem 2)
- Getting more versed with using MPI by implementing code to evaluate an integral (Problem 3)
- 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 4)
- Gauge the overhead associated with passing messages between compute nodes when using OpenMPI over a 40 Gbs Infiniband interconnect (Problem 5)

Problem 3. Drawing on the integral calculation example presented in class, write a program that uses the MPI parallel programing paradigm to evaluate the integral

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

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,

$$x_0 = 0,\ x_n = 100,\ h = 10^{-4},\ \text{and}\ n = \frac{100-0}{h} = 10^6$$

. This value of *n* goes to say that you divide the interval [0, 100] in 106 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.

Answer:

No. of nodes	No. of cores	Time taken for integral calculation (in ms)
1	1	138.632
1	4	61.8
1	8	22.01
2	4	24.35
4	2	28.54

The above results show that the time taken in case of 1 node and 1 core is serial execution and it takes more time than other cases in which parallel cores are executed for doing the same calculation.
Also, the case in which the no. of nodes are more than 1 have faster calculation time as compared to the one with one node but parallel cores.

Problem 5. 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 20 bytes; 21 bytes; 22 bytes; …; 230 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 (230 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 instead 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 230 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 0 30 $N = 2, \ldots, 2$ bytes and generate a **png**

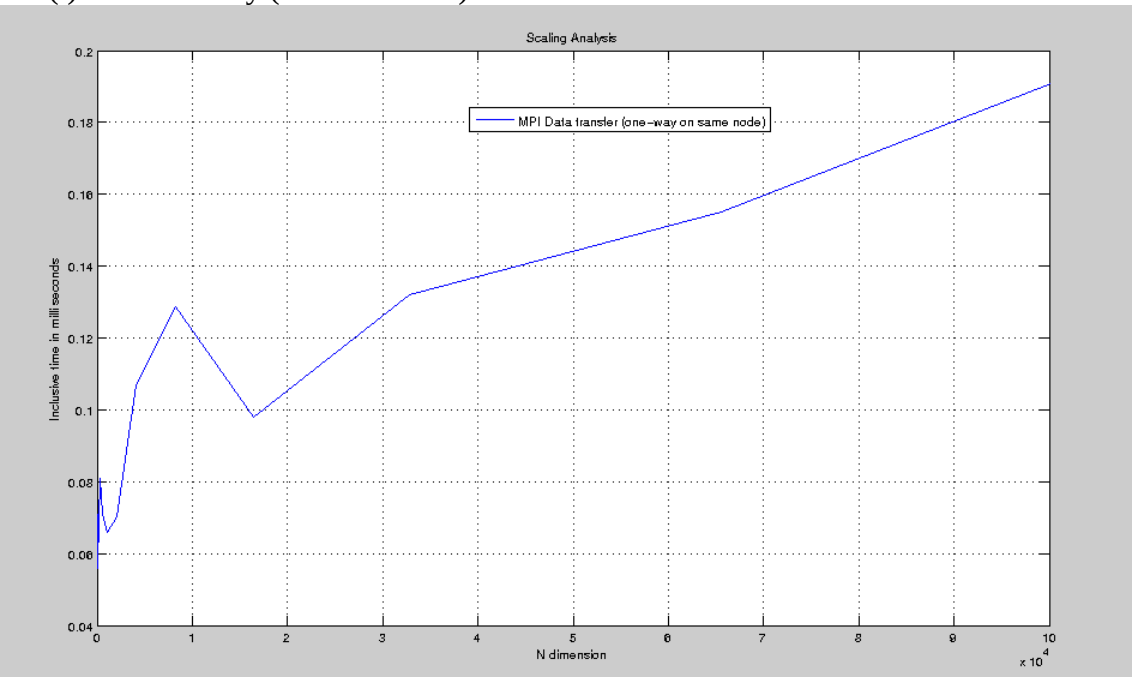
plot with timing results for each value of *N* .
For this problem, upload the FOUR **png** plots to the. The plots you provide should be identified as
follows and capture four different scenarios:
• "PLOT 1" – i) above & processes A and B live on the same node
• "PLOT 2" – i) above & processes A and 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.
In your report, answer these two questions:
• 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.

Answer: The results of data transfer time between process A & B when data transfer is taking place one-way on the same node and different nodes is on increasing trend with sharp up—down trends at some values.

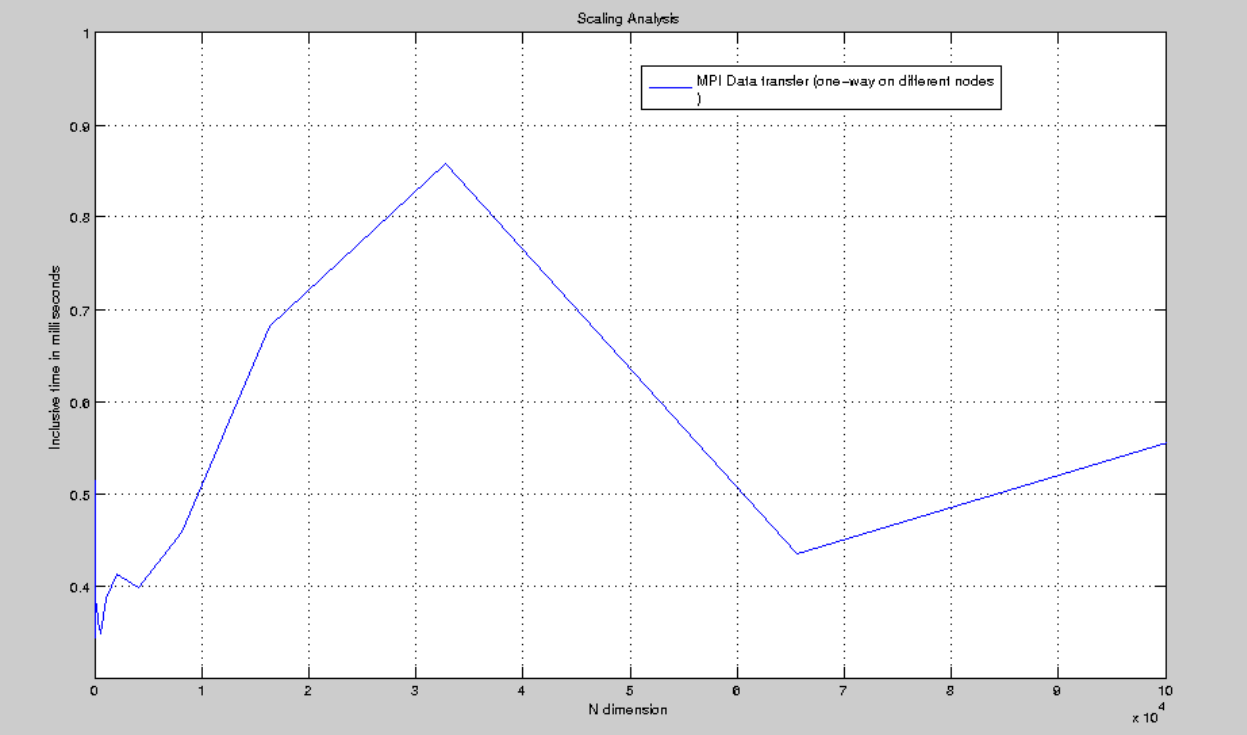
The results between one-way and two-way are related as the data transfer time is on the increasing trend in both the cases.

The plots are as follows:

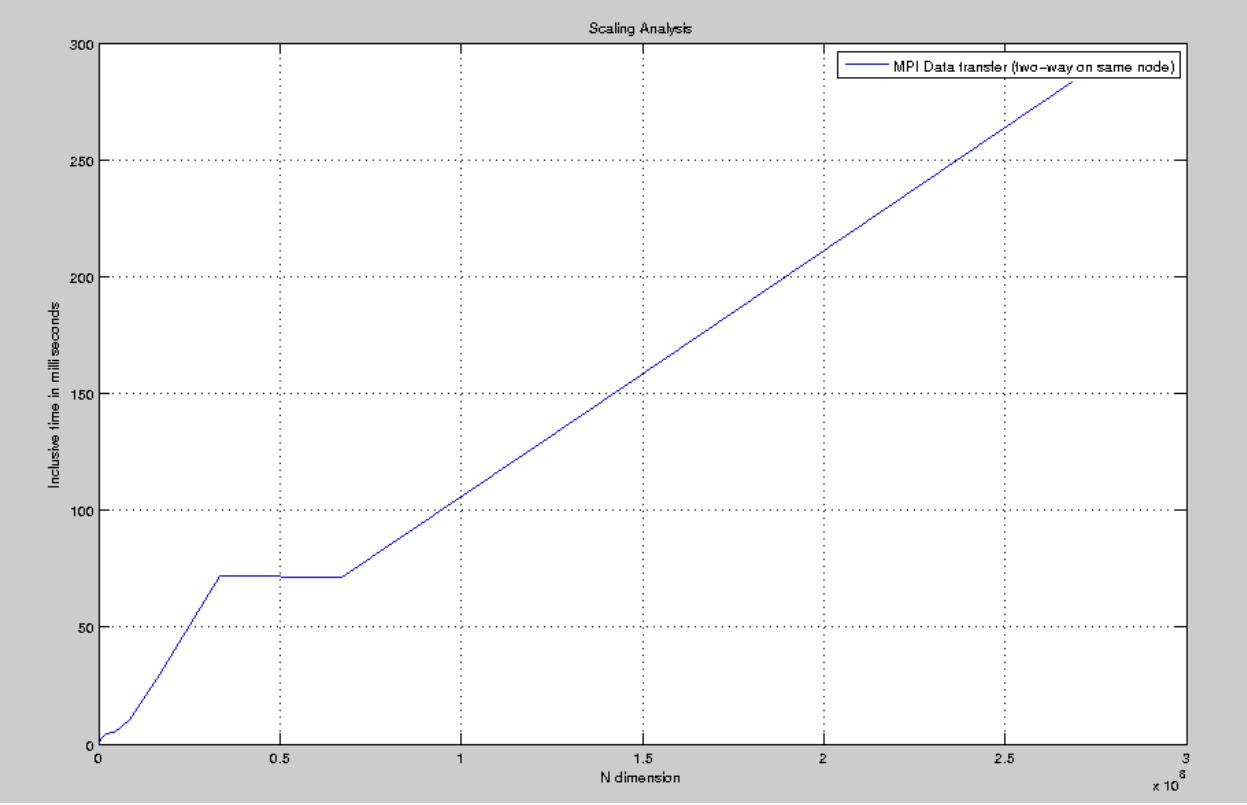
(i) One-way (on-same node)



(ii) One-way different nodes:



(iii) Two-way same node:



Two-way different nodes: