SINGAPORE-MIT ALLIANCE
COMPUTATIONAL ENGINEERING PROGRAMME
CME5232: Cluster and Grid Computing Technologies for Scientific Computing
COMPUTATIONAL LAB No.2

10th July, 2009

GETTING STARTED ON PARALLEL PROGRAMMING USING MPI AND ESTIMATING PARALLEL PERFORMANCE METRICS

Objectives

To experiment with MPI library functions for parallelizing sample codes, use MPI functions for point-to-point and collective communications and estimate parallel performance metricss.

Background Knowledge

C/C++ languages, Basis MPI Functions and codes from Computational Lab 1.

Parallelization Using MPI

There are a number of parallel software environments that allow users to use the networked computers as a unified computing resource, e.g. PVM (Parallel Virtual Machine), MPI (Message Passing Interface) and so on. MPI is a set of specifications that has become a de-factor standard in message passing protocols. MPI has been implemented through PVM, P4, CHIMP MPICH and LAM. MPI is designed to be language independent (it can work with c/c++ or fortran) and of computer architecture. It is designed to resolve all heterogeneities encountered in distributed computing in a transparent manner. MPI is flexible and offers users a variety of binding functions to write efficient parallel codes. There are many MPI functions that implement many aspects of parallelization but only about a dozen or so are really needed frequently. the focus of this lab is on exploiting these bare minimum MPI functions to help you get started on writing parallel code using MPI. The MPICH, an implementation of MPI, is available on SMA Hydra Linux Cluster for all your parallelizeation works.

Exercises (Due date: 16th July, 2009)

Please submit a short format report recording your observations based on the assigned exercises and programming assignment in a week's time. The report should be short, complete and clearly written to show that you have done and what you have observed and interpreted from these exercises. All equations, symbols etc. if any must be explained clearly.

1. Play Ping-Pong using 2 processors on the SMA Hydra Cluster

Write a program using simple send and receive MPI functions in which 2 processors are sending their process ranks to each other and at the end they print the values of what they sent and received.

As sample skeleton source source code **pingpong.c** or **pingpong.cpp** is provided for your reference.

2. Estimate bandwidth and message latency on the SMA Hydra Cluster

Write a short program to estimate the message latency t_s and reciprocal of the bandwidth t_c for SMA Hydra Cluster. In this problem, we are using two (02) processors to send and receive repeatedly the fixed-size messages varying from 10^2Bytes to 10^4 Bytes (you can choose step = 100 Bytes) among them. In order to get the correct timing, we repeat sending and receiving 10 times for each fixed size message. You are then required to record the average time corresponding with the different size of each message, then construct a least-square-fit line to the resulting (message size, time) pair. The intercept on the time versus message size curve should give an estimate of the startup cost t_s while the gradient of the slope will give an estimate for t_c. Discuss on the observations.

3. Exercises based on the sample codes

(a) Given a parallel code **integral.c** (Note: download from $..Home\backslash CME5232\backslash Clab2\backslash question3$). You are required to compile and run that parallel integration code using different number of processors varying ($n = 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20$)., record the timing for each case, and then plot the timing versus number of processor. Use the **Jumpshot** facility in MPICH to display the space time status of your parallel computation (Show only the jumpshot graph for the case with 10 processors). What happens if you try to run with it just one processor? Run the sequential code **ori-integral.c** on one processor. What are the different between running a sequential code and parallelized code using one processor.

(b) A more accurate alternative to trapezoidal rule is Simpson's rule. It takes the following form in the interval of the integration [a,b] which is divided in to even intervals;

$$\int_a^b f(x)dx = \left(\frac{h}{3}\right)\Big[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\Big]. \quad (1)$$

Assume that you integrate using even number of processor p so that the ratio of n/p is even. Modify the programs ori-integral.c and integral.c to estimate the value of the integral. State clearly how you have implemented message passing function and compare the results with those from the trapezoidal rule (Show the only Jumpshot graph for the case with largest number of processors).

4. Parallel your code for calculating the value of PI using Monte Carlo Method from Computational Lab1.

This question will focus on the parallelization of the sequential code what you have written in the computational lab1 to evaluate PI number using Monte Carlo method.

(a) Write your own code using (i) only $MPI - SEND$ and $MPI - RECV$ (point to point operation). (ii) MPI-Reduce (Collecting operations). Choose a fixed total number needle drops like 3.2×10^8. Then run the code by varying the number of processors, p = 2, 4, 8, 16 and 32. Record the communication time and the value of your PI. Use the Jumpshot facility in MPICH to display

the space time status of your parallel computation. Note: show only the case with 16 processors.
(b) Compare the value of PI and accuracy between both your parallel codes (point-to-point and collective) and serial code. Plot the convergence rate of parallel code and serial code versus number of varying needle drop ($N = 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8$), where the relative error of parallel code is $\epsilon_p = (PI25DT - PI_p)/PI25DT$, and the relative accuracy of serial code is $\epsilon_s = (PI25DT - PI_s)/PI25DT$ and
PI25DT = 3.141592653587932384262643 is the true π with 25 digits of accuracy.
(c) Calculate the speedups and efficiencies for your own parallel code. Plot the curves (α) of predict and actual speedup, and (β) predict and actual efficiency for your parallel version. Discuss on your observation.
(d) Discuss your observation and make the statement about the ratio of computational time to communication time. Explain clearly how you have estimated communication can computational time for this problem. That means show clearly where you insert $MPI - Wtime()$ function to measure the running time.