SINGAPORE-MIT ALLIANCE

Computational Engineering
CME5232: Cluster and Grid Computing Technologies for Science and Computing
COMPUTATIONAL LAB NO.2
10th July 2009

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

Question1: Play Ping – Pong Using 2 Processes on the SMA Hydra Cluster

In this question, you are required to write a program using simple **MPI_Send** and **MPI_Recv** functions to send and receive their process ranks to each other. At the end they both print the values of what they sent as well as what they received. A sample skeleton source code, named "**pingpong.c**", and "**pingpong.cpp**" are provided for your reference.

(a) Where to download the sample skeleton code?
Download from **/home/cme5232/clab/clab2/qn1/**.

(b) What should we add on or modify the skeleton source code?
Properly insert the MPI_Send () and MPI_Recv() functions into the program.

(c) How to compile the "**pingpong.c**" or "**pingpong.cpp**" program?

```
C       : mpicc pingpong.c –o pingpong –lm –mpilog
C++     : mpicxx pingpong.cpp –o pingpong –lm –mpilog
```

(d) How to submit and run the program in SMA Hydra Cluster?
To submit job directly to the nodes, you need to define your own machine file, i.e. the compute nodes use for computation. See the following sample_machinefile:

```
----------
node10
node25
----------
```

$mpirun –np 2 –nolocal –machinefile sample_machinefile pingpong
Note:
- You are not supposed to run mpirun command to submit your job in certain node since this will affect to the performance of the system greatly.
- Use the SGE to submit job instead.

It is compulsory to submit MPI job via SGE, Below are some commonly used SGE commands

SGE commands	Functions	Command Usage Examples
qsub	Submit batch jobs	qsub <script> qsub –cwd –S/bin/sh <script>
qstat	Retrieve job status information	qstat qstat –f qstat –u user qstat –j <job_id>
qdel	Delete/ cancel submitted job	qdel <job_id> qdel –f <job_id>
qmod	Suspend or Resume job	qmod –s<job_id> qmod –r<job_id>
qmon	Provide an X Window GUI command interface and monitoring facility	qmon
qconf	Provide the user interface for cluster	qconf -sql

	configuration and queue configuration	
qhost	Display status information about execution hosts	qhost

(e) A simple job script example **simple_sge.sh**
$vi simple_sge.sh

```
--------------------------------------------------------------------------------
#!bin/sh
#Above line is the default shell interpret for Sun Grad Engine
#
#Request Bourne shell as the shell for the job
#$ -S/bin/sh
#
#Run in current working directory, i.e. submit directory
#$ -cwd
#
#Print the date and time
date
# Print the hostname where the job is running
Hostname
--------------------------------------------------------------------------------
```

Little explanation about the simple serial job script

#$	- The prefix to add SGE option to the script file (special comment line)
#	- Comment lines. They are not read by shell

Some commonly used SGE options:

SGE options	Explanations
-S shell_name	Specify the shell interpreter to be used for the job.
-V	Export all environmental variables.
-N job_name	Specify the name of the job to be used by SGE. Without this option, the name of the job is the file name of the submitted job script.
-cwd>	Run the job in current working directory. Without this option, the job's output file (stdout, stderr) will always put in your home directory.
-o filename	Specify the file name where SGE should save the message from the standard out channel of your job. Without the **–o** option, the name of your job (usually the name of the job script, if not specified with **-N**) plus the extension **.o<job_id>** is used, where **<job_id>** is the job number assigned by SGE.
-e filename	Specify the filename where SGE should have the messages from the standard error channel of your job. Without the **–e** option, the name of your job (usually the name of the job script, if not specified with -N) plus the extension **.e<job_id>** is used, where **<job_id>** is the job number assigned by SGE.
-jy	Write a standard error to the same file as standard output.

(f) How to submit the job via SGE and monitor the job status?
$qsub simple_sge.sg
Your **job <job_id>** ("simple_sge.sh") has been submitted.
$qstat
…

By using **qstat** command, any jobs running or pending in the queue will have a job number and a job status.

- **qw** (queue and waiting).
- **t** (job transferring and about to start).
- **r** (job is running o listed hosts).
- **d** (job has been marked for deletion)

(f) Outputs produced by submit simple_sge.sh to SGE?
When a job is queued, it is allocated a job number. One it starts to run, output are usually sent to standard output and standard error, and they are spooled to files called

- **<job_name>.o<job_id>**: Standard output of the job.
- **<job_name>.e<job_id>**: Standard error of the job.

(g) Job submit via script for pingpong MPI job.
$via pingpong_sge.sh

```
-----------------------------------------------------------------------------------------------
#!/bin/sh
#$ -S /bin/sh
#$ -cwd
#$ -V
#$ -N pingpong-check
# pe(parallel environment) request for MPICH and number of CPUs ($NSLOTS)
# This command line is usually used outside the SGE file for convenience.
#$ -pe mpich 2 (Can be omit to put in the command line)
myjob=pingpong
mpirun –np $NSLOTS –machinefile $TMPDIR/machines $myjob
-----------------------------------------------------------------------------------------------
```

SGE uses the concept of a parallel environment PE (PE: a list of hosts along with a set of number of job slots and PRE/POST execution) to execute MPI jobs. Each host has an associated queue and resources (CPU, memory).

$qsub pingpong_sge.sh
Your job<job_id>("clab2_qn1 -pingpong") has been submitted.

Note: To put the pe (parallel environment) command line outside the SGE file for convenience, we omit this command line in SGE script file and insert it on the submitting job command line as follows:
$qsub –pe mpich 2 pinpong_sge.sh

In addition to the **<>.o** and **<>.e** files, there are two more output files with the parallel jobs
-<job_name>.po<job_id>: Parallel execution standard output of the job (hostfile lists).
-<job_name>.pe<job_id>: Parallel execution standard error of the job (hostfile lists).

If the job fail for any reason, it is **<>.o** and **<>.e** one should examine to determine failed reason. The **<>.o** can often be used to check on the progress of the job. Check **<job_name>.o<job_id>** for the output of your pingpong program. Check out other SGE output files as well.

(h) How to obtain a graphical view of the message passing between processes?
- Required PC software **X-Win32**, which is a **Window X-server** for 32-bit computer.
- Use **Jumpshot** – a Java-based visualization tool for doing postmortem performance analysis. **Jumpshot-4** is installed in SMA Hydra Cluster. It supports **.slog2** log file format. **mpicc/mpicxx** with option "-mpilog" will produce **pingpong.clog** file. Use clogTOslog2 to convert **.clog** to **.slog2**.
- Run the **X-Win32** software on your PC desktop. Copy the **IP address** of your computer.

$ clogTOslog2 pingpong.clog (Convert the pingpong.clog to pingpong.slog2)
$ export DISPLAY=<IP Adress of your PC Desktop>:0 (0: is the display number)
$jumpshot pingpong.slog2

Jumpshot-4 plays around the diagram for diagrams offer a lot of improvement features and useful information sometime. See http://www-unix.mcs.anl.gov/perfvis/software/viewers/ for more details.

Try to use command: **$jumpshot pingpong.clog**

Question 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 the SMA Hydra Cluster. This can be done repeated sending and receiving fixed size of messages from one process to another. When the receiving process receives the message, it should reply to the sending process. The pingpong program developed in the question 1 can be used for this question. This process should then be repeated with messages of difference sizes. After collecting the timings, construct a least-square-fit line to resulting (message size, time) pair. The intercept on the time versus message size line should give an estimate of the startup cost, t_s, while the gradient of the slope will give an estimate for t_c.

(a) Choose the step and number difference size of message to run.
The fixed size of messages start from 100 bytes, increase by a step size of 100 bytes until the message size reaches 10000 bytes. These will be total of 100 different message sizes and should be enough to construct a least-square-fit line to resulting (message size, time) pair.

(b) Proper timing measurement
Properly place MPI_Wtime() in the program to obtain more accurate timing.

(c) How many calculated Round Trip Time (RTT) for each message to get timing accurately?
RTT is the total time taken in completing a “pingpong” process. To obtain more accurate RTT, you might want to obtain several RTTs for each message size and taken average of them. In this case you are required to repeat 10 times.

(d) How to construct a least-square-fit line to the resulting (message size, time) pair?
You can use Microsoft Excel to plot out the (message size, time) pair and construct the least-square-fit line. Note that the value of “time” in (message size, time) pair.

+ Copy the results in Microsoft excel into 2 columns (1 for message size, other for time)
+ Choose both columns to plot using scatter.
+ Click on the data then right click, choose Add Trendline option.
+ Choose the linear line.
+ Double click on the line, choose “option”, then choose “Display the equation on chart”

Question 3: Exercises based on the sample codes.
(3a) Given a parallel code **integral.c** , you are required to compile and run that parallel integration code using different number of processor 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 the number of processors. Using jumpshot facility in MPICH to display the space time status of your parallel computation (show only the jumpshot graph for the case with 10 processes). What happen is you try to run the program with one processor? Run the serial **code ori_integral.c** on one processor. What are the difference between running a sequential code and parallel code using one processr?

(a) What is the implementation in the **integral.c** program?
- Parallel the code that perform integration using the trapezoidal method.
- Using MPE code that perform the customized logging

(b) Parallel the code that perform the integration using trapezoidal method
----
```
h = (b - a) / n;    /* h is the same for all processes */
local_n = n / p;    /* So is the number of trapezoids */

local_a = a + my_rank * local_n * h;
local_b = local_a + local_n * h;
integral = Trap(local_a, local_b, local_n, h);

/* Add up the integrals calculated by each process */
```

```
if (my_rank == 0)
      {
           total_integral = integral;
           for (source=1; source<p; source++)
                 {
                    MPI_Recv(&integral, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
                    total_integral = total_integral + integral;
                 }
      }
else MPI_Send(&integral, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
```

(c) MPE code that performs customized logging

The MPE (Multi – Processing Environment) attempts to provide a complete suite of performance analysis tools for MPI programs based on post processing approaches. These tools include a set of profiling libraries, a set of utility programs, and a set of graphical visualization tools. The profiling libraries provide collection of routines that can create log files. These log files can be created manually by inserting MPE calls in the MPI program, or automatically by linking with the appropriate MPE libraries, or by combining the above two methods. Currently, MPE offers the following threes libraries:

- Tracing library: This traces all MPI calls. Each MPI call is preceded by a line that contains the rank in MPI_COMM_WORLD of the calling process, and all followed by another line indicating that the call has completed. Most send and receive routines also indicate the value of count, tag, and partner (destination for send and source for receive). Output is to standard output. (-o mpitrace to compile and link this tracing library).
- Animation library: It does a simple form of real time program animation (-o mpiami to compile and link with animation library).
- Logging library: They form the basis to generate log files from user MPI programs. It is the most useful and widely used profiling libraries in MPE. (-o mpilog to compile and link with logging library).

In this course, the logging library is preferred profiling library whenever profiling is required. There are currently 3 different log file formats allowed in **MPE**. The default log file format is clog, which is basically a collection of events with single time stamp **.alog** is provided for backward compatibility reason. The most powerful one is **slog2** (Scalable LOGfile, iit was .slog previously), or can be generated (preferred approach), or can be generated directly when MPI program is execute (through setting the environmental variable **MPE_LOG_FORMAT** to **SLOG**). The set of utility program in MPE includes **clogTOslog2**, **logviewer**, and so on. Currently, MPE's graphical tools include 2 display program, **Upshot** for .alog and **Jumpshot-4** for **.slog2.**

<u>Customized logging</u>

In addition to using the predefined MPE logging library to log all MPI calls, MPE logging calls can be inserted into user's MPI program to define and log states. These states are called user defined stated. States may be nested, allowing one to define state describing a user routine that contains several MPI calls, and display both the user defined state and the MPI operation contains within it. the routine MPI_Log_get_event_number() should be used to get unique event numbers from the MPE system. The routine MPI_Describe_state() and MPE_Log_event() are then used to describe user defined states. The following example illustrate the use of these routine.

```
#include "mpe.h"
….
int eventide_begin, eventide_end;
…..
eventID_begin  =  MPE_Log_get_event_number();
eventide_end    =  MPE_Log_get_event_number();
….
MPE_Describe_state(eventide_begin, eventide_end, "Amult", "bluegreen");
….
```

```
MyAmult(matrix m, Vector n);
{
        /* Log the start event along with the size of matrix */
        MPE_Log_event(eventide_begin, m->n, (char*)0);
        … Amult code, including MPI calls…
        MPI_Log_event(eventide, 0,(char *)0);
}
```

If the MPI logging library, **'limimpe.a'** is not linked with the user program, **MPE_Unit_log()** and **MPE_Finish_log()** must be called before and after all the MPE calls.

The following is the MPE code that perform customized logging in **integral.c**

```
int event1a, event1b;
….
event1a  =  MPE_Log_get_event_number();
event1b  =  MPE_Log_get_event_number();
….
MPE_Describe_state(event1a,event1b, "Compute", "red");
…..
MPE_Log_event(event1a,0, "start compute");
/* Code to evaluate the integral within the interval assigned to each process*/
MPE_Log_event(event1b, 0, "end compute");
```

(3b) A more accurate alternative to trapezoidal rule is Simpson's rule. It takes the following form in the interval of integration [a,b] which is divided into n even intervals:

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

Assume that you integrate using even number of processes p so that the ratio n/p is even. Modify the programs ori_integral.c and integral.c to estimate the value of integral. State clearly how you have implemented message passing function and compare the result with those from the trapezoidal rule.

Note: In order for Simpson's rule to work properly, make sure n/p is even.

Question 4: Parallelize your sequential code for calculating the value of PI using the Monte Carlo method from Computational Lab1.

This question will focus on the parallelization of the sequential code, which you have written in the Computational Lab1 to evaluate the value of PI using Monte Carlo method.

+ Write your own parallel code using simple MPI_Send() and MPI_ Revc() (Point to point communication method).

+ Write your own parallel code using Collective communication (MPI_Bcast, MPI_Reduce, MPI_allgether, …).

+ Run your parallel code by varying number of processors (2, 4, 8, 16 and 32) for total number of "dart" thrown is M = 3.2×10^8. Record the computational time and communication time, the value of PI. Use the JUMPSHOT facility in MPICH to display the space-time status of your parallel computation (show with the case of 16 processors). Discuss on the observation.

+ Run your both codes with difference number of "dart" thrown (10^2, 10^3, 10^4, 10^5, 10^6, 10^7, $10^{8,}$ 10^9) using 8 processors. Record the execution time, value of PI. Plot the comparison of the execution time of your both parallel methods with serial code. Plot the convergence rate of your both parallel methods with serial code. Where the relative errors are calculated by

$$\varepsilon = \left| \frac{PI25DT - PI}{PI25DT} \right|$$

here, PI25DT = 3.1459265358793238462643 is the true value of PI with 25 digits of accuracy. Discuss on the observation.

+ Calculate the speedups and efficiencies for both parallel codes. Plot the curve of (i) Predicted and actual Speedups. (ii) Predicted and actual Efficiencies for both parallel versions. Comment on the observation. Where the Speedup is calculated by

- Predicted Speedup: $S_P = \dfrac{1}{(1-a) + \dfrac{a}{P} + \dfrac{T_D}{T_1}}$
- Actual Speedup: $S_a = \dfrac{T_1}{T_p}$

And Efficiency is calculated by

- Predicted Efficiency: $e_P = \dfrac{S_p}{P}$
- Actual Efficiency: $e_a = \dfrac{S_a}{P}$

where, P is the number of processor used.
A is the fraction of operation that can be parallelized
T_D is the delay due to the communication and memory contention
T_1 is the time taken on single processor
T_p is the time taken on p processors

+ Discuss on your observation and make the statement about the ratio of computational time to communication time. Plot the ratio (T_{comp}/T_{comm}) versus number of processors => give the comment on that. Properly place the MPI_Wtime() in the program to obtain a better estimate of the computational time and communication time (Explain how to implement the code to measure the timing with MPI_Wtime).