**HW5: Programming for Performance**

This is the last assignment of this semester. In this assignment, you are expected to not only write and run programs correctly, but also achieve good performance of the programs. All programs should be run on the cluster and recorded times (Before that, you may test them on your own PC).

**1**. Measure the execution time, speedup and efficiency of your MPI programs for PA 3.5 and PA 3.6 in HW3.

In your implementation, you need to generate the random matrix and vector on one process, and then to distribute them to other processes. In addition to the implementation, you need to do the following:

a) Compute the 2-norm of the product vector in parallel.

b) Measure the wall-clock time spent in matrix-vector multiplication and vector 2-norm computation, respectively.

c) Present the performance evaluation of your algorithms and compare them. Discuss your findings and make comments on the results.

**2**. Programming assignment 4.5.

a) We’ll not measure the performance of this program.

b) The program has two inputs: <number of threads> and <number of tasks>

c) The tasks are the operations on a linked list. We can define the task option as:

0, 1: insert

2: delete

3: check if the data is in the list

4: print the linked list

d) Each task is specified with a random number (0~4) as the task option. The main thread generates the tasks and enqueue them to the “task queue”.

e) In this program, you should have a linked list and queue. And, use mutex and conditional variable, or other similar things.

f) The main function is like this (incomplete but show the main steps):

int main(int argc, char\* argv[]) {

if(argc != 3) Usage(argv[0]);

thread\_count = strtol(argv[1], NULL, 10);

n = strtol(argv[2], NULL, 10);

thread\_handles = malloc(thread\_count\*sizeof(pthread\_t));

/\* Initialize mutexes and conditional variables \*/

/\* Start threads \*/

/\* Generate tasks \*/

/\* Wait for threads to complete \*/

Free\_list();

/\* Destroy mutex and conditional variables \*/

} /\* main \*/

**3.** Programming assignment 5.3

**4**. Let . The computation of the i-th element of is defined as:

where .

1. Write an OpenMP parallel program to implement the computation of for the given input of and . Note that the computational time for depends on . When the value of varies, the execution of also varies. Assume has random distribution between 1 and n. A sequential code describing the computation to help you understanding the problem is attached in a separate file for your reference. Use static and dynamic schedule methods with difference chunk size.
2. Observe and compare the performance of the different schedule strategies.
3. Implement the parallel computation using Pthreads. You may use the “task queue” implemented in Programming assignment 4.5 for the implementation of dynamic schedule. (Note: When compare the performance between methods, use a fixed . But your algorithm should not be optimized for a particular , instead, your algorithm should be optimized for general random .)

**Important note 1**: To complete the assignment, you need not only write your program, you need also measure the performance of the programs and **write your report** on your experiments. When you present the performance of your program or the comparison of the performance in your report, please use tables or figures for presentation. Do not simply paste the screen shot of raw data from program execution. The raw data and screen shot pictures could be submitted as attachment if you think they are important supporting documents.

**Important note 2**: While testing the performance of your program, make sure that you set a problem size properly so that the program’s execution time is neither too short nor too long (say, not exceeding 5 minutes), and the number of cores requested would not exceed, say, 20 cores on the cluster. Notice there may be many tasks running at the same time that request cores for execution. For shared-memory program, it could only run on a single (multi-core) node, while for MPI program you could use a couple of nodes for execution.