CPSC/CPEN 435 Project

Purpose: to learn how to write a hybrid program that combines MPI and Pthread programming paradigms as well as how to utilize different techniques to achieve a better performance.

MPI is a natural choice for parallel computations on a cluster of loosely-connected nodes. However, to fully utilize the computing resources (e.g. multiple CPUs) on each individual node, shared address space programming is a better choice since creating and terminating a thread is less expensive than managing a process. To take advantage of both programming paradigms, we can write hybrid MPI/Pthread programs where MPI is used to distribute tasks (or data), and several threads are created within each node to do the computation. Note: a simple hybrid MPI/Pthread program as well as how to compile it have been attached along this project.

As we have done in Lab#5, balancing the workloads among the computing nodes and using non-blocking send/receive can also improve the performance. In addition, the message size can affect the performance of a program since it affects the communication time. When non-blocking send/receive is used, the computation and communication can be best overlapped when the computation time and communication time are comparable.

You are required to re-write the matrix-matrix multiplication (A x B = C). A, B and C are *nxn* matrices. You program must

a. Utilize both ***MPI and Pthread*** programming techniques. While MPI is used to distribute tasks and collect results, multiple threads are created within each node to do the computation.

b. Take a ***master-slave*** model. All three matrices exist only on the master process at the start and finish of the computation. Each slave process should create more than one thread to do the computations, and then return results to the master.

c. Use the ***self-scheduling*** scheme to balance the workloads: The master process manages a pool of available tasks. Whenever a slave process has no work to do, it requests an available work from the master process until all work is completed.

d. Utilize ***non-blocking*** send and receive to overlap computation and communication.

e. Be able to ***adjust the size*** of the message (data) transmitted between the master and slaves so that the computation and communication can be best overlapped.

f. If n <= 16, your program should print out the matrix A, B, and C.

g. Use the master process to record the total computation time. Set the starting timer right after initializing the matrices and the ending timer right after collecting the computation results from all slaves.

Project Report - your final project report must contain at least the following components:

1. Introduction – introduce your project and provide relevant background.
2. Implementation – introduce how you parallelize the algorithm as well as how you implemented it.
3. Performance analysis – compare and explain the performance you achieved. This should be the major part of your report. You need to address at least the following questions ***with supporting data***:
    1) Given that the matrix size is fixed (e.g. n=512), what is the scalability (number of nodes again speedup) of your program?
    2) How do different matrix sizes affect the scalability of your program?
    3) Given fixed number of processes (e.g. 8 processes), how does the number of threads with each process affect the performance?
    4) How does the message size (or data size – amount of data transmitted per send/receive) affect the performance?
    5) Compare the performance of your program with previous labs (i.e., lab#3, 5, 6) and describe how different techniques affect the performance.
4. Discussion and conclusion – discuss relevant issues or difficulties you encountered as well as their implications in the context of parallel computing. Make a conclusion.
5. Contributions: describe each member's contributions to your project.

Project groups:

Group 1: Bek, Kevin R., Deel, Collin R., Dewey, Gregory R.

Group 2: Costello, Zachary J., Interrante, Nicholas R., Kanda, David P.

Group 3: Harris, Alexander D., Pacheco, Tyler N., Rumbaua, Marvin Jake M.

Group 4: King, Christopher J., Rahm, Daniel J., Wendt, Nathaniel A.

Submit your work – zip all your work into one file and email it to ji@gonzaga.edu no later than 11:59PM on May 1. Your file must contain the following documents:

1. A *readme* file that describes the running environment of your program as well as how to compile and run your program.
2. Your final project report.
3. Your project package including your source code.
4. Other documents that may help understand your project.

Schedule:

May 1 - Final presentation (1:10pm in HK202): each group has 5-10 minutes to present their work, and give a demo of their program.

May 1 (11:59pm) – the final zip file is due.

Grading (total 15 points)

Presentation – 2 points

Project report – 6 points

Quality of code, comments, overall performance – 7 points