COMP 605: Introduction to Parallel Computing Quiz 3: Module 3 Quiz: Comparing OpenMP and MPI Matrix-Matrix Multiplication

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Due: 05/12/17
Updated: 05/04/17

Table of Contents

Comparing OpenMP and MPI Matrix-Matrix Multiplication

- Objective:
  - Develop and test an OpenMP Mat-Mat-Mult code
  - Compare your OpenMP results to MPI reference data
  - All input and source code can be found in */COMP605/quiz3/*
- Serial mat-mat-mult: a copy has been place in quiz4 directory
- For MPI mat-mat-mult:
  - You do not need to run any MPI code.
  - You can use the reference data provided in the *quiz3* directory
  - MPI version in Pacheco text, Parallel Programming with MPI, Ch7.
- OpenMP mat-mat-mult:
  - Your may write your own code, or modify existing code.
  - Working OpenMP source code available in */COMP605/quiz3/*:
    - Blaise Barney (LLNL): *omp_mm.c*
      `https://computing.llnl.gov/tutorials/openMP/samples/C/omp_mm.c`
    - John Burkhardt (FSU): *mxm$_o$penmp.c*
      `https://people.sc.fsu.edu/~jburkardt/c_src/mxm_openmp/mxm_openmp.html`

Programming Instructions

- Generate input matrices A & B from within code
- All key variables and filenames read from command line
- Matrix size N should be dynamic
- Matrices can be square $[N_i \times N_i]$
- Vary #Threads
- All jobs should be run using batch scripts
- For *small* test cases (< 10), include logic to print out examples of A, B, and C.
- Vary #OpenMP threads: Recall: OpenMP number of threads is defined by hardware.

Performance

- keep track of what node/core you used
- bind the processes
- Timing:
  - Time critical blocks
  - Compare MPI CPU to GPU timings.
- What is the largest #threads you were able to test? What happened, why do think this happened

Suggestions on what to Report/Turn in for both problems:

- Create the homework directory *USER/quiz/q3* with correct access permissions.
- Short lab report with comments, figures and table labels.
- Explain your results for Thread and ProbSize scaling.
- Include relevant tables of your test data
- Evidence you ran your jobs using the batch queue (short/small job); examples of batch scripts
- Plots of key results.
- A copy of your code (single spaced, two sided, two column format is OK).
- Reference key sources of information *in your report and code* where applicable.

Comparing MPI and OpenMP

- You cannot directly compare scaling for MPI #cores against OpenMP number of threads.
- You can compare common run-time characteristics and variables:
  - All runs can have same (or close) problem sizes
  - All runs can be timed
  - Identify $T_{optimal}$ for each programming model:

    $T_{optimal}$ is defined as the point where increasing the number of processors or the number of threads/block no longer significantly reduces the run-time ('turnover' point).
- Figures 1 & 2: determining $T_{optimal}$ for MPI and CUDA programming models.
- Figure 3: comparison of $T_{optimal}$ for the two programming models .
- Figures 4-6: MPI reference data provided for this assignment.
- Note: Figures are *not* for mat-mat-mul, so your data values may differ, but the trends should not change.

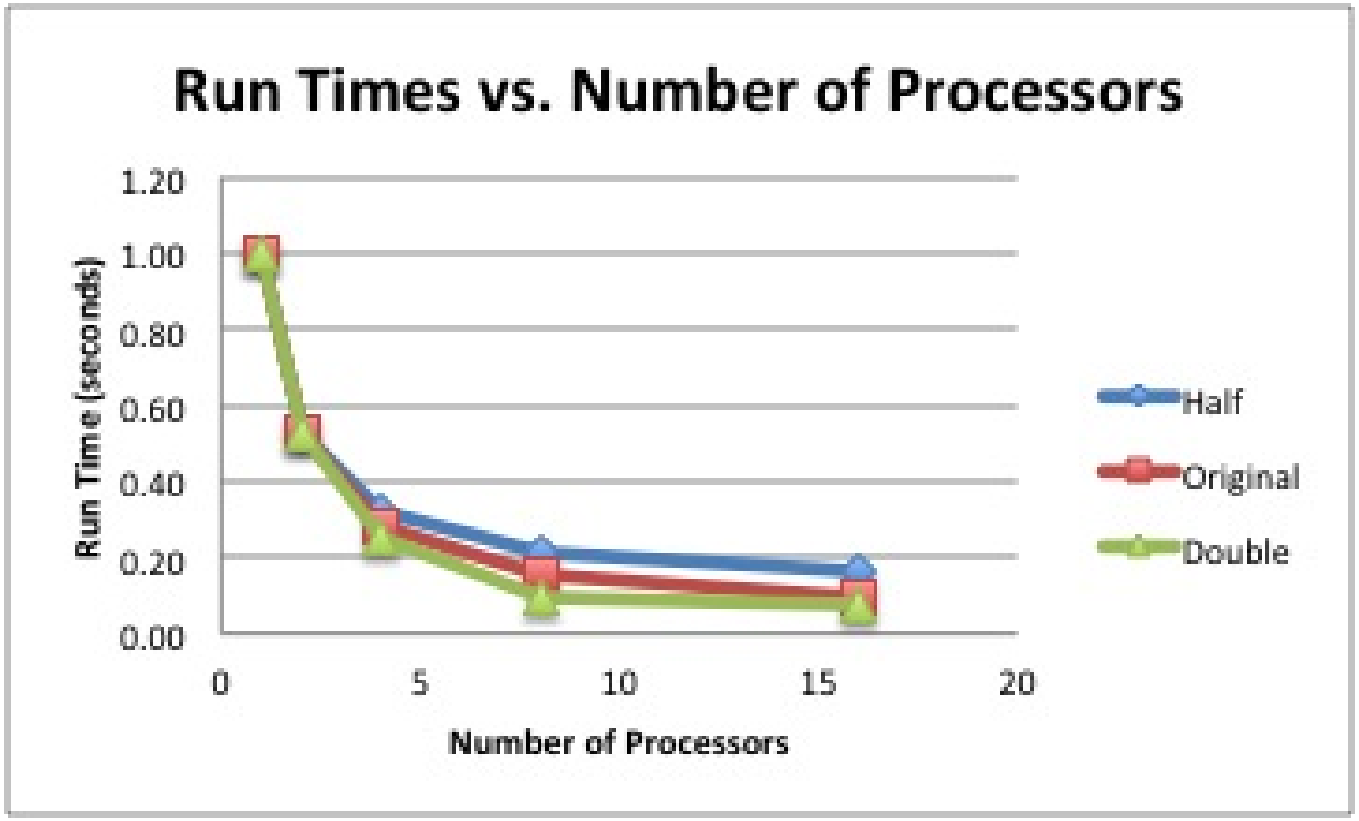

Figure 1: The figure above shows the run-time as a function of the number of processors, for different problem sizes,using MPI. The run time decreases as the number of cores increases, up to a limit where there is not much improvement. In this case, $T_{optimal}$ 16 cores

Example Comparing MPI and OpenMP

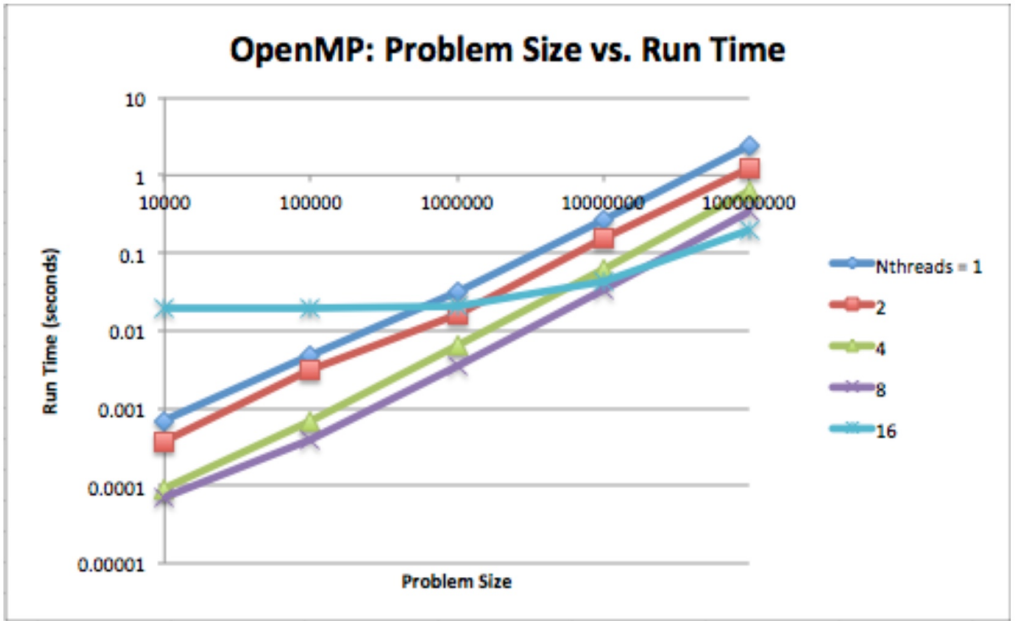

Figure 2: ProbSize vs. Run-time for OpenMP (upper). Using more threads decreases the run time, the desired result, bur for small ProbSizes, using a large number of threads slows down the calculation. Note that the maximum number of possible threads should be 32 since this is shared memory and our largest nodes have 16 cores.

Source: H. Hirakawa, comp 605, Spring, 2016

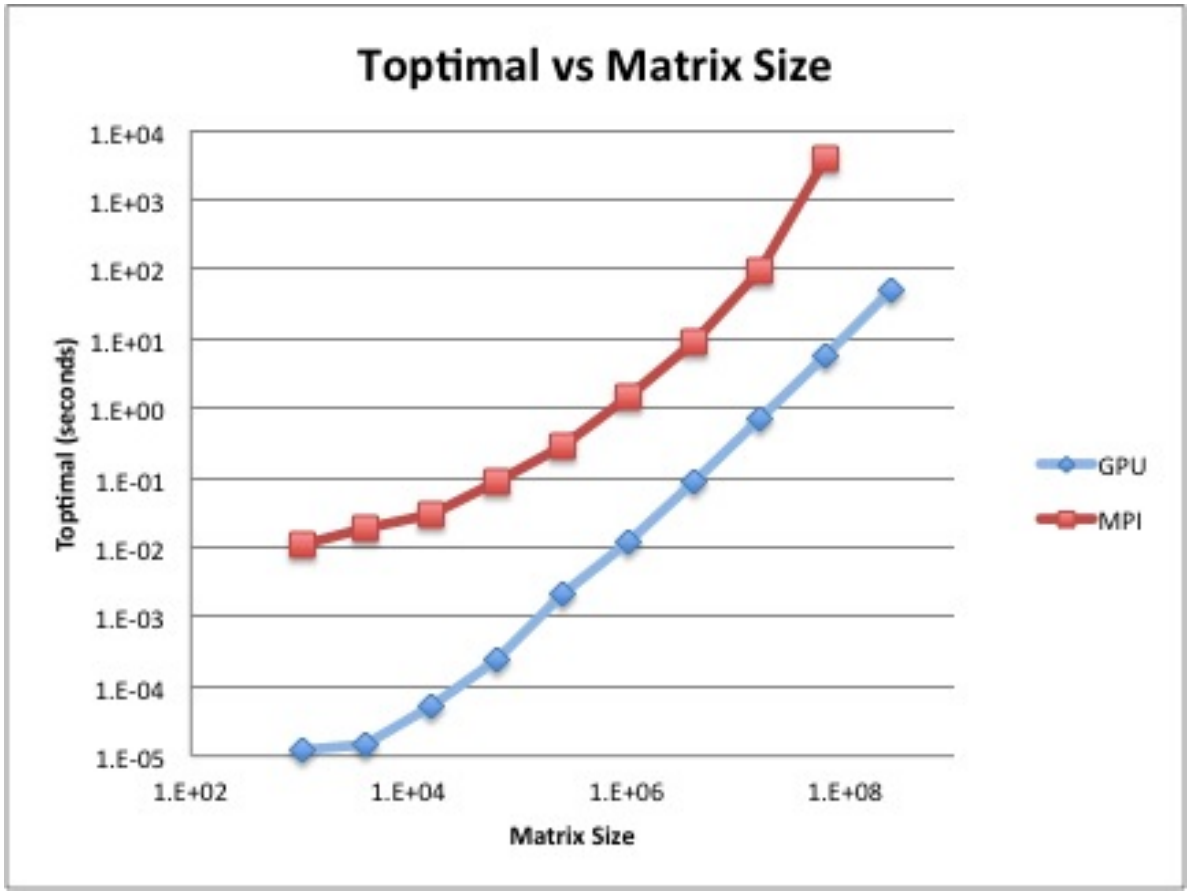

Figure 3: $T_{optimal}$ as a function of matrix size for MPI and CUDA/GPU tests. The figure above shows that for a given problem size, $T_{optimal}$ for the GPU programming model is better that MPI. This problem is for a matrix-matrix multiplication problem.

Source: Fall 2012: G. Pham

MPI Matrix-Matrix Multiplication ref data

Cores	420	1260	1680	2520	3360	4200	5040
1	7.78E-01	2.56E+01	6.91E+01	2.34E+02	4.98E+02	1.09E+03	1.88E+03
4	1.96E-01	5.61E+00	1.37E+01	5.85E+01	1.37E+02	2.71E+02	4.66E+02
9	1.10E-01	2.35E+00	6.84E+00	2.39E+01	5.91E+01	1.25E+02	2.10E+02
16	6.28E-02	1.64E+00	3.68E+00	1.38E+01	3.30E+01	8.05E+01	1.39E+02
25	4.88E-02	1.45E+00	3.52E+00	8.23E+00	3.00E+01	6.79E+01	8.67E+01
36	9.40E-02	9.70E-01	2.43E+00	8.23E+00	2.20E+01	4.08E+01	7.58E+01
49	4.01E-02	7.08E-01	1.98E+00	4.59E+00	1.25E+01	2.49E+01	3.99E+01

Figure 4: MPI Matrix-Matrix Multiplication ref data. The Table shows the runtime (in seconds) as a function of the number of processors (Cores) vs matrix size M, for matrices of dimension [MxM].

MPI Matrix-Matrix Multiplication ref data

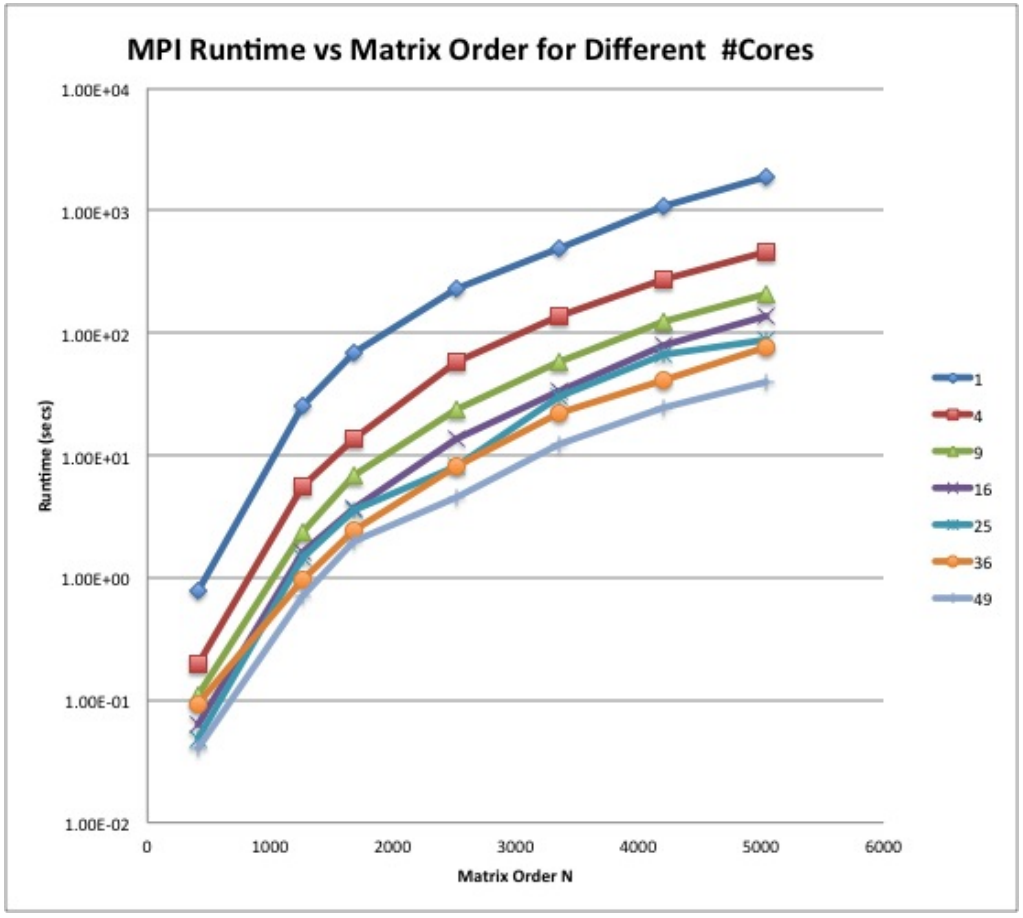

Figure 5: MPI Matrix-Matrix Multiplication ref data: Curves show the runtime (in seconds) as a function of the matrix size for different number processors.

MPI Matrix-Matrix Multiplication ref data

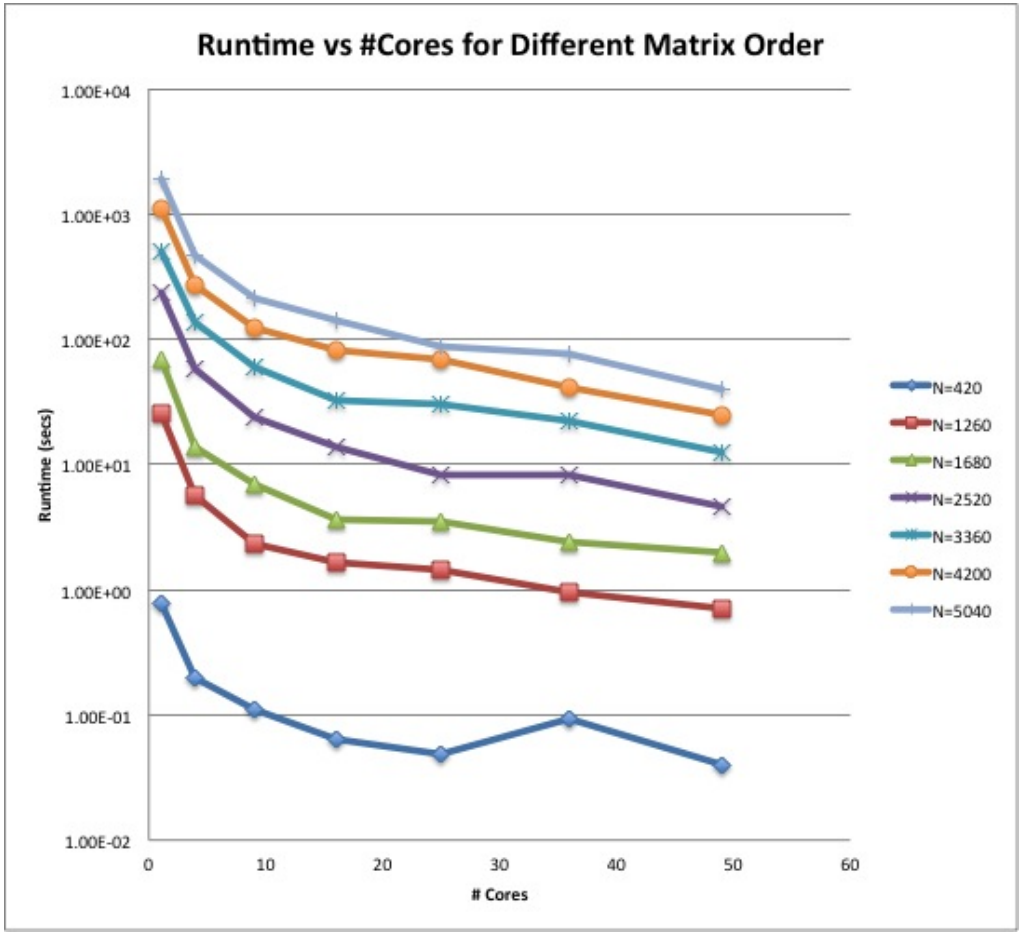

Figure 6: MPI Matrix-Matrix Multiplication ref data: Curves show the runtime (in seconds) as a function of the number of processors for different matrix sizes.