# Homework 5: Cannon's Algorithm for Matrix Multiplication

Han Tran

The code is attached to this homework submission. This code is to do the matrix multiplication C = A*B using Cannon's algorithm. The matrix A and B is partitioned into *p* processes which is distributed in a square grid of *q* by *q* (i.e. $q = \sqrt{p}$). Each process only stores its portion of the matrices A and B. The result matrix C is also partitioned and stored in *p* processes.
For each process, the size of matrices that it stores and handles the multiplication is:

$$C = (m \times n),\ A = (m \times k) \text{ and } B = (k \times n).$$

To test the correctness, the code is run with small matrices, e.g. $m = n = k = 2,\ \ m = n = 2;\ k = 3$. All cases give correct results.

1) To compare a pure MPI version (28 tasks/node, 1 OpenMP thread/task) with the hybrid version (2 tasks/node, 14 OpenMP threads/task): the size of matrix C is kept constant $C = (20{,}000 \times 20{,}000)$. Note that *m*, *k* , *n* are the sizes of the sub-matrices stored in each process. If we multiply these sizes with the number of process, we can get the size of the global matrix C. For example, running with *p* = 49 processes (i.e. *q* = 7 processes in each side), then the size of global matrix is $7 \times 2858 = 20{,}006$.

| threads | nodes | p | task/socket | task/node | m | k | n | time |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 49 | 13 | 26 | 2858 | 2858 | 2858 | 2426.266463 |
| 14 | 2 | 4 | 1 | 2 | 10000 | 10000 | 10000 | 2152.454609 |

It can be seen that, for the same problem size of $C = (20{,}000 \times 20{,}000)$, the pure MPI version takes longer time than the hybrid version. This could be explained by the communication cost of pure MPI version compared with hybrid version.

2) To perform the strong scalability, I keep the size of matrix $C = (20{,}000 \times 20{,}000)$ fixed, and run with increasing number of processes. The results are shown as below.

| threads | nodes | p | task/socket | task/node | m | k | n | **Time** (s) |
|---|---|---|---|---|---|---|---|---|
| 14 | 2 | 4 | 1 | 2 | 10000 | 10000 | 10000 | 2152.454609 |
| 14 | 5 | 9 | 1 | 2 | 6667 | 6667 | 6667 | 2034.593798 |
| 14 | 8 | 16 | 1 | 2 | 5000 | 5000 | 5000 | 825.458194 |

It is seen that the code is strong scalable, i.e. the computing time decreases when number of processes increases.

3) To perform the weak scalability, I keep the ratio size/p be constant (= 5000). Following is the results.

| threads | nodes | p | task/socket | task/node | m | k | n | **Time** (s) |
|---|---|---|---|---|---|---|---|---|
| 14 | 1 | 1 | 1 | 2 | 5000 | 5000 | 5000 | 256.391463 |
| 14 | 2 | 4 | 1 | 2 | 5000 | 5000 | 5000 | 661.5608249 |
| 14 | 5 | 9 | 1 | 2 | 5000 | 5000 | 5000 | 839.2447641 |
| 14 | 8 | 16 | 1 | 2 | 5000 | 5000 | 5000 | 825.458194 |

It is seen that the time increases when number of tasks increases. This could be explained as the communication also gets more when the number of tasks increases.