# CPSC 424 | HW3

## kd538 - Keyang Dong

# Task 1

```
Matrix multiplication times:
   N      TIME (secs)
 -----   -------------
   1000        0.1817
   2000        2.2703
   4000       19.2888
   8000      153.1305
```

# Task 2

Here `cpt` stands for `computational time (secs)`, `cmt` stands for `communication time (secs)`.

**Important**: all displayed time, namely cpt and cmt, are timed by Master process (rank 0), which usually shoulders least computational workload if without load balance strategies applied.

| procs | N | cpt | cmt |
|---|---|---|---|
| 2 | 1000 | 0.0624 | 0.0868 |
| 2 | 2000 | 0.5991 | 0.9757 |
| 4 | 4000 | 1.8141 | 12.1807 |
| 8 | 8000 | 5.5364 | 78.8701 |

(Submit using script `build-run.sh`)

(As blocking send & recv in single process will cause deadlock in my code, use 2 processes instead for testing purpose.)

### For procs=8 only:

| tpn | N | cpt | cmt |
|---|---|---|---|
| 4 | 4000 | 0.2834 | 4.8810 |
| 2 | 8000 | 2.4479 | 37.0018 |

(Submit using script `build-run-p8-n4.sh` & `build-run-p8-n2.sh`)

(When testing timing for tpn=2, N=8000 case, sometimes deadlock still arises. In these case, just `qdel` the process and start again, and mostly the problem could be fixed.)

# Conclusions draw from runs of p=8

By spread workload to multiple physical nodes, rather than doing context switching between various processes within a single node, computation could be significantly expedited, while the communication time reduced greatly.

# Discussion

1. The raw performance itself is not ideal, as the clustered program spend most of its run time waiting for others to send next piece of data it requires, or waiting for its successor to receive the piece of columns it has already used. Computation and communication is not sufficiently overlapped, and CPU time is wasted during blocked waiting as well as context switching between processes.
2. The scalability is poor. Both cpt & cmt grows nearly linearly w.r.t. size of data (N*N, or O(N^2)), quadratically to N.
3. Load balance is not something that has been well designed to achieve. As the partition of rows into row-groups is continuous and fixed, the first several sections always shoulder less computational tasks as comparing to later sections, leading to longer wait time (previous sections wait for later sections to finish its work).
4. There's deadlock problem upon applying blocking MPI operations, and finesse programming techniques have to be used to eliminate them. The strategy I used is, for each process with an even rank, receiving the columns from its predcessor to its local buffer, sending its own column-group to successor, than execute partitioned computation; for those with an odd rank, order of recv & send is reversed to make a dual.

# Suggestions

1. Use non-blocking operations to overlap new column-group receiving with computational task; this will shorten the predominant cmt time, yielding better performance. This will also get rid of deadlock

problem.

2. Separate processes on different nodes to achieve better scalability and less overhead in context switching.

3. Balance the partition of rows to different processes, so that each unit of calculation will be relatively consume same level of time. For example, pick rows from front and rear using two pointers, rather than just from one end.

# Task 3

| procs | N | cpt | cmt |
|---|---|---|---|
| 4 | 4000 | 1.7937 | 12.3106 |
| 8 | 8000 | 6.0632 | 78.8786 |

(Submit using script `build-run.sh`)

| tpn | N | cpt | cmt |
|---|---|---|---|
| 4 | 4000 | 0.5174 | 5.1062 |
| 2 | 8000 | 5.4667 | 28.7481 |

(Submit using script `build-run-p8-n4.sh` & `build-run-p8-n2.sh`)

The raw performance does not upgrade too much, mainly because: (1) load is still not balanced, master still needs to wait for others complete their works; (2) context switching on single node still costs much computational resource; (3) simplified logic leads to relatively lower performance than that of used finesse deadlock prevention strategies.

But as we can see, the scalability outperforms blocking operation version.

# Task 4

As I mentioned in the suggestions, I used a different way to partition rows (and for generalization purpose, also columns). I use two pointers, namely i and j, set them to i=0, j=N-1, and keep add corresponding row to current process' partition while incrementing i and decrementing j. Constructing of each partition ends when its volume get close to the average size, which is int((N-1)/np) + 1, and the last partition might be a little shorter than others.

| tpn | N | cpt | cmt |
|---|---|---|---|
| 4 | 4000 | 2.7771 | 2.1769 |
| 2 | 8000 | 21.1236 | 10.9459 |

(Submit using script `build-run-p8-n4.sh` & `build-run-p8-n2.sh`)

As we can see from the runtime stat, the improvement is obvious. Computational time as timed by master process arises, because it now takes as much partitioned matrix multiplication task as other processes do; but communication time significantly decreases, cuz they no longer have to let others to fulfill more jobs, and themselves just wait.

It also scales well to large datasets, as long as the computation is well distributed to different nodes, and the communication overhead just grows quite smoothly.

# Task 5

I don't have to make any adjustment to the program in Task4, just write up a new submission configuration `build-run-p7.sh`, and the runtime is as following:

| procs | tpn | N | cpt | cmt |
|---|---|---|---|---|
| 7 | 2 | 7633 | 38.3130 | 10.4230 |

# P.S. Debugging Notes

Most mysterious `Invalid Rank` or similar unroutable problem are caused by invalid memory rewriting, especially the use of `memcpy`, or too small buffer or cache area for message passing as well as temporal storage.