# Comp3036J Parallel and Cluster Computing

---

| | |
|---|---|
| **Assignment Type:** | Code and report |
| **Project Date:** | 22/April/2024 |
| **Assignment Compiler:** | Eddie Antonio Santos |
| **Weighting:** | 50% |
| **Due Date:** | 14/May/2024, 23:59 PM Dublin time |
| **Method of Submission:** | Moodle (.pdf and .c file) and beckett4 (.c file) |

1. Link is on Moodle.

2. Please place solution.c in your home directory on beckett4 in a folder called assignment_1 so that it can be accessed at: `/home/`*your-username*`/assignment_1/solution.c`

*Do not edit the file after the due date or the file date on beckett4 will indicate a late submission.*

---

## Task:

In tutorial 3, you saw how to use MPI collective communication to split a calculation and distribute it among several processors. However, these are just convenient wrappers that MPI provides for you. Collective communication is ultimately implemented in terms of point-to-point communication.

Your task: **Implement MPI_Reduce using only point-to-point communication**. Explain how this is possible by writing a short (1 or 2 page) document and provide the C code to accomplish it.

In the provided file `solution.c`, there is call to **MPI_Reduce** that sums the menu prices from all processes in the communicator to rank 0 (the root process). **Remove the call to MPI_Reduce** and **replace it with calls to MPI point-to-point communication** (**MPI_Send** and **MPI_Recv**) as appropriate to ensure that the sum of all items will be present on the root node. Assume that all processes (including the root node!) have partial sums that must be summed to contribute to the grand total.

1. Code
    - Modify solution.c to **use point-to-point communication instead of MPI_Reduce**. You must submit code that compiles with the following command line:
        - **`mpicc -std=c11 -Wall -Werror -O2 -o solution solution.c`**
        - Submissions that do not successfully compile with the above command will be penalized.
    - Your submission must perform the parallel reduction **using point-to-point communication** and it must successfully compute the sum using the partial solutions calculated on all $p$ processes (including the root node).
    - The grand total must be finalized on the root node (process of rank == 0)
    - The reduction must make use of the size of the communicator (that is, the total number of processes) from MPI. You should not use a constant, hardcoded number.
    - Partial credit will be given if the solution works only in the case of $p = 2$ processes.
    - The final answer calculated by your version of solution.c that uses point-to-point communication must match the answer calculated by the version of solution.c that uses MPI_Reduce.

- Full credit will only be awarded if the code does all of the above and does so using the **minimum amount of time steps**, namely $\mathrm{O}(\log_2 p)$. Another way to think of "time steps" is **the maximum number of MPI_Send/MPI_Recv calls that any given node will need to execute** to complete the reduction. For example, to reduce on 8 nodes, you will need at most $\log_2 8 = 3$ time steps, or in other words, a single node cannot perform more than 3 MPI_Send/MPI_Recv calls. **Hint**: write code to do the reduction in $O(p)$ time steps, then **if and only if that soultion works**, try attempting the optimal solution. **Hint**: you may assume that $p$ is a power of two.

2. Report (max 2 page report)
    - Write a report describing how you implemented solution.c. You must provide:
        i. Explanation of the approach chosen.
        ii. An explanation of the required communication involved in MPI_Reduce.
        iii. Any specific choices you made in your implementation.
    - Report: use 11pt font (Linux Libertine, Computer Modern, or Times New Roman).

# Provided files

We will provide the following files

- `solution.c` — MPI program that calculates the total from `order.bin`. **You must modify this file and submit it**.
- `make-order.c` – compile and run this to create `order.bin`
- `menu.tsv` — provides prices for each menu item

# Deliverables

Two files:

- Design document, at most 2 pages, PDF format, uploaded to Moodle before the deadline.
- `solution.c`, with your modifications, uploaded to Moodle before the deadline, **AND** on beckett4 as described above.

# Questions

Questions should be asked on the Moodle Forum.

# Grading

You will be graded on the following:

- Code correctness (50%). Full marks will be awarded if your submission does all of the following:
    - Computes sum correctly using exactly 2 processes
    - Computes sum correctly using $p$ processes, where $p > 2$
    - Computes sum correctly using $p$ processes, where $p$ is given by MPI
    - Computes sum correctly using $p$ processes, where $p$ is given by MPI, using only $\mathrm{O}(\log_2 p)$ time steps. You may assume that $p$ is a power of two.

  Programs will be awarded partial credit in the presence of minor bugs or syntax errors — as long as the intent is clear!

- Report (50%). Partial credit will be awarded as appropriate, including all specifications above and:
    - Demonstrates a fundamental understanding of communication required
    - Explanation of algorithm implemented in C code
    - Documentation of implementation choices
    - Attention to detail
    - Overall professionalism of the report

# Plagiarism

When you submit your assignment, you will get a URKUND plagiarism report. Generally, all submissions will have a plagiarism score of <~10. This is normally ok. Ultimately, I will look at these "by hand" so the number, by itself, doesn't mean that much. More important to a high score is WHY IS IT HIGH.

I encourage you to submit early, and if you think your score is too high, revise and resubmit. A score of up to ~20 can happen due to legitimate references, etc.