Università della Svizzera italiana

Institute of Computing
CI

High-Performance Computing Lab **Institute of Computing**

Student: Kaan Sen Discussed with:Francesco Gnan

Solution for Project 7

1. Parallel Space Solution of a nonlinear PDE using MPI [in total 35 points]

1.1. Initialize and finalize MPI [5 Points]

We used the **MPI_Init** thread to initialize MPI as well as **MPI_Comm_rank** and **MPI_Comm_size** for the rank and size respectively. Finally, we used **MPI_Finalize** to finalize MPI.

1.2. Create a Cartesian topology [5 Points]

To create a Cartesian topology we first created the dimensions of the decomposition using **MPI_Dims_create** followed by a non-periodic Cartesian topology using **MPI_Cart_create**. We get the coordinates of the current rank using **MPI_Cart_coords** and the neighbours using **MPI_Cart_shift**

1.3. Extend the linear algebra functions [5 Points]

In the linalg.cpp we used **MPI_Allreduce** to compute the dot product and the norm

1.4. Exchange ghost cells [10 Points]

In the operators.cpp we implemented point-to-point communication for all neighbours in all directions for the exchange of ghost cells using **MPI_Irecv** and **MPI_Isend**

1.5. Scaling experiments [10 Points]

The figures 1 2 are the results for the strong scaling experiments, it was not possible to carry out scaling for more than 8 processes on a single compute node so those results are not included. It is evident, when compared to the results of the OpenMP implementation of Project 3 that using multiple processes using MPI is much more effective at reducing the time required for the calculations.

2. Python for High-Performance Computing (HPC) [in total 50 points]

2.1. Sum of ranks: MPI collectives [5 Points]

We implemented two different forms of carrying out the sum of ranks task in two different files:

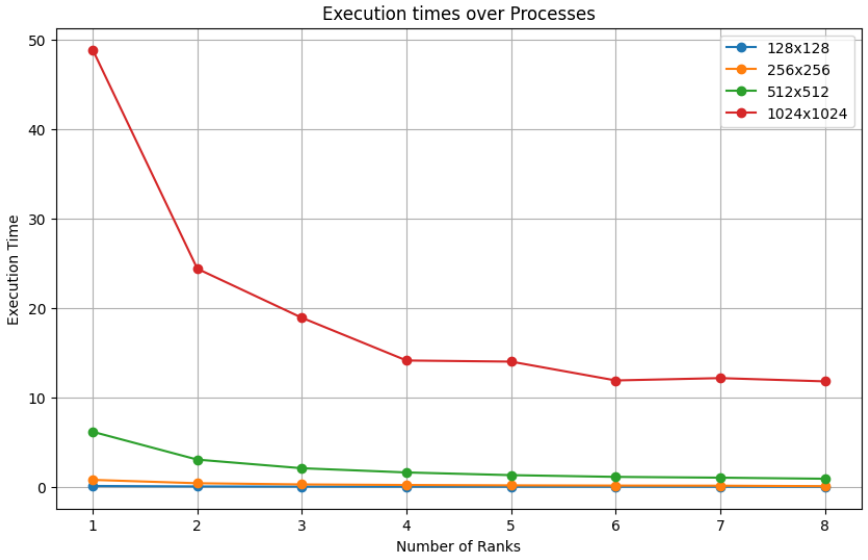

Figure 1: Strong Scaling for ghost cell exchange

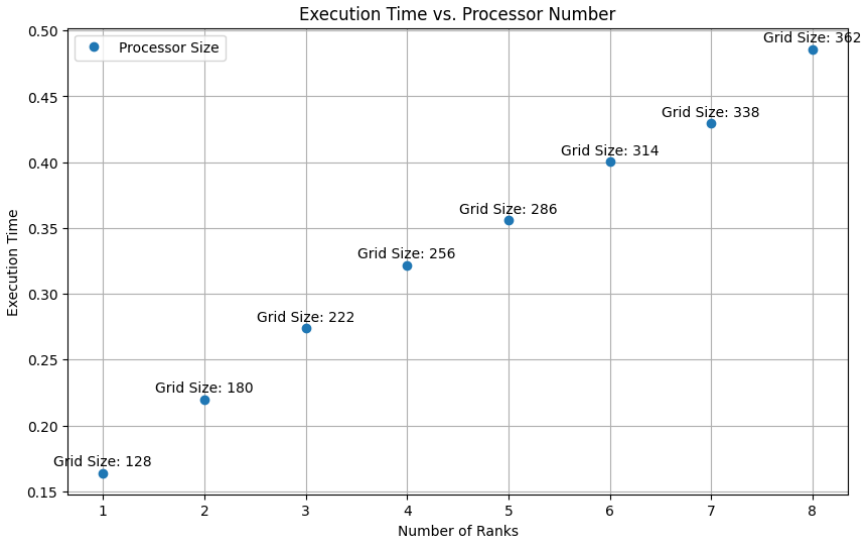

Figure 2: Weak Scaling for ghost cell exchange

- Using the pickle-based communication of generic Python objects, i.e. the all-lowercase methods in file 2_1_1.py
- Using direct array data communication of buffer-provider objects, i.e. the method names starting with an uppercase letter in file 2_1_2.py

2.2. Domain decomposition: Create a Cartesian topology [5 Points]

To create a Cartesian topology we first created the dimensions of the decomposition using **MPI_Compute_dims** followed by a non-periodic Cartesian topology using **Create_cart**. We get the coordinates of the current rank using **Get_coords** and the neighbours using **Shift**. This is implemented in file 2_2.py

2.3. Exchange rank with neighbours [5 Points]

The exchange of rank with neighbours is implemented in 2_3.py

2.4. Change linear algebra functions [5 Points]

We used numpy arrays to carry out the calculation for **hpc_dot** and **hpc_norm2** as they provided a much faster way than the traditional python for loops.

2.5. Exchange ghost cells [5 Points]

We implemented the exchange of ghost cells using the **Isend** and **Irecv** methods. The capitalization of the first letters indicates that these are buffer-based communication. The first method to carry out the exchange was **exchange_startall** where we calculated the appropriate values for each direction whereas the other method exchange **waitall** used the Request package. The obtained results were similar to what we did with C++ earlier.

2.6. Scaling experiments [5 Points]

The figures 3 4 represent the results of the strong and weak scaling experiments we carried out. It is obvious that the python version took significantly more time than the C++ version. Additionally, as is apparent from the figures, at a lower number of processes the scaling in Python is much less effective than it is in C++ but higher number of processes the behaviour becomes similar.

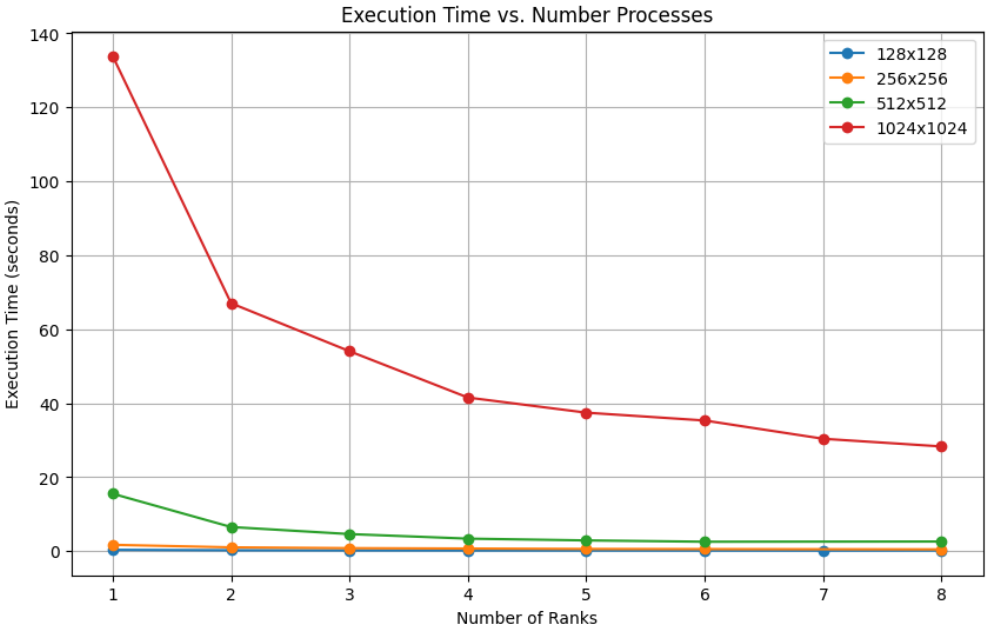

Figure 3: PDE application using MPI with Python Strong Scaling

2.7. A self-scheduling example: Parallel Mandelbrot [20 Points]

We implemented a manager and worker algorithm that uses non-blocking pickled send and receive to establish communication between the manager and worker to compute the values for the Mandelbrot set in the file **manager_worker.py** The scaling for 100 and 50 tasks over 2, 4 and 8 processes is represented in the figure 5. As is apparent, the addition of more processes has a better impact when the workload is split into the smaller 50 tasks instead of the larger 100 tasks.

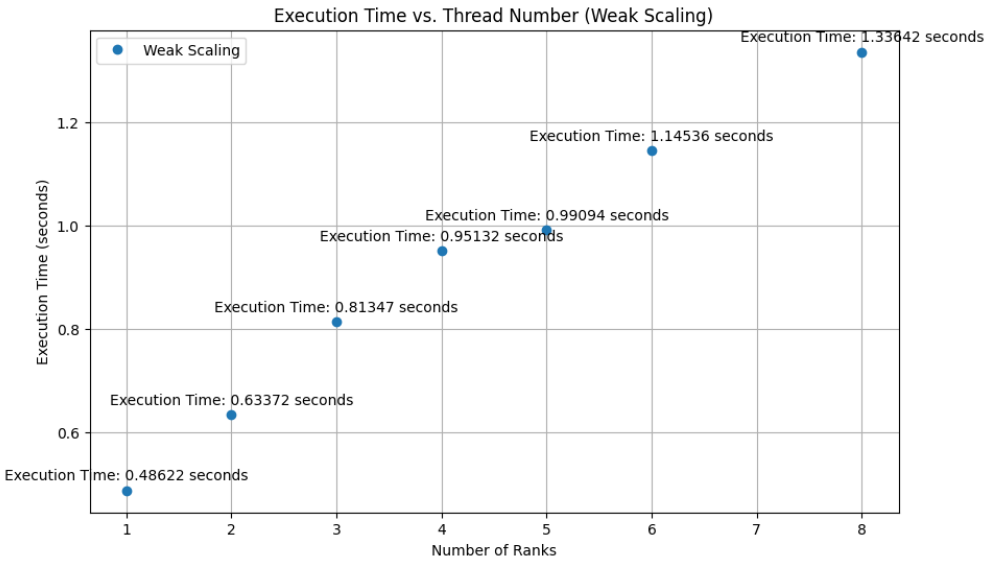

Figure 4: PDE application using MPI with Python Weak Scaling

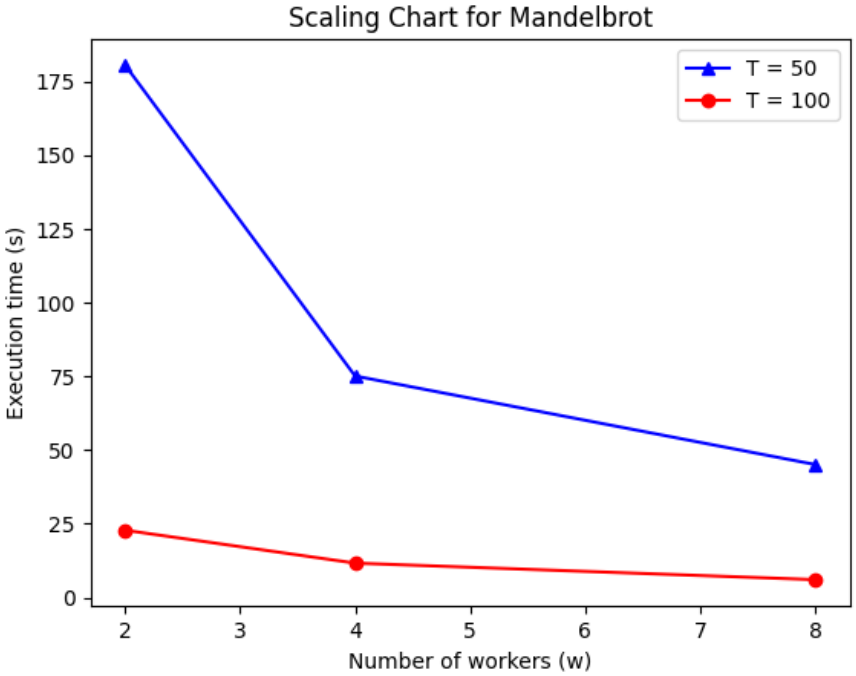

Figure 5: Scaling chart for different number of workers and tasks