Università della Svizzera italiana

Institute of Computing
CI

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

Student: Metehan Dündar Discussed with: Yusuf Eren Tunç

Solution for Project 4

1. Ring maximum using MPI [10 Points]

- Each process's left/right neighbors are determined by $(k \pm 1) \bmod p$, where k is the rank of process and p is the number of processes.
- Non-blocking communications are believed to avoid deadlock. In each loop, the non-blocking `MPI_Sendrecv`, function is called, which sends a send buffer and receives the result in a receive buffer.
- The content of message sent to the right neighbor, which is put in `snd_buf`, is copied from the `rcv_buf` received from the left neighbor in the last iteration.

2. Ghost cells exchange between neighboring processes [15 Points]

- The Cartesian communicator is formed using `MPI_Cart_create` with `dims={4,4}`, `periods={1,1}` and `reorder=0`. `MPI_Cart_shift` in both directions is used to find the proper neighbors.
- `MPI_Type_vector` with `count=SUBDOMAIN` and `stride=DOMAINSIZE` defines the derived data type for conveying column boundaries. `MPI_Sendrecv` is used as a method that exchanges information.
- Ghost cells go in all directions. To begin, each process transmits its top row to its top neighbor with `count=SUBDOMAIN` and receives a duplicate of its bottom neighbor's top border, saving the message in the bottom ghost cells. The lower rows are then delivered in the same manner. The left and right boundaries are communicated to the left and right neighbors in the derived data type, accordingly. This diagram depicts how ghost cells communicate information with nearby cells in all directions.

3. Parallelizing the Mandelbrot set using MPI [20 Points]

3.1. Implementation

- The Cartesian grid is computed using `MPI_Dims_create` in `createPartition`. The input `dims` are set to {`0,0`}, allowing MPI to decide dimensions of all directions.
  `MPI_Cart_create` with `reorder=0` is used to create the Cartesian communicator.
  `MPI_Cart_coords` is used to determine the coordinates of the process in the Cartesian grid. Since no reordering is allowed, the rank in `MPI_COMM_WORLD` is the same as that in the Cartesian communicator and thus used directly as the input of `MPI_Cart_coords`.

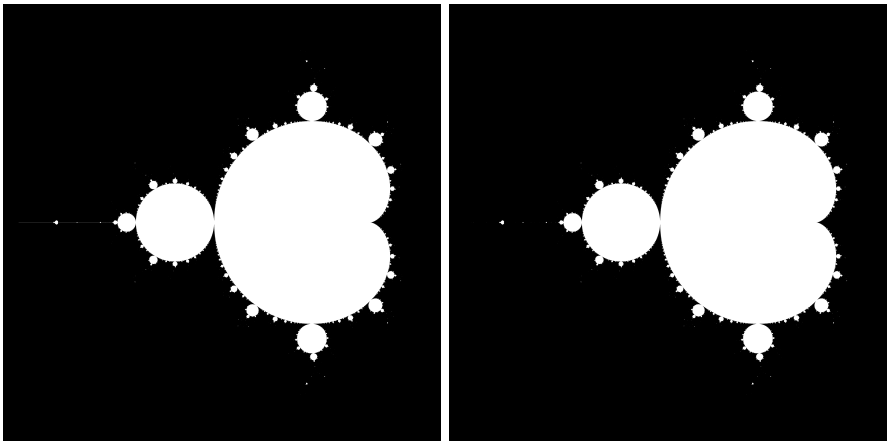

Figure 1: Result of Mandelbrot set computation with (left) and without (right) MPI

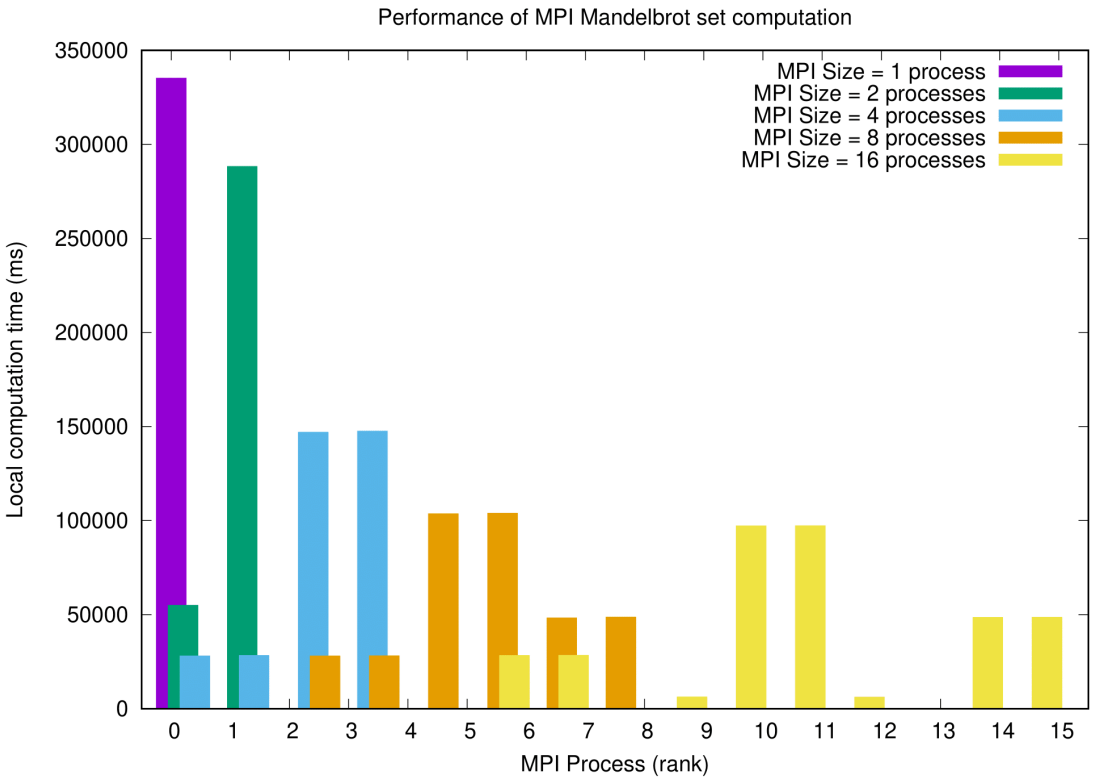

Figure 2: Performance of Mandelbrot set computation using MPI

- The range of the local domain is determined in `createDomain`. To partition the domain across processes evenly, when the size of the image is not a multiple of the size of the Cartesian grid, the remainders are distributed to the first several processes in each direction, instead of all being assigned to the last process in each direction.
- The local domain is sent to the master process via `MPI_Send` and received by the master process via `MPI_Recv`. Since the status of the received message is not needed, `MPI_STATUS_IGNORE` is used as the last argument of `MPI_Recv`. The outputs of the paralleled program and the sequential program are identical, which are shown in Figure 1.

3.2. Result and discussion

- The performance of a varying number of processes is shown in Figure 2.
- The maximum local computation time decreases as the number of processes increases. It indicates that MPI parallelization speeds up the computation time. However, we could not get better results with 12+ processors compared to 8- processors. To further improve the performance one could maybe partition the workload better between the different processors.
- The run time of the parallel version is determined by the maximum height of columns in each execution. It can be seen from Figure 2 that the performance of running on multiple processes is not scaled perfectly. Since each process is responsible for the same number of pixels, the computation time of the local domain is determined by the total number of iterations of pixels. The black area in Figure 1 means the computation of these pixels terminated ahead of the maximum number of iterations is reached, while the white pixels correspond to the computation for `MAX_ITERS` iterations. It results in load imbalance among processes and is reflected by the different heights of columns in Figure 2.

4. Option A: Parallel matrix-vector multiplication and the power method [40 Points]

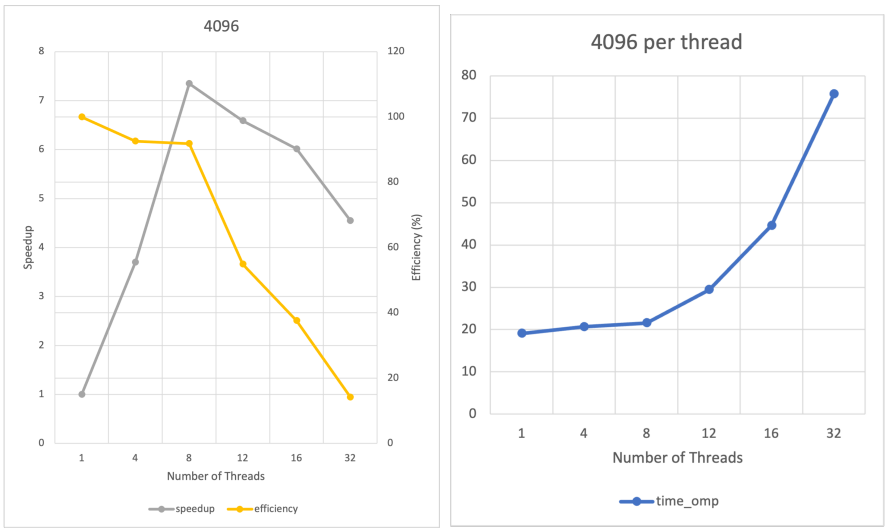

Figure 3: Stong Scaling & Weak Scaling

- In the `powermethod.c` file it is needed to implement different functions as stated in the task. In the main function, a matrix is generated and called the function `powermethod`. To stop the time and calculate effectively the `hpc_timer` function is called right before and after.
- The implementation of the `norm` function was pretty straightforward. To implement the `powermethod` function, a vector is initialized with size N with random values. Then in the loop where the vector is normalized and calls the function `matVec`. The last function to implement was the `matVec` function where the parallelization is implemented. Then the vector is broadcasted to all processes using `MPI_Bcast`. A vector is allocated on each process and computed as the matrix-vector multiplication. The result was then stored in the local vector. Finally, all the local vectors is gathered into a final result vector using `MPI_Gather`.
- The results for the strong scaling and the weak scaling are in Figure 3 above. In the strong scaling plot the time cuts in half when the number of processors is doubled.