Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing **2021**

Student: Ashutosh Singh Discussed with: Self

Solution for Project 7

Due date: 22.12.2021, 23:59

HPC 2021 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

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

1.1. Initialize and finalize MPI [5 Points]

I initialized MPI before getting rank and communicator size in `main.py` . With

```
MPI_Init_thread(&argc, &argv, MPI_THREAD_FUNNELED,&threadLevelProvided);
MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
```

1.2. Create a Cartesian topology [5 Points]

Creating a Cartesian involves guessing the slots in each dims given the number of processes using `MPI_Dims_create` then using the dims to create cartesian topology using `MPI_Cart_create`.

```
MPI_Dims_create(mpi_size, 2, dims);
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm_cart);
```

Then using `MPI_Cart_shift` get the rank of neighbouring processes in all 4 directions.

```
MPI_Cart_shift(comm_cart, 1, +1, &neighbour_west, &neighbour_east);
MPI_Cart_shift(comm_cart, 0, +1, &neighbour_south, &neighbour_north);
```

1.3. Extend the linear algebra functions [5 Points]

Collective operations using `MPI_Allreduce`

```
MPI_Allreduce(&result, &result_global, 1, MPI_DOUBLE, MPI_SUM, data::domain.
  comm_cart);
```

This statement collects the value of `result` from processes into `result_global` of all processes using the `MPI_SUM` op.

1.4. Exchange ghost cells [10 Points]

To exchange information between multiple processes asynchronously I use `MPI_Isend` and `MPI_Irecv` with proper tags to make the processes communicate properly.

The data(first/last row/column) is first copied into respective buffers and then using the above mentioned MPI functions sent to their destinations. Example sending data to `north_neighbour`

```
for(int i=0; i<nx; i++){
          buffN[i] = s(i, ny-1);
     }
MPI_Isend(&buffN[0], nx, MPI_DOUBLE, domain.neighbour_north, domain.rank, domain.
  comm_cart, &request);
```

The data incoming from a neighbour is sored in the respective boundary variables. Example receiving data from `north_neighbour`

```
MPI_Irecv(&bndN[0], nx, MPI_DOUBLE, domain.neighbour_north, domain.neighbour_north,
  domain.comm_cart, &request);
```

Finally before calculation for boundary values we need to wait for all send and receive requests to complete. In total there are 8 request,. 4 send requests and 4 requests. I use `MPI_Waitall` to do this.

1.5. Scaling experiments [10 Points]

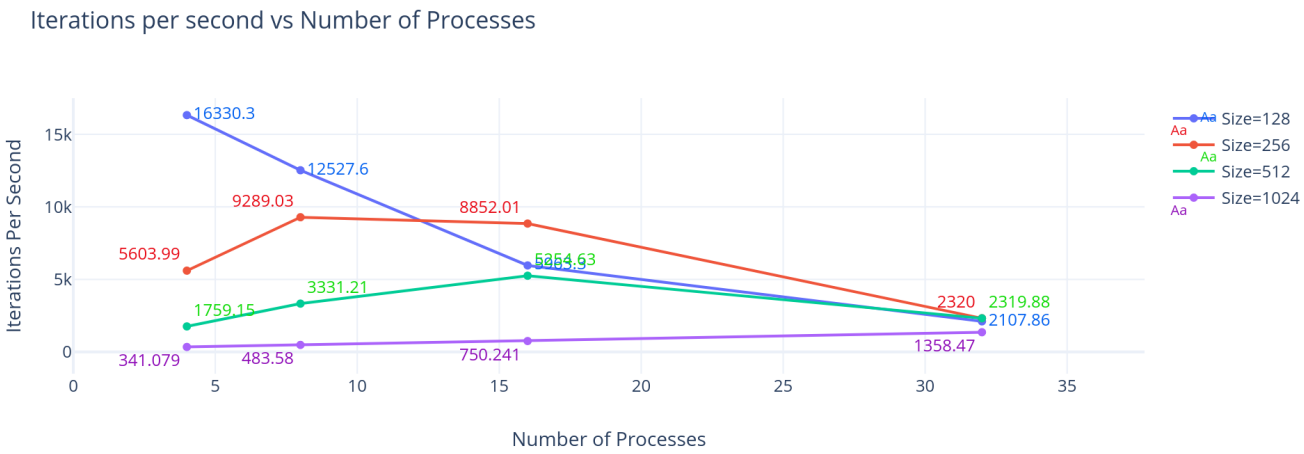

Figure 1: Iterations per second vs Number of processes for multiple Grid Sizes. Note: Till 16 processes only one node is used for running with 32 processes I used 2 nodes. Click here for Interactive figure

From Figure-1 it is clearly visible that for grid-size=128(blue line) the number of iterations per second decrease as processes are increased beyond 4; this could be explained by the argument that overhead of communication becomes comparable or even greater than the speed/parallelization attained.

As grid size is increased the number of iterations first increase for example for grid-size=256 an increase till 8 processes and for grid-size=512 an increase till 16 processes; then decrease afterwards. Again by the same argument after an optimal number of processes the communication overhead deteriorates the efficiency.

For grid-size of 1024 we do not see any decrease till 32, maybe we are not beyond optimal number of processes for this size. I do not report results beyond those many processes.

NOTE: Use the following scripts -

1. run_n32N2.sh - 32 processes on 2 Nodes
2. run_n4-16N1.sh - Use 1 node and run for 4 till 16 processes.

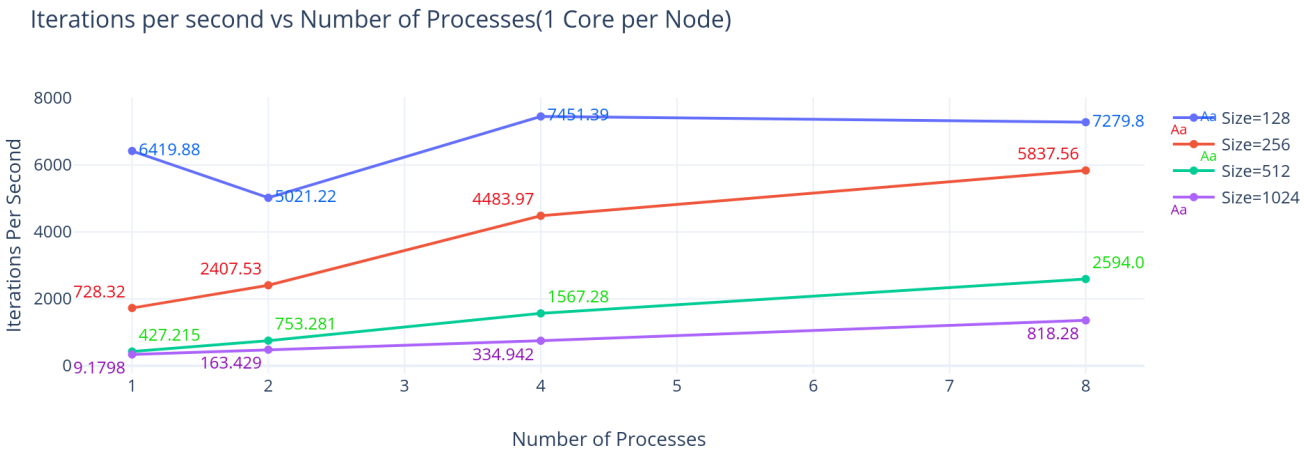

Figure 2: Iterations per second vs Number of processes for multiple Grid Sizes using 1 process per node for weak scaling study. Click here for Interactive figure

In Figure-2 we see quite similar results as we increase nodes rather than processes on a single node. For grid-size 128 and increase till 4 process(nodes) and then a decrease.

Similarly for 256 and 512 grid-sizes, iterations/s are increasing till 8 processes.

NOTE: Use the following scripts -

1. run_n1N1.sh - 1 process on 1 node
2. run_n2N2.sh - 2 processes on 2 node
3. run_n4N4.sh - 4 processes on 4 nodes
4. run_n8N6.sh - 8 processes on 8 nodes

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

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

I have used `mpi4py.MPI.Comm.send` and `mpi4py.MPI.Comm.recv` for communicating with Python objects and `mpi4py.MPI.Comm.Send` and `mpi4py.MPI.Comm.Recv` for direct array data communication. Main Process receives ranks from all other processes adds them up and prints the result.

To run with pickle based communication -

```
python rank_sum.py pickle
```

To run with direct array communication -

```
python rank_sum.py
```

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

To create a periodic cartesian topology I use `MPI.Compute_dims` and `MPI.Comm.Create_cart`

```
dims = MPI.Compute_dims(size, 2)
cart_comm = comm.Create_cart(dims, periods=[True, True])
```

2.3. Exchange rank with neighbours [5 Points]

To the neighbouring ranks I use `MPI.Comm.Cartcomm.Shift` then I use `mpi4py.MPI.Comm.isend` and `mpi4py.MPI.Comm.irecv` to send and receive neighbouring ranks which are `int` objects in python. Example - Sending rank to left neighbour and receiving its rank.

```
coords = cart_comm.Get_coords(rank)
rank_left, rank_right = cart_comm.Shift(0, +1)
rank_bottom, rank_top = cart_comm.Shift(1, +1)

req = cart_comm.isend(rank, dest=rank_left, tag=1)
req.wait()

req = cart_comm.irecv(source=rank_left, tag=1)
received_ranks[0] = req.wait()
```

Example of result for a 4×4 Cartesian Topology with 16 processes-

```
Process with rank=0. Coordinates=[0, 0]. Neighbours: left=12. right=4, bottom=3 and
  top=1
Process with rank=0.Received Ranks: left=12. right=4, bottom=3 and top=1
```

First line is the ranks of neighbours calculated using `MPI.Comm.Cartcomm.Shift` and the second line is the ranks received from neighbours using `isend` and `irecv`.

2.4. Change linear algebra functions [5 Points]

This has been implemented in two ways.

1. Using python `for` loops
2. Using `numpy` library

Writing only about `numpy` implementation. In the code all vectors are stored in a 2D array. So dot product can be computed as - doing an element-wise multiplication and then taking the sum.

```
dot_prod = np.sum(x * y)
```

For 2-norm of a vector `x`

```
norm2 = np.sqrt(np.sum(np.square(x)))
```

2.5. Exchange ghost cells [5 Points]

Since the exchange are `numpy` arrays I used Direct Array Communication. Example of ghost cells' exchange with the west neighbour

```
self.send_requests = [None] * 4
self.send_requests[0] = domain.comm_cart.Iend(self._buffW, dest=domain.
  neighbour_west, tag=0)
self.recv_requests = [None] * 4
self.recv_requests[0] = domain.comm_cart.Irecv(self._bdryW,
```

2.6. Scaling experiments [5 Points]

I wasn't able to run the solution perfectly. Sometimes it would just stuck without any logs. I couldn't make to any tutorial sessions because of other exams and work. I do want to discuss this and will get an appointment with you after the Holidays. Sorry for all the trouble.

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

Since python objects are to be communicated I again use pickle based communication. Process -

1. `manager` gets all the tasks.
2. It divides the complete list of tasks into chunks. The total number of chunks is equal to number of workers(slaves)
3. Number of workers(slaves) is computed as total number of processes in the communicator - Master process does do work itself.
4. `manager` sends each of these chunks to a slave process.
5. `worker` process completes the tasks and sends the updated `mandelbrot_patch` objects back to master where `manager` appends to a list.
6. When all tasks are accumulated in this list, master process uses `mandelbrot.combine_tasks` to generate the complete image data.

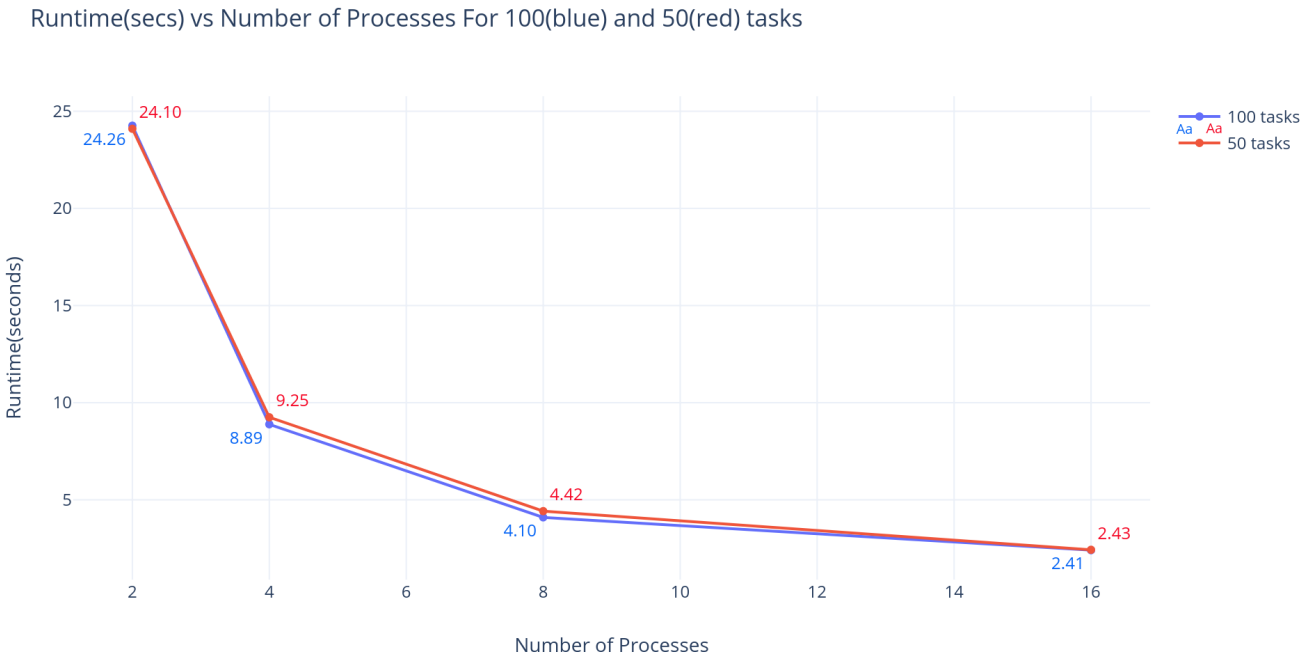

Figure 3: Runtime(seconds) vs Number of processes for 100 taska and 50 tasks. Click here for Interactive figure

As can be seen from Figure-3 the two lines i.e. one for 100 tasks and one for 50 tasks almost overlap. This is because even though the number of tasks are lesser in case of 50 tasks, but each task has more work. The little improvement in case of 100 tasks might be because the nested loops' become smaller when the image is broken down in parts. For example instead of having one 50×50 loop we have 4 25×25 loops.

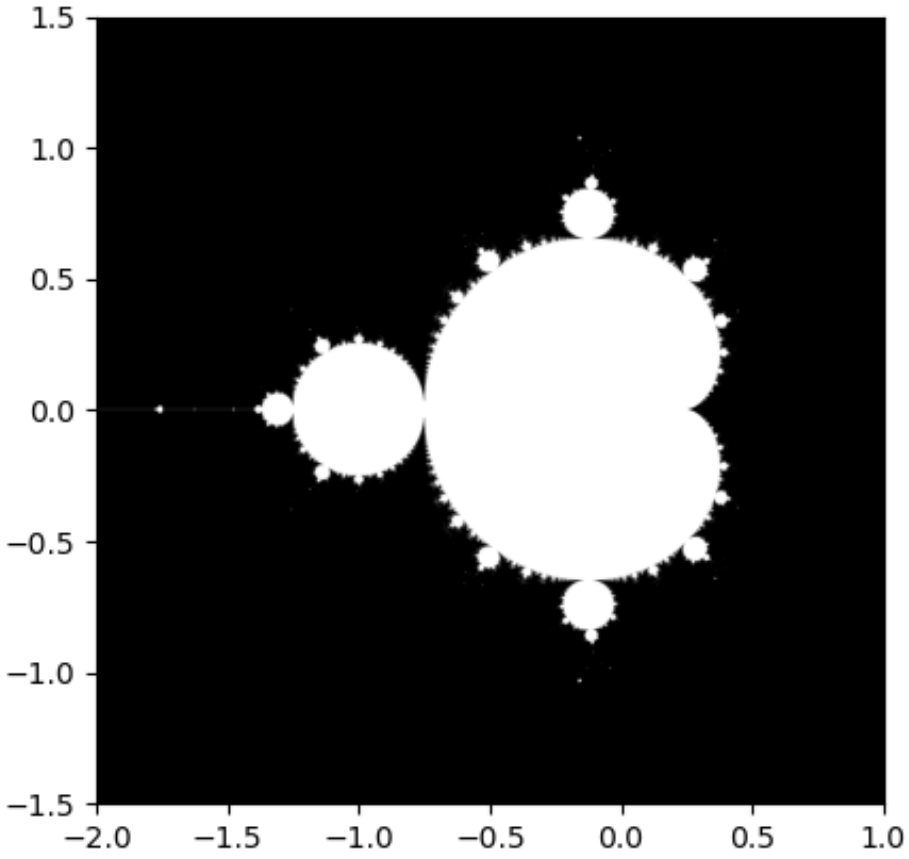

Figure 4: Generated mandelbrot image

3. Task: Quality of the Report [15 Points]

Each project will have 100 points (out of which 15 points will be given to the general quality of the written report).

Additional notes and submission details

Submit the source code files (together with your used `Makefile`) in an archive file (tar, zip, etc.), and summarize your results and the observations for all exercises by writing an extended Latex report. Use the Latex template from the webpage and upload the Latex summary as a PDF to iCorsi.

- Your submission should be a gzipped tar archive, formatted like project_number_lastname_firstname.zip or project_number_lastname_firstname.tgz. It should contain:
  - all the source codes of your solutions;
  - your write-up with your name project_number_lastname_firstname.pdf.
- Submit your .tgz through Icorsi.