INF3201- Parallel Programming

Assignment 1 - MPI

28 August - 17 September 2019

1 Mandelbrot set

Mandelbrot set is a bit mapped image processing, which is computed with significant number of computations. It is the set of complex numbers C (giving the position of the points in complex plane) for which the function (z), defined as $z(k+1) = z_k^2 + C$ does not diverge when iterated by k^{th} iteration of z remains bounded in absolute value. Initially, the value of $z = 0$ and the iterations are continued until it seems that z will eventually become infinite. The Mandelbrot set within a continuously colored environment looks like the white part shown in Figure 1. For a more detailed description check [4].

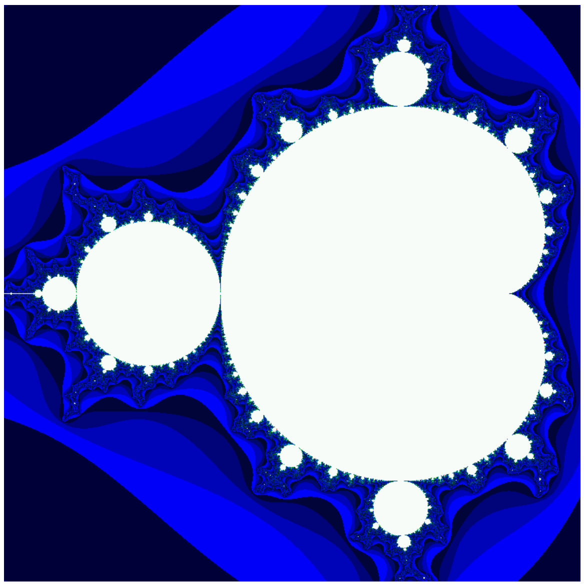
Figure 1: Mandelbrot Set

2 Task

Implement two parallel versions of the given Mandelbrot code, using MPI[5] Sec. 3.1.2 p. 83.

```
do
  temp = z.real* z.real - z.imag * z.imag + c.real;
  z.imag = 2 *  z.realb* z.imag + c.imag;
  z.real = temp;
  lengthsq = z.real * z.real + z.imag * z.imag;
  count++;
while((lengthsq < 4.0) && (count < max_iter));
```

Listing 1: Sequential Mandelbrot code [8]

Implement the following:

1. Use a static scheme without shared memory. Read more in [7].
2. Use a dynamic load balancing scheme without shared memory. It uses the work-pool approach as shown in Figure 2. Read more in [7].

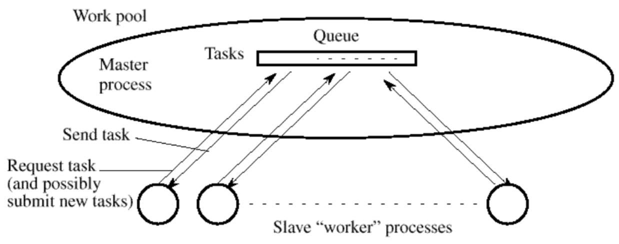

Figure 2: Work-pool approach [8]

Your solution can not use any shortcuts that reduce the functionality of the program. It must produce the same CRC. All the pixel data has to be sent to the master node. It must maintain the rendering functionality. Evaluate your implementation using profiling and tracing tools. Anomalies and lackluster optimizations must be explained in the report

3 Precode

In the repository you can find a directory called `Code`. Inside that directory there are two directories called `RoadMap` and *examples*.

- `Roadmap` - contains precode for the assignment. *Roadmap.c* is the main file with the mandelbrot code. When you compile the given file you should be able to run it. You will find there different python implementations of Mandelbrot set.

- `plot_data.py` - python3 code used to draw mandelbrot set from data saved by `Roadmap.c`. It can be used for a visual validation of your solution. `mandelbrot_roadmap.py` is a python version of `RoadMap.c`. Please remember that you will need to modify `Makefile` to use MPI - an example is provided in the `example` directory. You should use `run.sh` as well in order to run your code.

- `examples` - contains some simple example of MPI code. Try to run this code before writing your own MPI code. `generate_hosts.sh` is useful to create a list of hosts for MPI so you might want to use it for MPI implementation of Mandelbrot as well. Please remember that number of processes and number of host are tightly bounded together which means that if you type: `./run.sh 4 2` it will run 4 processes on two nodes in total. See Listing 2.

```
[uvcluster examples]$ ./run.sh 8 2
Root is running on compute-2-2.local says rank 3, compute-2-2.local
Root is running on compute-2-2.local says rank 1, compute-2-2.local
Root is running on compute-2-2.local says rank 7, compute-2-4.local
Root is running on compute-2-2.local says rank 0, compute-2-2.local
Root is running on compute-2-2.local says rank 5, compute-2-4.local
Root is running on compute-2-2.local says rank 2, compute-2-2.local
Root is running on compute-2-2.local says rank 4, compute-2-4.local
Root is running on compute-2-2.local says rank 6, compute-2-4.local
```

Listing 2: MPI example code output

4 uvcluster

At the Computer Science department we have a small cluster of nodes that can be used to run the parallelized solutions. The cluster has a distributed file system which means that you will need to copy your files only to the frontend server (uvcluster.cs.uit.no) in order to access them from all the other nodes in the cluster.

In order to login to the cluster please use the following command (in linux): `ssh abc123@uvcluster.cs.uit.no` where abc123 is your UiT login.

Please remember to run a command `source scl_source enable devtoolset-8` in order to enable gcc 8 for your current session and `source /share/apps/python385.sh` to enable python3. You can put these commands in your $HOME/.bashrc in order to have them always active.

Please make sure that you are familiar with welcome message from the cluster - see Listing 3. For the given assignment you should use `./generate_hosts.sh` script which will generate a pseudo-random list of hosts for MPI.

```
List nodes available right now: /share/apps/ifi/available-nodes.sh
List the cluster hardware: /share/apps/ifi/list-cluster-static.sh
List nodes by load: /share/apps/ifi/list-nodes-by-load.sh
List all your processes: /share/apps/ifi/list-cluster-my-processes.
    sh
```

```
Clean out all your processes: /share/apps/ifi/cleanup.sh
```

Listing 3: uvcluster welcome message

5 Requirements

- Implement and parallelize both techniques described in Section 2:
  - write code using C/C++ and MPI[6].
  - write code in understandable way and make sure the code is well-commented.
  - Use some additional technique to solve the assignment other than just processing rows or columns by MPI workers. Some approaches that could be used are as follow: create a heat map of most demanding parts to balance the workload for each node, alternate row segmentation [4], apply First Come First Served approach [4] etc. If you are unsure about the additional technique please discuss it with TAs.
  - The delivered code should support odd and even number of nodes as the execution parameter. The minimum number of nodes that the solution should work with is 2.
- Analyze your solution:
  - Do a profiling of sequential code for the two techniques required to solve Mandelbrot set. You might use Gprof[1] or gperftools[2] in order to complete this requirement. You can use Ravel MPI trace visualization tool [3] to show physical timelines along with the logical, while beautifully capturing the intended organization of communication operations. Use the tracing tools when implementing MPI stuff, it will enable you to pinpoint communication and load balance problems quickly.
  - Compare these two techniques - how well does each of these techniques scale with regard to problem size, number of MPI threads. Show the difference in overall execution time, speedup factor and computation/communication ratio. In assumptions, do emphasize on how checksum varies for sequential code with different sizes for best solution. Do not include graphic library to calculate the needful execution time but you need to have the rendering functionality working in the parallelised code as well. So, turn off graphics for the performance measurements when you focus on the computation of the image. Change the variables (zooms, resolution, number of nodes) when doing experiments, to make sure your solution scales as well as possible, while detailing reasons for such behaviour. It is required to do some experiments in order to answer these questions.
- Write report:

- Required sections for the report are as follows (section names might be slightly different):
  - `Introduction` - describe your task.
  - `Sequential solution analysis` - profiling, speedup, expectations, assumptions.
  - `Design` - description of your parallelized techniques - your approach to parallelization (Have you adequately explained your solution?).
  - `Experiments` - describe your experiments with different problem size - different input parameters like resolution, number of zooms, number of hosts etc.
  - `Discussion` - discuss positive and negative sides of your solution. Compare theoretical maximum speedup with your results in details(2-3 pages) with different parameters like sizes from 400*400 to 3200*3200 while using different number of nodes starting from 8 nodes upto maximum nodes in cluster. (Have you critically evaluated your solution? Have you made your assumptions clear and separate from your measurements?). Details of how code runs for even and odd number of nodes.
  - Summary.
- Did you find any unexpected results or behaviour of the system? Try to investigate and find a possible explanation. This might require tweaking the source code or experiment parameters to test ideas and compare results.
- Report should be of 6-7 pages, explaining everything in detail answering the questions like why you choose any particular method, problems in trying different approaches, possible solutions etc.
- Make your figures clear and understandable. Remember about references, captions and axes description.
- Remember to take into consideration the hardware you are using. Hint: MPI cluster is heterogeneous.

6 Hand-in

You will work in groups of 2 for the assignment. Give details of both students working together in the report. If you are unable to find a second student for the group please let us know before September 3, 2020. GitHub classroom is used as a hand-in platform for the course. You can and probably should commit and push as often as you would like. Group members should share a repository.

Please remember to push all your changes before 17 September 2020 23:59:59. Any changes pushed to your repository after that deadline will not be evaluated.

References

[1] Himanshu Arora. Gprof tutorial – how to use linux gnu gcc profiling tool. `https://www.thegeekstuff.com/2012/08/gprof-tutorial/`, 2012.

[2] HoluWu. Main gperftools repository. `https://github.com/gperftools/gperftools`.

[3] Kisaacs. Ravel mpi trace visualization tool. `https://github.com/LLNL/ravel`.

[4] Bhanuka Manesha Samarasekara Vitharana Gamage and Vishnu Monn Baskaran. Efficient generation of mandelbrot set using message passing interface. *arXiv*, pages arXiv–2007, 2020.

[5] Peter Pacheco. *An introduction to parallel programming*. Elsevier, 2011.

[6] ARCHER High Performance Computing Service. Introduction to mpi. `https://www.youtube.com/watch?v=r99hJ04tfG8`.

[7] Mirco Tracolli. Parallel generation of a mandelbrot set. *VIRT&L-COMM*, 2016.

[8] Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.