3325ENG Numerical Engineering Methods
Programming assignment 1

Dr Stephen So

Trimester 2, 2022

Contents

1 Introduction

The aims of this assignment are to parallelise the Mandelbrot set program studied in HPC Studio 2 using several methods to reduce processing times as well as compare and contrast the efficiency of these methods. You will also be using UNIX text processing tools (such as sed, awk, and bash) to automatically run your program to perform difference analyses of execution time and memory usage. You will then need to upload your C source files, a makefile, bash/awk scripts, a 10 minute presentation video, and slides used in the presentation to Learning@Griffith.

You can work in pairs for this assignment but you need to distribute the workload amongst each member. In your slides, you should *indicate clearly the contributions of the code from each team member*. You can enrol your partner and yourself into the assignment group enrolment in the assessment section on Learning@Griffith. If you wish to work alone, you will still need to enrol in a solo group. Please do not join a group that has one person unless the other person has agreed. If anyone wants to change groups, please email me your current group number.

1.1 Submissions instructions

The due date for this assignment is **Friday, 23 September 2022 (Week 9)** at 11.59 pm. You are to submit on Learning@Griffith:

- all of your C source files (`*.c`) and gnuplot plotting files (`*.gp`);
- a `make` file such that all the programs in your assignment can be compiled simply by typing `make` and object/executable files removed by typing `make clean`;
- a bash script that runs your program and then gnuplot to display the fractal;
- scripts using awk, bash, sed, etc. to perform the performance analyses;
- a 10 minute video presentation (in mp4/m4v or mkv format); and
- the PowerPoint slides used for your presentation (in PDF format)

In the submission comments, please put a description of each file and what it does. Both members of your group should be involved in the presentation and powerpoint slides should be used as a presentation aid. More details on how to prepare the video as well as what to mention will be detailed in a later section of this assignment.

1.2 Marks breakdown

The total number of marks in this assessment is 100 and the weighting of the assessment item is 20%. The breakdown of the marks is as follows:

Marking criteria	Marks
Code implementation (70 marks total):	
OpenMP implementation	5
– correct public/private variables, optimal use of chunk size, etc.	
– code elegance, comments, minimal global variables, error checking	
– multiple threads (> 2) specified at command line	
– program design and parallelisation strategies, no memory leaks	
pthreads implementation	20
– correct but minimal use of mutexes	
– innovations (e.g. thread pools and job queue)	
– optimal data chunk size	
– multiple threads (> 2) specified at command line	
– code elegance, comments, minimal use of global variables, error checking	
– program design and parallelisation strategies, no memory leaks	
`fork()` implementation	35
– correct and innovative use of two types of IPC, well-structured data exchange, etc.	
– user-written functions `chread()` and `chwrite()` for sending and receiving appropriately sized data chunks over IPC	
– analysis of efficiency and performance of two types of IPC	
– innovations e.g. multiple processes (> 2), process pools, etc.	
– code elegance, comments, minimal use of global variables, error checking	
– program design and parallelisation strategies, no memory leaks	
Scripts for automatic performance analysis	10
– efficient and correct use of UNIX tools (e.g. grep, sed, awk, bash, gnuplot)	
Presentation video (30 marks total):	
Complexity and elegance of solution	7
Program presentation and analysis	8
Technical rigour	8
Presentation skills	7

For each component shown above, you will be marked based on the categories of 'exceptional', 'very good', 'average', and 'not attempted'. The 'exceptional' category is normally reserved for work that is exceptional and has gone well beyond the usual expectations.

2 Software resources required

You will require the following software resources:

- Linux/UNIX environment (WSL on Windows, Linux OS, or macOS[1]);
- gnuplot (for Debian WSL, `sudo apt install gnuplot-x11`; for Mac with Homebrew installed, `brew install gnuplot`)

For students using the WSL, you will need to install the `time` package separately, since it does not come by default. You can do this by issuing the following command at the terminal:

```
sudo apt install time
```

After successful installation, you can access the `time` command by typing:

```
/usr/bin/time
```

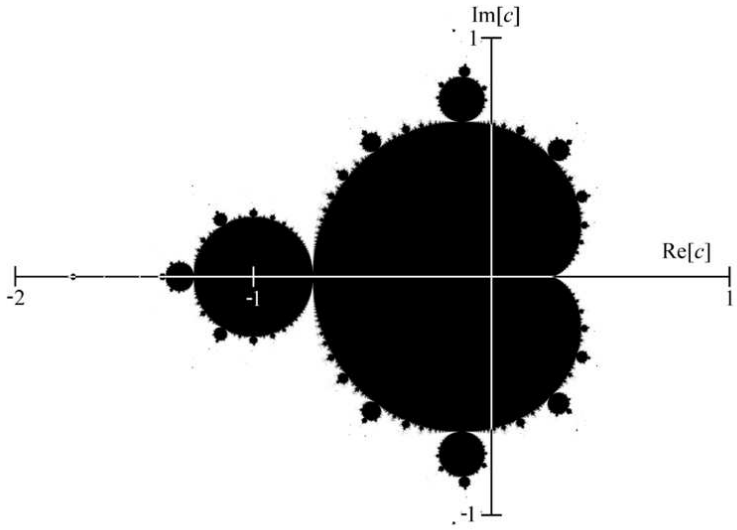
[1]Note that POSIX message queues are not available on macOS. If you want to use message queues for IPC, you can only use System V message queues.

3 Introduction

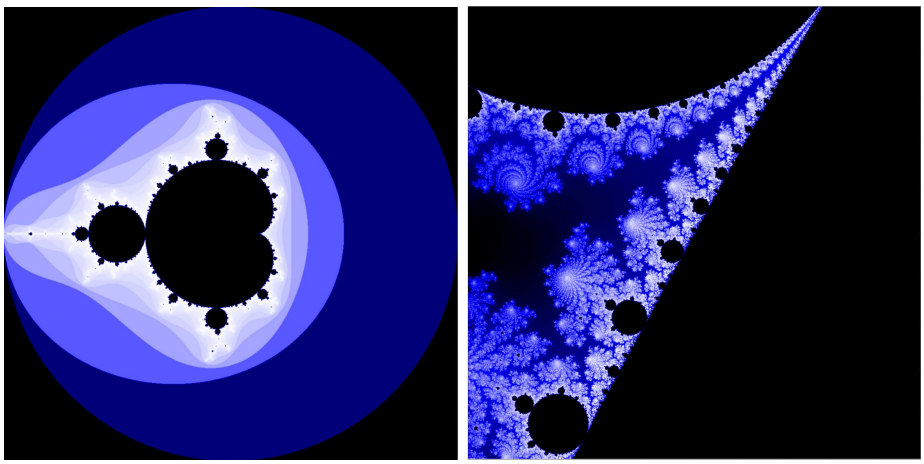
The Mandelbrot set is a mathematical set of complex numbers c, where z remains bounded (i.e. does not go to infinity) in the following recursive equation:

$$z_{n+1} = z_n^2 + c \tag{1}$$

where z_0 is initialised to $0 + 0j$. For complex numbers that are **not part of the Mandelbrot set**, z will continually increase **indefinitely**. A Mandelbrot set can be plotted in the complex plane, where points that belong to the set are coloured black while those that extend to infinity are given a colour (or white, as shown in the Figure below).

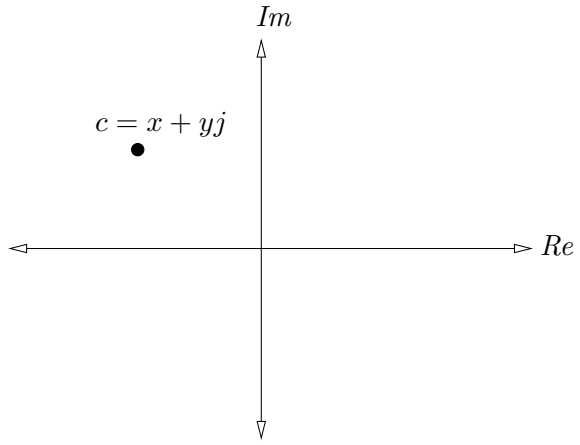


The resulting graphic is called a fractal and it has infinite detail. You can zoom into any section to reveal more and more elaborate structures (as seen below)[2].



The value or 'colour' of each point where z goes off into infinity (i.e. they are not part of the set) is determined using an 'escape-time' algorithm.

[2]The region seen in the second graphic is centred around the value $-0.668 + 0.32j$, where the width of the region is 0.02.



Given a certain point $c = x + yj$ in the complex plane (see figure above), we say this point is not part of the set if the corresponding z value $(a + bj)$ has a magnitude that is greater than 2 (i.e. it has **escaped**):

$$|z| = \sqrt{a^2 + b^2} > 2 \tag{2}$$

Therefore, for each point c in the complex plane, we perform the test to see if it is part of the Mandelbrot set:

$$z_{n+1} = z_n^2 + c \tag{3}$$

We perform these iterations a number of times (as determined by the variable `maxIter`):

- if the point has not escaped after `maxIter` iterations, it is part of the Mandelbrot set
- if the point has escaped before reaching `maxIter` iterations, it is not part of the Mandelbrot set

The colour of a point that has escaped is related to the number of iterations it took to escape. Based on the above description, if the `iterations = maxIter`, then the point is black (since it did not escape and is therefore part of the Mandelbrot set).

4 Assignment tasks

4.1 Parallelisation of Mandelbrot set

Your goal is to modify the serial Mandelbrot program so that it utilises more processing cores so that efficiency and processing speedup is achieved. Please use the template that is provided on L@G for this assignment. It differs slightly from the one you saw in the studios. Specifically, the `width` and `height` variables are now passed to each function via the `Parameters` structure, rather than using the preprocessor macros `WIDTH` and `HEIGHT`. This is more convenient for the parallelisation task.

You will be investigating **three methods of parallelisation** as separate and independent programs[3]. These methods have been (or will be) covered in the lectures and studios and are shown below (in increasing levels of difficulty):

[3]That is, you should only use one of these parallelisation methods for each program, rather than combining them.

1. Multithreading using OpenMP (`omp`);
2. Multithreading using pthreads (`pthr`); and
3. Multiple processes using `fork()` (`fork`).

You will need to write a make file to ensure that everything can be compiled by simply typing `make`. You will need to make a bash script that runs your code and gnuplot as well.

4.1.1 Details on the `fork()` implementation

For the `fork()` implementation, you should implement at least two interprocess communication strategies and compare them in terms of efficiency and performance. Recall that pipes and socket pairs have a data limit per read/write of 65536 bytes[4]. Therefore, if you need to send a large amount of data down a pipe or socket pair, you will need to split the data array into *uniform chunks*[5] (e.g. 1024 elements of 4 byte integers), and then read/write each chunk by calling `read()` and `write()` *multiple times*. To do this in a portable fashion, you are to write two functions that mimic the arguments of the `read()` and `write()` system calls. Their prototypes are given below:

- `int chread(int fd, void *buf, int count, int chunk_size);`
  - `fd` is the field descriptor from the pipe or socket pair
  - `buf` is a pointer to the array to store the bytes being read
  - `count` is the number of bytes to read from the file descriptor
  - `chunk_size` is the chosen size of each chunk
  - This function should return the number of bytes read, 0 if end of file is reached, or -1 if an error occurred.
- `int chwrite(int fd, void *buf, int count, int chunk_size);`
  - `fd` is the field descriptor from the pipe or socket pair
  - `buf` is a pointer to the array to write to the file descriptor
  - `count` is the number of bytes to write to the file descriptor
  - `chunk_size` is the chosen size of each chunk
  - This function should return the number of bytes written or -1 if an error occurred.

4.1.2 Multiple processes/threads and choosing a suitable region

Start your parallelisation design with two threads/processes first. Once you are confident that this is working, then, modify them so that they have a user-selectable number of threads/processes that are specified as a command-line argument (via the `argc` and `argv`) mechanism.

[4] Note that you will notice that you have hit the limit when you discover that no data or not all the data was received over the pipe or socket pair.

[5] Note that your code should handle the case where there are not a whole number of chunks. For example, if we have an array of 10 elements, and we choose chunks of 4 elements, then we will have two chunks plus an extra 2 elements, which will then need to be read/written separately.

You are free to name your C source code files but they should end with an underscore (_) and the suffixes listed above. For example, if the base name of our C code file is `mandelbrot`, then the file names for each method would be:

- `mandelbrot_omp.c`
- `mandelbrot_pthr.c`
- `mandelbrot_fork.c`

You should choose a set of co-ordinates and maximum iteration counts that result in a *slow serial program*. Generally, the more points that are in the Mandelbrot set (dark points) in the range, the slower your program would be. A suggestion is the region centred around the value of $-0.668 + 0.32j$, with a width of 0.02. Feel free to explore the Mandelbrot set to look for better looking and more intricate regions. You should aim for an **serial execution time of about 40–60 seconds**, in order to better appreciate the speedups of the parallelisation strategy.

4.2 Program design and parallelisation strategy

You are free to make design modifications to the serial Mandelbrot template program in order to make it work with the different parallelisation methods. You can also add extra command line arguments (via `argc` and `argv`) to change certain parameters for the analysis part. However, the main structure and functions of the program should be quite similar to the template and be recognisable. This is to prevent the use of 'other code implementations' being used, so *please do not copy code from the Internet*. You will need to justify these decisions in your presentation.

It is also up to you to design your parallelisation strategies, e.g. how to divide up the data, how to exchange data between threads/processes, what model to use (manager/worker, peer-based, thread pool[6], etc.), etc. You will be assessed on the complexity, correctness and elegance of your solutions. For instance, use of global variables should be minimal unless they are critical to your design.

4.3 Analysis

The idea is to analyse the efficiency of your program design and parallelisation strategy as well as identify the advantages and flaws.

4.3.1 Timing and memory usage

Timing the execution of your programs is an important part of the analysis, especially comparing it with the serial program as well as varying numbers of threads/processes. You are to use the UNIX tool (`/usr/bin/time`)[7] to measure the execution time of your programs.

[6] You must write your own thread pool code when not using OpenMP, i.e. using the job queue code from the lecture slides.

[7] Note that the bash shell also provides a `time` command, but this is not the same as the `/usr/bin/time` tool, since on some platforms, such as macOS, it shows the total time of all processes/threads instead of the actual program executation time. On WSL, you will need to install the `time` package separately and run via `/usr/bin/time`.

The `/usr/bin/time` tool prints its information to the *standard error* stream. In order to be able to pipe the output of `/usr/bin/time` tool, we need to redirect standard error to standard output (`2>&1`) and direct the existing standard output to `/dev/null`. Here is an example run on the mandelbrot program `mb5` with 20000 iterations, where we are piping the information to `awk`, which is printing it out entirely.

```
/usr/bin/time ./mb5 20000 2>&1 >/dev/null | awk 'print $0'
```

This is the output that we get:

```
6.97user 0.09system 0:07.32elapsed 96%CPU (0avgtext+0avgdata 29180maxresident)k
0inputs+89848outputs (0major+6939minor)pagefaults 0swaps
```

The time elapsed shown is 7.32 seconds while the program used 29180 kB of resident memory.

4.3.2 Suggested analyses to perform

The following analyses are suggested to be run:

- An analysis using Amdahl's law and the ratio of serial/parallel code for each method;
- time comparison of pipes vs socketpair IPC and effect of different chunk sizes;
- analysis of total execution time for varying numbers of processes/threads;
- analysis of total resident memory for varying number of processes; and
- comparison of parallelisation methods (processes vs threading vs OpenMP) using timing and total resident memory and discuss possible differences in speed, memory, communications delay, operating system overhead, etc.

You should write a bash script called `performance_analysis.sh` that automatically runs the various analyses, collates the results for each parallelisation strategy, prints a numerical report in the terminal, and generates plots using the `gnuplot` program. It should save the `gnuplot` plots into PNG formatted image files for inclusion into your Powerpoint slides.

You are to use `/usr/bin/time` to do the timing and memory measurements. Also feel free to use any combination of `bash`, `sed` and `awk` to facilitate the analyses (e.g. using bash `for` loops to cycle through parameters to pass to your program, `sed` and `awk` to extract the relevant numbers, etc.).

4.4 Video demonstration and presentation

You will also need to prepare a 10 minute video presentation of your assignment that is presented by both members of your team (or yourself if working individually), which presents:

- your final parallelised Mandelbrot set programs using the different methods of parallelisation;
- an analysis of your parallel program design and technical justification on design decisions (e.g. why you believe xyz was the best way to achieve speed-up);

- the analyses suggested in Section 4.3.2. It should include graphs of processing time, etc.;
- a reflection on the difficulties encountered while designing and implementing the programs; and
- a real-time demonstration of each program and performance script running at the end of the presentation (not included in the 10 minute limit).

It is recommended that you use a webcam from a PC (most University PCs will have a webcam) or laptop to capture each member of the group presenting their slides using the OBS Studio software, which you can freely download from the Internet[8]. OBS Studio can also be configured to perform desktop screen capture as well. A video on how to use OBS Studio will be included on Learning@Griffith.

You can use video editors, such as Shotcut[9], the free version of DaVinci Resolve by Blackmagic Design[10], or any video editing program that you are familiar with, to edit your video and stitch together the footage. Try to keep your video to about 10 minutes in length, minus the final real-time demonstration of each program running. Export it using the H.264 video codec into an mp4/m4v file and upload it to the Learning@Griffith assignment submission point, along with your PowerPoint slides, bash/awk scripts, and C source code. Note that depending on the size of your video file, it may take some time for the submission to complete. Please email me if you are having difficulties uploading your video file to Learning@Griffith.

You will be assessed on the following criteria:

- Presentation skills (clarity of explanation, eye contact, intonation, etc.);
- Technical rigour (accuracy and validity of the explanation, correct use of technical terms, etc.);
- Program presentation and analysis; and
- Complexity and elegance of the solution.

[8] You can download OBS studio from https://obsproject.com/download

[9] Shotcut is free software that you can download from https://shotcut.com/downloads/

[10] From my use of DaVinci Resolve in making videos, despite it being highly feature-rich and powerful, it has quite a bit of a learning curve. It can be downloaded for free from https://www.blackmagicdesign.com/au/products/davinciresolve/