Sheet 04

PS Parallel Programming

Patrick Wintner

April 1, 2025

1 Mandelbrot

The execution time of the program mandelbrot is measured.

1.1 Source Code

```c
#include <pthread.h>
#include <errno.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Include that allows to print result as an image
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

// Default size of image
#define X 1280
#define Y 720
#define MAX_ITER 10000

struct thread_info{
        size_t start;
        size_t end;
        uint8_t (*image)[X];
```

```
};


void* calc_mandelbrot_partial(void* arg) {
        struct thread_info* thread_info=arg;
        for(size_t i=thread_info->start; i<thread_info->end; ++i) {
                for(size_t j=0; j<X; ++j) {
                        double x=0;
                        double y=0;
                        double cx=(double)j/(X-1)*3.5-2.5; // scale j to [-2.5, 1]
                        double cy=(double)i/(Y-1)*2-1; // scale i to [-1. 1]
                        size_t iteration=0;
                        while(x*x+y*y<2*2 && iteration<MAX_ITER) {
                                double x_tmp=x*x-y*y+cx;
                                y=2*x*y+cy;
                                x=x_tmp;
                                iteration=iteration+1;
                        }
                        char norm_iteration=iteration*255/MAX_ITER; // scale iteration t
                        thread_info->image[i][j]=norm_iteration;
                }
        }
        return NULL;
}

void calc_mandelbrot(uint8_t image[Y][X], size_t n_threads) {
        size_t chunk_size=Y/n_threads;
        size_t remaining=Y%n_threads;
        pthread_t threads[n_threads];
        struct thread_info args[n_threads];
        for(size_t i=0; i<n_threads; ++i) {
                args[i].start=i*chunk_size;
                args[i].end=(i+1)*chunk_size+(i+1==n_threads ? remaining:0);
                args[i].image=image;
                pthread_create(threads+i, NULL, calc_mandelbrot_partial, args+i);
        }
        for(size_t i=0; i<n_threads; ++i) {
                pthread_join(threads[i], NULL);
        }
}

int main(int argc, char** argv) {
        if(argc!=2) {
                fprintf(stderr, "Usage: %s <n_threads>\n", *argv);
```

```
                return EXIT_FAILURE;
        }

        errno=0;
        char* end;
        long n_threads = strtol(*(argv+1), &end, 10);

        if(errno || *end) {
                perror("strtol");
                return EXIT_FAILURE;
        }

        uint8_t image[Y][X];

        calc_mandelbrot(image, n_threads);

        const int channel_nr = 1, stride_bytes = 0;
        stbi_write_png("mandelbrot.png", X, Y, channel_nr, image, stride_bytes);
        return EXIT_SUCCESS;
}
```

1.2 Measurement Method

The measurement was done on the LCC3 cluster by calling sbatch job.sh. The following scripts are involved in the measurement process.

1.2.1 SLURM Job Script

```
#!/bin/bash

# Execute job in the partition "lva" unless you have special requirements.
#SBATCH --partition=lva
# Name your job to be able to identify it later
#SBATCH --job-name test
# Redirect output stream to this file
#SBATCH --output=output.log
# Maximum number of tasks (=processes) to start in total
#SBATCH --ntasks=1
# Maximum number of tasks (=processes) to start per node
#SBATCH --ntasks-per-node=1
```

```
# Enforce exclusive node allocation, do not share with other jobs
#SBATCH --exclusive
# Enable hyperthreading
#SBATCH --hint=multithread

./main.sh
```

1.2.2 Main Script

```
#!/bin/bash
# Usage: ./main.sh
MEASUREMENT_RESULTS=measurements
PROCESSED_RESULTS=results
make
for i in {1,2,4,8,12}
do
        rm $i"_"$MEASUREMENT_RESULTS $i"_"$PROCESSED_RESULTS
        for j in {1..3}
        do
                /bin/time -f "%e" -a -o $i\_$MEASUREMENT_RESULTS ./mandelbrot $i
        done
        ./toTex $i\_$MEASUREMENT_RESULTS $i\_$PROCESSED_RESULTS 3
done
make clean
```

The measurement results are stored in mandelbrot_measurments.log, which is read by process_results to compute the average execution time and standard deviation, which are stored in mandelbrot_processed.log.

1.3 Measurement Results

#threads	time/s	mean/s	standard deviation/s