Sheet 03

PS Parallel Programming

Patrick Wintner

March 24, 2025

1 Mandelbrot

The execution time of the program mandelbrot is measured.

1.1 Source Code

```
#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

void calc_mandelbrot(uint8_t image[Y][X]) {
        for(size_t i=0; i<Y; ++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
                        image[i][j]=norm_iteration;
                }
        }
}

int main() {
        uint8_t image[Y][X];

        calc_mandelbrot(image);

        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 measurements were taken using the script mandelbrot/main.sh (which is called in the slurm script):

```
#!/bin/bash
# Usage: ./main.sh
MEASUREMENT_RESULTS=mandelbrot_measurements.log
PROCESSED_RESULTS=mandelbrot_processed.log
make
rm $MEASUREMENT_RESULTS $PROCESSED_RESULTS
for ((i=0;i<=2;++i)); do
        /bin/time -f "%e" -a -o $MEASUREMENT_RESULTS ./mandelbrot
done
./process_results $MEASUREMENT_RESULTS $PROCESSED_RESULTS
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

time/s	mean/s	standard deviation/s
17.77	17.74	0.0265
17.72		
17.73		

1.4 Suggestions for performance improvement and parallelisation

The calculations of the colour of different pixels are independent, therefore those calculations could be done parallel.

2 False Sharing

2.1 Differences and Implications

Both versions allocate storage dynamically to a pointer called sum. Each thread is responsible for incrementing exactly one value of the allocated memory (therefore they should not influence each other), until it equals the value given as command line parameter. The final result is the sum of all elements.

In the first version, there is no padding between the memory locations used by the threads for the computation. This means that it is likely that several memory locations used by different threads will be stored in the same cache line. If a thread writes to a memory location in a cache line, it invalidates also all other data stored in the same cache line. Therefore if another thread wants to increment another value stored in the same cache line, the previous thread has to write the cache line back into memory, causing significant delay (even though otherwise the threads were independent).

In the second version, there is padding between the memory locations used for incrementing, hopefully preventing that memory locations of different threads are getting loaded into the same cache line.

2.2 Comparision same processor - different processors

number of cpus	1	2
false_sharing 1	0.311	0.390
false_sharing 2	0.206	0.206

The second version is not affected by increasing the number of processors, while the first suffers an increase in execution time for the reasons stated above.

2.3 Perf Analysis

2.3.1 Overview

```
running with 6 Threads on cores 0,1,2,6,7,8
Total sum: 600000000
Time taken: 0.401349 seconds

 Performance counter stats for './false_sharing 100000000':

          2,220.34 msec task-clock:u
                 0      context-switches:u
                 0      cpu-migrations:u
                80      page-faults:u
     6,451,931,844      cycles:u
     5,263,553,697      stalled-cycles-frontend:u
     1,479,760,818      stalled-cycles-backend:u
     2,411,603,664      instructions:u



Total sum: 600000000
Time taken: 0.207501 seconds

 Performance counter stats for './false_sharing_2 100000000':

          1,245.48 msec task-clock:u
                 0      context-switches:u
                 0      cpu-migrations:u
                82      page-faults:u
     3,618,081,567      cycles:u
     2,427,423,905      stalled-cycles-frontend:u
```

```
       686,292,218      stalled-cycles-backend:u
```

The number of cycles of the first version is way higher.

2.3.2 Details

```
Performance counter stats for './false_sharing_2 100000000':

               409      LLC-load-misses:u
                94      LLC-store-misses:u

       0.210323834 seconds time elapsed

       1.236407000 seconds user
       0.000000000 seconds sys


running with 6 Threads on cores 0,1,2,6,7,8
Total sum: 600000000
Time taken: 0.398705 seconds

 Performance counter stats for './false_sharing 100000000':

         1,160,871      LLC-load-misses:u
         1,460,061      LLC-store-misses:u

       0.402514828 seconds time elapsed

       2.199481000 seconds user
       0.002966000 seconds sys


Total sum: 600000000
Time taken: 0.205975 seconds

 Performance counter stats for './false_sharing_2 100000000':

               270      LLC-load-misses:u
               107      LLC-store-misses:u

       0.209600113 seconds time elapsed
       1.225613000 seconds user
       0.008948000 seconds sys
```

The number of LLC-load-misses of the first version is by several magnitudes higher.

(Sorry for ugly report, time is running out)