TDDD56 Multicore and GPU computing :

Lab 1 : Load Balancing on Mandelbrot set

Daniel Johansson

Arnaud Pecoraro

November 2015

Preparatory questions

Question 1

Generating a Mandelbrot set is not an equilibrated task. Indeed some pixels need more computation than others. Pixels are generated via the method *is_in_Mandelbrot* which consists in a loop bounded by a given number of iteration *maxiter* and a distance *dist2*. If the distance is superior than 4, the computation stops and the pixel is not part of the Mandelbrot set. However if this pixel is actually part of the set the computation stops when *maxiter* is reached, which implies more computation. This results in an unbalanced workload.

Question 2

The naive load-balancing method consists in dividing the picture same size blocks, each assigned to a thread. For example, running the program with two threads results in dividing the the workload in two equal parts. Obviously this method is unbalanced. A better solution would be to divide the work in way smaller parts, a square or a rectangle of (x,y) pixels resulting in a lot more *workunit* than the number of threads. This way, each thread would compute a tiny piece of work, and after completion look for another one among the remaining workload. A system of mutex has to be implemented to protect the work assignation. Statistically this method provides a more balanced computation because each of them will process *hard* and *easy* parts.

```
int x_work = 1; // Chunk size specification
int y_work = 1;

while(x_num <= parameters->width && y_num < parameters->height)
{

    pthread_mutex_lock(&mutex);

    if(x_num < parameters->width)
    {
        parameters->begin_w = x_num;
        parameters->end_w = x_num+x_work;
        x_num += x_work;
    }
    else
    {
        x_num = 0;
        y_num += y_work;
    }

    parameters->begin_h = y_num;
    parameters->end_h = y_num+y_work;

    pthread_mutex_unlock(&mutex);

    compute_chunk(parameters);
}
```

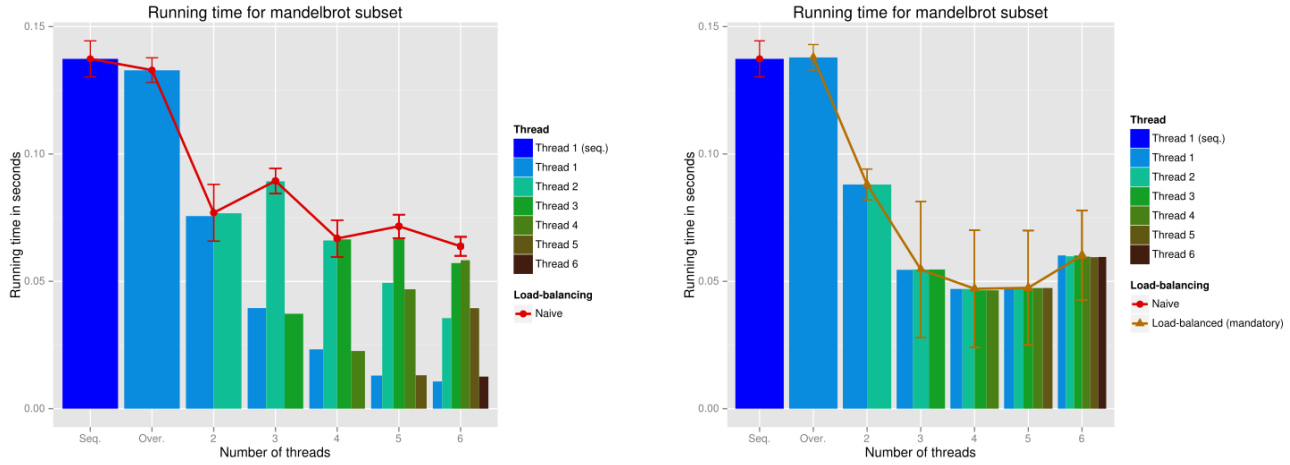
FIGURE 1 – Load-balanced implementation

Performance results and comparison

On our laptop(i5 2520M, 2Cores 4Threads)

On the following figures, we cleary notice that the workload is unbalanced with the *naive* solution, for example the computation is actually slower using 3 threads rather than 2. On the other hand, with our balanced implementation the workload scale with the number of threads.

Note that with this 2C/4T machine we observe slower performances while using 5 or 6 threads, which is normal since the computation is using more threads that the machine actually has.



(a) (b)

FIGURE 2 – Naive (a) and Load-balanced (b)

On a multicore lab computer

(a) (b)

FIGURE 3 – Naive (a) and Load-balanced (b)