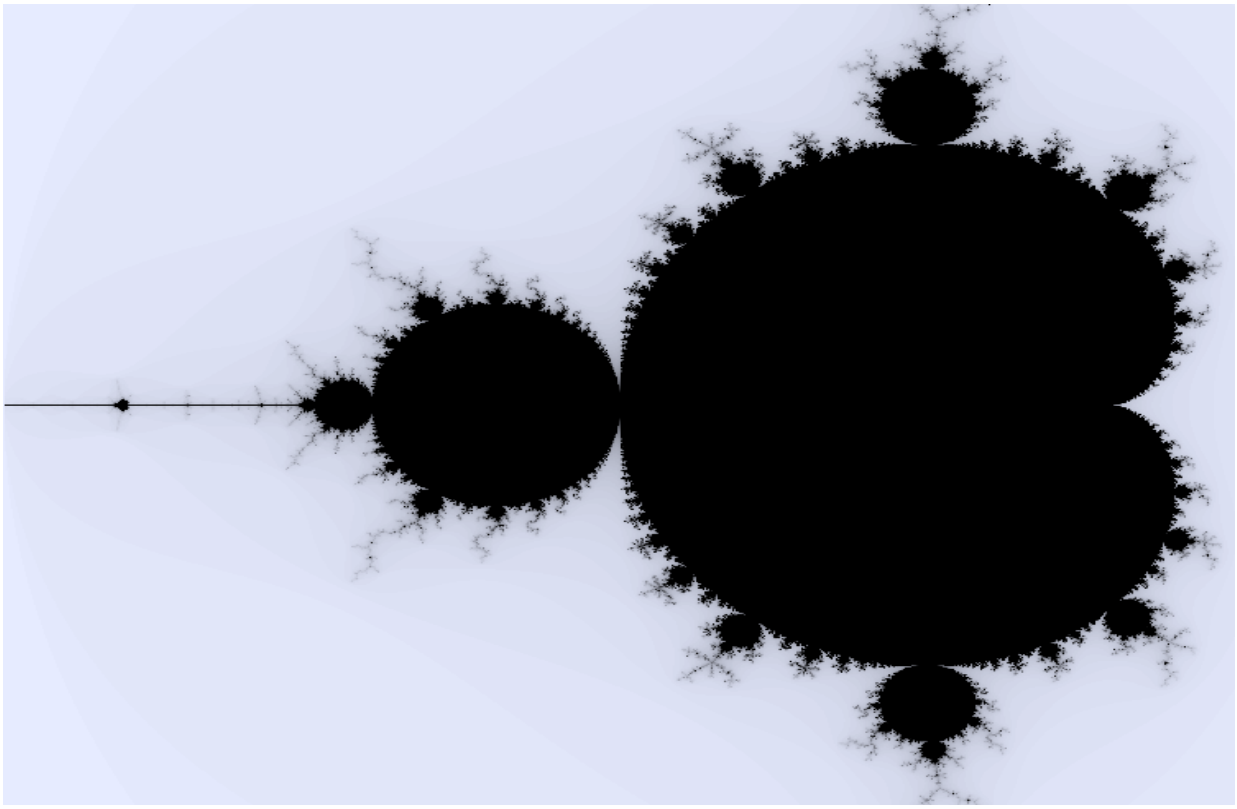
Jean-Baptiste Dumat
Univerisity of Kent

Mandelbrot in Go



Report :

You can find all documentation relative to both implementation in the files “geometric-distrib-doc.pdf” and “worker-pool-doc.pdf”. You can also read README.txt to find more details about the benchmarking.

In this report, we will focus on the comparison of both implementation in term of performance.

It was really hard to do proper benchmarks; firstly, for an unexpected reason, the results on Raptor are drastically lower compared to the one computed on my own laptop.

```
jfd7@raptor [~/Go-Asses] $ python3 script.py mandelbrot_worker-pool-expe.go 10
Experiment using mandelbrot_worker-pool-expe.go
        Number of repetition: 10
        Average of sample is 19.259580541774632
        Standard Deviation of sample is 0.42794802129356674
        Standard error of the mean is 0.1353290467449908
```

Figure 1 shows the experiments results on Raptor on a 3072x1228 px image.

These results are 5 times better on my own laptop.

Secondly, for no conceivable reason it seems that the more logical heart you use on your CPUs, the slower it seems to be.

```
[~/Documents/Go-Asses » python3 script.py mandelbrot_worker-pool.go 10
 Experiment using mandelbrot_worker-pool.go
        Number of repetition: 10
        Average of sample is 5.7967351551
        Standard Deviation of sample is 0.17059770233345928
        Standard error of the mean is 0.053947730296515325
```

Figure 2 shows the experiments results on my laptop using runtime.NumCPU threads with worker-pool pattern on a 3072x1228px image.

```
[~/Documents/Go-Asses » python3 script.py mandelbrot_worker-pool.go 10
 Experiment using mandelbrot_worker-pool.go
        Number of repetition: 10
        Average of sample is 3.8198396457999992
        Standard Deviation of sample is 0.12692818276377263
        Standard error of the mean is 0.04013821567996473
```

Figure 3 shows the experiments results on my laptop using only 1 thread with worker-pool on a 3072x1228 px image.

I first thought that I was launching too much goroutines at the same time and my CPU couldn't handle it, but it seems inconceivable because with this pattern I only use a dozen of goroutines and my CPU (a 2,6 GHz Intel Core i7 6 cores) should handles this.

```
Experiment using ../test2.go
        Number of repetition: 10
        Average of sample is 2.5242650951000005
        Standard Deviation of sample is 0.09682989231748558
        Standard error of the mean is 0.03062030053120944
```

Figure 4 shows the experiments results using no concurrency on a 3072x1228 px image.

Thirdly, results using concurrency are never better than the one we can obtain by using no concurrency. At low cost computations we obtain similar results. The difference growths almost exponentially if we up the resolution of the picture.

Geometric distribution pattern:

This approach in other hand, showed very good results. It was around 1.4 sec faster than the default implementation for a 3072x1228px image generation and more than 2.7 sec faster than the worker-pool design.

```
~/Documents/Go-Asses » python3 script.py mandelbrot_geo-distrib.go 10
Experiment using mandelbrot_geo-distrib.go
        Number of repetition: 10
        Average of sample is 1.1256412519999999
        Standard Deviation of sample is 0.021551973451882804
        Standard error of the mean is 0.006815332417943098
```

Figure 5 shows the experiments results using geometric distribution pattern on a 3072x1228 px image.

As explained in the documentation, geometric distribution consists of splitting the image in sub-rectangle which will be computed concurrently.
It is interesting to see how much the number of sub-rectangles in the image can interfere on the result. Too much can slow down drastically the program because the CPUs cannot handle it. On the other hand, too less leads the same result for another reason which is that you do not take advantage of all your hardware capabilities.

In conclusion, these results are very surprising because I had the assumption that the worker-pool pattern would be the faster of the three. In fact, it seems to be the slowest of all three experiments in this case.
Maybe the image processing is not complex enough to show good results compared to the geometric distribution. However, I wanted to push further and applied it in a similar way to other experiment such as an implementation in Go of the merge sort algorithm and got very good results. So, I conclude that it might not fit well to the Mandelbrot case.