Prof. I. F. Sbalzarini
ETH Zentrum, CAB G34
CH-8092 Zürich

Prof. P. Arbenz
ETH Zentrum, CAB G69.3
CH-8092 Zürich

Exercise 3

Release: 13 Oct. 2009
Due: 20 Oct. 2009

Question 1: Beyond loop parallelism

Last week, we investigated different scheduling strategies for load balancing in `Mandelbrot fractals exercise`. In this exercise, we will `dither` the image generated by using `mandelbrot(...)` function, to soften the resulting image. In order to do so, we will apply a `vertical filter` to each pixel in the image.

$$\begin{bmatrix} 0.25 \\ 0.5 \\ 0.25 \end{bmatrix}$$

Now, the program will contain two sets of nested for loops.

```
DO j=1:COL_MAX
  DO i=1: ROW_MAX
     field(i,j) = mandelbrot(...)
  END DO
END DO

DO j=1:COL_MAX
  DO i=1: ROW_MAX
     dither_field(i,j) = ...
  END DO
END DO
```

Following the `loop-level` or `fine-grained parallelism` approach, the two loop nests can be parallelized separately, but this will unnecessarily force the parallel threads to synchronize between the `mandelbrot` loop and the `dither` loop. Instead we will follow `coarser-grained parallelism` wherein each thread will move on to dithering it's part of image without waiting for other threads.

a) The file `mandel_dither.c` contains the sequential program. Parallelize the program by splitting up the image matrix in vertical strips and assigning the strips to threads depending on the thread ID. Execute the program using different number of threads and comment on your results.

b) The image matrix is now stored in COLUMN_MAJOR layout. Do you think the change from ROW_MAJOR to COLUMN_MAJOR layout is justified?

Question 2: Monte Carlo method

`Monte Carlo`[1] methods are a class of computational algorithms that rely on repeated random sampling to compute their results. Because of their reliance on repeated computation of random or pseudo-random numbers, Monte Carlo methods are most suited to calculation by a computer. Monte Carlo methods tend to be used when it is unfeasible or impossible to compute an exact result with a deterministic algorithm.

In the present exercise, we will calculate the combined area of three intersecting circles enclosed in a square of unit length. It is possible to calculate the area analytically, but the solution is cumbersome. Instead, we calculate the area as follows:

> Generate N randomly distributed points enclosed within the bounding square. Count the number of points M that lie within the circumference of any one of the circles.

$$Combined\ area\ of\ circles = \frac{M}{N} \times (Area\ of\ bounding\ square)$$

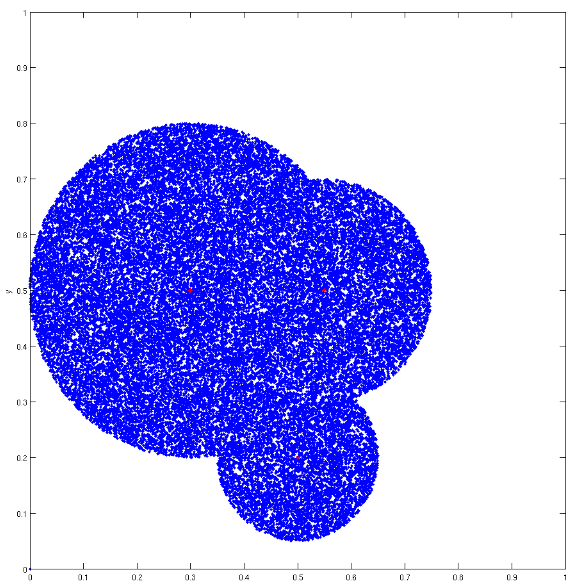

Figure 1: Circles centered at (0.3,0.5) (0.55,0.5) and (0.5,0.2) having radii 0.3,0.2 and 0.15 respectively. The bounding square has unit length

The sequential version of the above algorithm is implemented in file `area.c`. Parallelize the code using OpenMP and compare the results and execution time with the serial version.

[1] `http://en.wikipedia.org/wiki/Monte_Carlo_method`