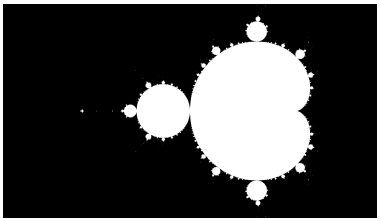
February 11, 2014

OpenMP Tutorial

Jon Eyolfson

Provided Program

We create a black and white image similar to:



Each pixel colour is determined by the Mandelbrot algorithm

Code

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define WIDTH 1400
#define HEIGHT 800

int main(int argc, char *argv[])
{
  ...
}
```

Code - main (1)

```
int i_max = 0;
{
  int c;
  while ((c = getopt (argc, argv, "i:")) != -1) {
    switch (c) {
    case 'i':
      i_max = atoi(optarg);
      break;
    default:
      return EXIT_FAILURE;
    }
  }
}
```

Code - main (2)

```
if (i_max <= 0) {
  printf("%s: option missing or requires an argument > 0"
         " -- 'i'\n", argv[0]);
  return EXIT_FAILURE;
}

FILE* pgmFile = fopen("output.pgm", "wb");
if (pgmFile == NULL) {
  fprintf(stderr, "Failed to open pgm file\n");
  return EXIT_FAILURE;
}
fprintf(pgmFile, "P5\n%d %d\n255\n", WIDTH, HEIGHT);

int i;
double x0, y0, x, xtemp, y;
```

Code - main (3)

```
for (int h = 0; h < HEIGHT; ++h) {
  for (int w = 0; w < WIDTH; ++w) {
    i = 0; x = 0.0, y = 0.0;
    x0 = 3.5*((double) w)/((double) WIDTH - 1.0) - 2.5;
    y0 = 2.0*((double) h)/((double) HEIGHT - 1.0) - 1.0;
    while ((x*x + y*y < 4.0) && (i < i_max)) {
      double xtemp = x*x - y*y + x0;
      y = 2*x*y + y0;
      x = xtemp;
      ++i;
    }
    if (i == i_max) fputc(255, pgmFile);
    else fputc(0, pgmFile);
  }
}
```

Code - main (4)

```
fclose(pgmFile);

return EXIT_SUCCESS;
```

Brief summary:

1. Get a command line argument for maximum number of iterations
2. Error checking, open file and write image header
3. The actual algorithm we're interested in
4. Close the file and exit the program

Automatic Parallelization Probem

Let's target the outermost loop (most profitable) in **`main (3)`**

The compiler says "`not parallelized, call may be unsafe`"

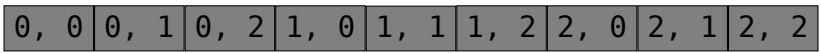
Specfically, the call to `fputc` is unsafe

Unsafe Illustration (1)

The pixel data in the file requires it to be in sequential order

Consider a simple 3x3 image (9 bytes of data)

The expected order is (each box represents a byte):



Which is what we'd get for this program as `fputc` writes a byte and advances to the next position

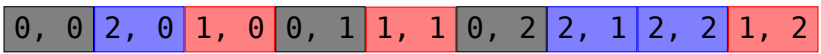
Unsafe Illustration (2)

Assume we thread the outer look with 3 threads, each handling a row

We know `fputc` writes the bytes in the order calls happen

Therefore, we have a datarace

One possible outcome is (each colour represents a different thread):



This is not what we want or expect

Enabling Automatic Parallelization (1)

Writing the file is preventing us from automatic parallelization

The calcuation of the pixel colour could be done in parallel

We could create a buffer for the pixel data and write sequentially

Enabling Automatic Parallelization (2)

```
char colour[HEIGHT][WIDTH];
for (int h = 0; h < HEIGHT; ++h) {
  for (int w = 0; w < WIDTH; ++w) {
    // ... as before...
    if (i == i_max) color[h][w] = 255;
    else color[h][w] = 0;
  }
}
for (int h = 0; h < HEIGHT; ++h) {
  for (int w = 0; w < WIDTH; ++w) {
    fputc(colour[h][w], pgmFile);
  }
}
```

All data within our main loop is independent

Automatic parallelization now takes place

Automatic Parallelization Results

The program runs much faster with 4 threads

It's still slower than the maximum theoretical speed, can we do better?

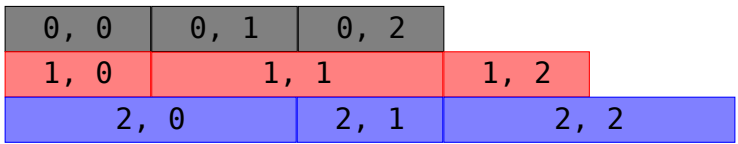
Slowdown Illustration (1)

Observing the Mandelbrot algorithm, we see it can exit early

This means white pixels are calculated faster than black ones

Assume that white pixels can be calcated in half the time as black

One possible outcome is (width is proportional to time):



Slowdown Illustration (2)

The work isn't evenly distributed

`Thread 0` takes 3 time units

`Thread 1` takes 4 time units

`Thread 2` takes 5 time units

We have to wait 5 time units for our program to complete

We could've completed in 4 time units

OpenMP Solution

Recall that OpenMP addresses this problem

The `for pragma` has a `schedule` clause

The `dynamic` schedule more evenly distributes the load

We can add something like:

```
#pragma omp parallel for schedule(dynamic, 10)
```

Now, we achieve close to our ideal speedup