并行计算实验报告 – 2

# 关于Mandelbrot图形

Mandelbrot图像中的每个位置都对应于公式N=x+y\*i中的一个复数。其实数部分是x，虚数部分是y，i是-1的平方根。图像中各个位置的x和y坐标对应于虚数的x和y部分。

图像中的每个位置用参数N来表示，它是x\*x+y\*y的平方根。如果这个值大于或等于2，则这个数字对应的位置值是0。如果参数N的值小于2，就把N的值改为N\*N-N（N=（x\*x-y\*y-x）+（2\*x\*y-y）\*i）），并再次测试这个新N值。如果这个值大于或等于2，则这个数字对应的位置值是1。这个过程一直继续下去，直到我们给图像中的位置赋一个值，或迭代执行的次数多于指定的次数为止。

# 修改思路

原算法的核心步骤在119行（mandelbrot.c）开始的循环中，所以我们需要并行化的部分就是这个ij两层循环。但考虑到每个循环只处理一个元素粒度过小，可能会带来线程过多以及切换上下文代价占比过大的情况，所以只将最外层循环（i循环）并行化即可。

只需在循环语句之前加入一行openMP指导语句即可实现并行化（mandelbrot\_mpi.c 130行）：

#pragma omp parallel for default(shared) private(x, y, c, c0, v, j, d, k)

然后加上计时语句即改造完成。

# 运行结果

默认参数下运行得出的图像如下图：

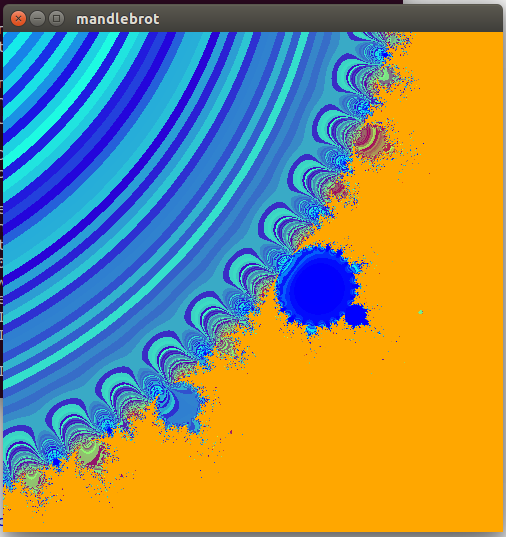


Figure 1 运行图像

不同线程数运行的时间如下图：

因为实验用计算机具有16个硬件线程，所以在小于等于16个线程规模的时候随着线程数的提升性能计算效率也随之提升；当线程数大于16时由于切换上下文的代价上升，所以运行效率受到影响。

修改线程数只需要修改omp\_set\_num\_threads()函数中的参数即可，这也是openMP功能方便之处。

Figure 2 改造后程序运行时间统计