Introduction to multiprocessor architecture:

Report Assignment 1

# Part 1:

1. Algorithm analysis
   1. In function “caculate\_pi”, we parallelized the process of the generation and statistics of random points. The random generated points are trivially independent from each other. The variable num\_in\_cycle is to be shared between the execution threads since it has to sum all the points that are in the correct area (unit circle)
   2. Multiplication (maybe random generation instead)
   3. Obviously, we need to compute ‘samples’ numbers of points, then ‘samples’ affects the number of performance-critical operations. It is O(n) since add one point to the sample set will add one execution in the for loop.
   4. In the following graph we have the time of execution on the Y-axis and the number of threads on the X-axis. To estimate the running time starting from the single threaded execution we simply assume that the number of threads is linearly related to the speedup

# Part 3:

1. Algorithm analysis
   1. In function “integrate”, we parallelized the process of the generation of points and the compute of the area. It is obviously that the area is affected by the generated point and the points are independent to each other.
   2. Multiplication
   3. Obviously, we need to pick ‘samples’ numbers of points, then ‘samples’ affects the number of performance-critical operations. It is O(n)
   4. As for the first estimation we have only one for loop that is parallelized so we can easily assume that the speedup would be linear in the number of threads.
2. Results on SCITAS

[TODO]

1. Comparison between prediction and SCITAS execution

[TODO]

1. From the results, it is found that there is not much difference in the running time between these two methods. In fact, most of the time a single line of #pragma from Openmp will result on a parallel run equivalent to a well-structured Pthread-based code.

In terms of running speed, Pthread seems to be a little bit better. But in terms of code work, the OpenMP code is more concise. Pthread API provides a more fine-grained control over thread management so for example in a program that has a lot of different tasks to run in parallel it would be more convenient to assign every task to a dedicated thread using Pthread. On the other hand, Openmp is very handy to parallelize reductions and for loops for example.