DTS202TC Foundation of Parallel Computing

Lab 3: Profiling, Shared-Memory Programming with Pthreads

This lab weight 35 marks of the A3 lab report, please save all screenshots of activities, source code for future reference.

Profiling (13 marks)

Download the source code from https://github.com/AndrewCarterUK/mnist-neural-network-plain-c. Use any profiling tools you are familiar with to find out the bottle net of the program. You may consider Instruments on Mac or gprof on Cygwin Windows. Please screenshot the results and provide brief explanations (maximum 100 words).

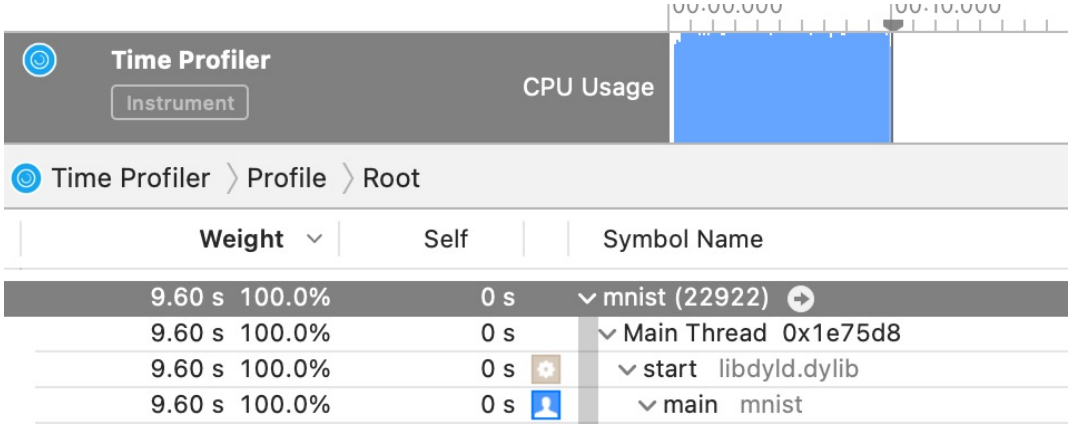

Figure 1: Instruments on Mac

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.13% of 7.70 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     99.7    0.00    7.68                 main [1]
```

Figure 2: gprof on Cygwin Windows

Pthreads Hello World (2 marks)

Type the following source code manually (no copy and paste) in a pth_hello.c file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

/* Global variable:  accessible to all threads */
int thread_count;

void* Hello(void* rank);  /* Thread function */

int main(int argc, char* argv[]) {
   long       thread;  /* Use long in case of a 64-bit system */
   pthread_t* thread_handles;

   /* Get number of threads from command line */
   thread_count = strtol(argv[1], NULL, 10);

   thread_handles = malloc (thread_count*sizeof(pthread_t));

   for (thread = 0; thread < thread_count; thread++)
      pthread_create(&thread_handles[thread], NULL,
          Hello, (void*) thread);

   printf("Hello from the main thread\n");

   for (thread = 0; thread < thread_count; thread++)
      pthread_join(thread_handles[thread], NULL);

   free(thread_handles);
   return 0;
}  /* main */

void* Hello(void* rank) {
   long my_rank = (long) rank
         /* Use long in case of 64-bit system */

   printf("Hello from thread %ld of %d\n", my_rank,
         thread_count);

   return NULL;
}  /* Hello */
```

The program is compiled like an ordinary C program, with linking in the Pthreads library:

```
gcc -g -Wall -o pth_hello pth_hello.c -lpthread
```

Run the program:

```
./pth_hello <number of threads>
```

Estimate π with Pthreads (20 marks)

Suppose we toss darts randomly at a square dartboard, whose bullseye is at the origin, and whose sides are 2 feet in length. Suppose also that there's a circle inscribed in the square dartboard. The radius of the circle is 1 foot, and it's area is π square feet. If the points that are hit by the darts are uniformly distributed (and we always hit the square), then the number of darts that hit inside the circle should approximately satisfy the equation

$$\frac{\text{number in circle}}{\text{total number of tosses}} = \frac{\pi}{4} \tag{1}$$

We can use this formula to estimate the value of π with a random number generator:

```
number_in_circle = 0;
for (toss = 0; toss < number_of_tosses; toss++) {
  x = random double between -1 and 1;
  y = random double between -1 and 1;
  distance_squared = x*x + y*y;
  if (distance squared <= 1) number_in_circle++;
}
pi_estimate = 4*number_in_circle/((double) number_of_tosses);
```

This is called a "Monte Carlo" method, since it uses randomness (the dart tosses). Write a Pthreads program that uses a Monte Carlo method to estimate π. The main thread should read in the total number of tosses and print the estimate. You may want to use long long ints for the number of hits in the circle and the number of tosses, since both may have to be very large to get a reasonable estimate of π.