Parallel Processing Lab 2: Pthreads Numerical Integration

Sarah Peachey & Nathan Schomer

February 21, 2018

Abstract: pthreads is used to parallelize an executable function. When the pthread is created it is given a pointer to a function and a structure containing all the information and needed to run that function. The data structure than needs to be dereferenced, and the code can then run over a section of the data.

1 Design

The parallel numerical integration algorithm was designed such that using N threads and a trapezoid width of W, each thread would calculate W/N sub-trapezoids assuming $W \mod N = 0$. With this design, it would be expected to see a decrease in execution time with an increased number of threads.

When implemented, the algorithm was broken into 3 for loops which follows the basic design pattern of a pthreads program. The first packed a custom struct with all necessary data for each thread (start x, end x, number of trapezoids, and height). The next loop created each thread, passing in the respective argument struct and a pointer to the function that calculates the current trapezoid. The last for loop, waits for all threads to join and then adds the result from each thread to a global sum. The global sum is then returned to the calling function. Pseudocode is included on the next page.

```
Data: start_x, end_x, num_traps, height
Result: estimated integral of function between start_x and end_x
initialization;
for i < num_threads do
    pack argument struct for thread i;
end
for i < num_threads do
    create thread i;
    serially calculate sub-trapezoid i;
end
for i < num_threads do
    join each thread;
    global_sum = global_sum + thread_sum;
end
```

Algorithm 1: Pthreads Numerical Integration

2 Results

The pthreads numerical integration function was executed using 2, 4, 8, and 16 threads. Each thread count was executed 10 times and the average speed-up was recorded. The results are listed in Table 1. Speed-up was calculated by dividing time of serial execution by the average execution time for each thread count as shown in Equation 1.

As seen in Table 1, continuous speed-up was seen up to 8 threads. Then, there was a slight drop off with 16 threads. Overall, the parallelized algorithm achieved substantial gains in execution time over the serial version.

$$s = \frac{t_{serial}}{t_{parallel}} \tag{1}$$

Thread Count	Speed-Up
2	4.70
4	13.91
8	15.50
16	14.77

Table 1: Average Speed-Up over 10 Iterations