Assignment 2 Open MP Histogram Parallelization

Jason Gallagher

Steve Kravitsky

In this assignment we parallelized a section of code which generated a histogram of random data. The serial code we were given was comprised of two main logic loops. First the histogram array was initialized to zero, then the histogram was filled using the input data (see figure 1).

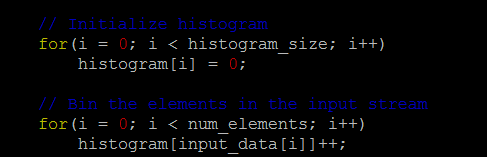


Figure 1: Serial Code

The parallel code that we wrote used the same initialization loop however, the execution of the second loop was broken up into threads. First, we create a private variable for each thread called private\_hist, which is initialized to zero with the calloc command (Figure 2). Each thread will have one of these private variables which they will be used to store the partial summations of the input data.

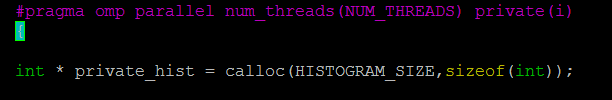


Figure 2: Private histogram initialization

The partial summation occurs in a set of nested loops where the input\_data is being accessed by all of the threads, however, the private\_hist is private to each thread (Figure 3).

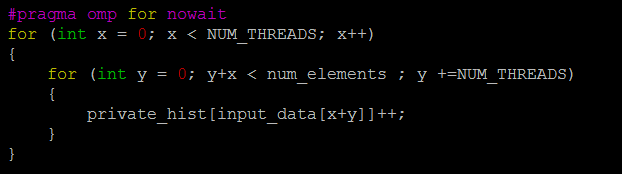


Figure 3: Partial summation logic

Finally, after all threads have completed they hit the critical section where all of the private\_hist values are summed up and stored in the histogram array (Figure 4).

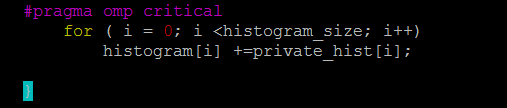


Figure 4: Join data

As a result of parallelizing the histogram code we achieved speedups up to 5x which occurred when we ran the code with 16 threads. Even with as little as two threads we saw speedups around 2x.

Figure 5: Speedup chart

To compile the code run the make command: make

To execute ./histogram <value>

Data

Threads 2

Serial Parallel

0.01 0.00

0.04 0.03

0.45 0.25

Threads 4

Serial Parallel

0.01 0.00

0.05 0.02

0.36 0.18

Threads 8

Serial Parallel

0.01 0.00

0.04 0.02

0.37 0.12

Threads 16

Serial Parallel

0.01 0.00

0.05 0.01

0.36 0.10