Parallel Processing Lab 1: OpenMP

Sarah Peachey & Nathan Schomer

February 9, 2018

Abstract: OpenMP is a general parallelization pragma. To implement the parallel functionality, just identify a parallel section and OpenMP will spawn a set of threads to run the code in parallel. If a parallel for loop is created then it will chunk the for loop into several pieces, and give each thread a chunk to run. The for loop has an implicit barrier synchronization at the bottom of the loop, so threads will wait here before moving onto the next line of code. OpenMP also has a built in to critical sections and will serialize that section, so that multiple threads won't try to write to some critical variable in shared memory.

1 Histogram

The histogram code generates an n length vector and then sorts all the data into a histogram with 500 bins. In the multi-threaded approach each thread is assigned a chunk of data to sort. This chunk of data is statically assigned so they sort equal sized chunks. Each thread keeps a local copy of the histogram containing the data that it sorted. Then after each thread sorts its own chunk of data it reaches a critical section in which the histogram in shared memory is incremented with the values in the local histograms. As seen in the below table peak performance is reached with 8 threads, performance is not improved with 16 threads because of the overhead associated with spawning more threads. Then an explanation in pseudo-code of how the parallel code is structured follows the table.

Threads	1 million items	10 million items	100 million items
2	1.40	1.29	1.10
4	2.33	2.35	2.44
8	3.00	4.17	4.42
16	2.50	4.15	3.56

```
Result: Parallel Organization of data into Histogram
initialization;
creation of threads;
for i in amount of data in thread chunk do
    localHist[array[i]]++;
end
critical section;
for all the bins in the histogram do
    sharedHist[i]+=localHist[i];
end
```