Aleena Jimmy George
801254680

Topic 7 - Assignment OpenMP Loops

1. Reduce

Question: Run the code on the cluster, in the reduce/ directory, using make bench. Plot speedup charts with make plot. Does the plots make sense? Why?

At n=10000 the speedup is negligible, as no of threads isnt enough to get more parallel speedup. but as n become 1000000 speedup increases but until we have too many threads to merge. At n=100000000 the speedup increases. This is because larger thread count helps with splitting elements and adding in parallel, inturn increasing the speedup.

2. Numerical Integration

Question: Run the code on the cluster, in the numint/ directory, using make bench. Plot charts with make plot. Does the plot make sense? Why?

At n = 1000 and granularity is 1 → The speedup readings are irregular. As intensity increases sppedup increases upuntil thread = 8 where the speedup decreases.

At n = 1000 and granularity is 10 → The structure of the chart is much similar to that of n=1000. But the dip in speedup occurs at 12 threads.

At n = 1000 and granularity is 100 → as granularity increases the speedup increases as the capacity increases. When thread count increases(16) the speedup decreases. At higher intensities though the speedup increases because it helps in increasing the efficiency of a high intensity program, with a low n value.

At n=100000 gran=1 → the speedup got low, when the number of threads reached 16 that the syncing became a real issue causing this drop in speedup.

At n=10000000 gran=100 → This is a very high number with a high granularity, and here we notice a similar trait of having a noticable linear speedup with only a drop in the lower intensities, as at these intensities the need for such a high number of threads becomes moot as the syncing would cause an unwanted complexity.

3. Prefix Sum

Question: Provide the structure of the parallel algorithm for Prefix Sum. (Highly recommend the “cut in P and fix it” approach”.)

In the cut and fix method, each thread calculates the sum separately. This method helps when there are no of threads but the count doesnt grow high enough to make sync process longer.

Parallel alog:

1. Divide array into fixed size blocks

2. Calculate prefix sum sequentially
3. Calculate prefix sum of right most element in each block
4. Excluding right block, add right most element of each block to subsequent block
5. The result will be the final array

Question: Run the code on the cluster, in the prefixsum/ directory, using make bench. And then plot the results using make plot. Does the plot make sense? Why?

Speedup is less for lower values of n; since it causes an over head rather than decreasing time efficiency. As n increases (n = 100000000, n=1000000000) speedup increases. At these high values of n, the higher the number of threads the better. But at lower values dips are noticed at a specific number of threads.

4. Merge Sort

Question: Provide the structure of the parallel algorithm for Merge Sort that does not try to make Merge parallel. How would you map that parallelism to OpenMP's looping construct?
Recursive sort calls can be called while running the parallel merge sort. Overhead exists because merging part of the code is not parallel.

```
parallelMergeSort(a,p,r) {
        If p<r
                q = (p+r)/2;

        //execute in parallel (omp parallel sections)
        parallelMergeSort(a,p,q);
        parallelMergeSort(a,q+1,r);

        MergeArr(a,p,q,r);
}
```

Question: Run the code on the cluster, in the mergesort/ directory, using make bench. And then plot the results using make plot. Does the plot make sense? Why?

For low no of elements, there isnt much speedup until no of thread increases to 2, the overhead increases and the performance deteriorates. As thread count increases the speedup increases linearly.
For all other higher values of n we see the speedup increases and as parallelism increases as no of threads increases. Splitting the tasks increases parallel efficiency increasing speedup.

Question: (Extra Credit) Still using only OpenMP loops, make Merge Sort more parallel by making Merge parallel.