**Assignment 4**

OpenMP – loops

Name: Manish Ladkat

Student ID: 801167905

1. **Reduce**

Q. Does the plots make sense? Why?

Answer:

Yes, the plot makes sense.

* reduce\_plots\_threads.pdf: There are 24 different plots for different scheduling, granularity, threads and N. We can observe from the graphs that as the number of threads and the granularity is increased, we can observe an increase in the speedup.
* reduce\_plots.pdf: Three different graphs are plotted for different values of n, as the value of n increases the speedup is increased, we can also see that the speedup for dynamic, 100000 scheduler is the maximum.

1. **Numerical Integration**

Answer:

Yes the plots make sense.

* Numint\_plots.pdf: Plots are for speedup against number of threads. They are plotted with a different intensity. Speedup is stable and constant for higher intensities when n is highest, which means an optimum balance can be achieved for the two graphs of num\_int plots.
* Numint\_plots\_threads.pdf: We can see linear speedup wherever optimum balance of all the factors is achieved. Also, the computations where the intensity is less and the granularity is high a good speedup is achieved.

1. **Prefix Sum**

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

Answer:

Parallel Structure of the parallel algorithm for Prefix Sum:

* Array P is cut into pieces. Number of pieces is equivalent to number of threads.
* In every chunk, the first element will get appended by zero.
* In every chunk, we keep track of offset correction values in a different array in which last element is added.
* Perform prefix operation again on this array containing offset values.
* Add the array of correction values, to the succeeding chunks in parallel. Validate the resulting ‘pr’ array with expected output.

Q. Plot the results using make plot. Does the plot make sense? Why?

Answer:

In prefix graph, we can infer that we have not received optimum parallelism, as there is significant increase in the speedup. We have cut in the ‘pr’ array to receive partial computations. A negligible part of code is sequential after this computation to acquire prefix of offset values needed for further correction. Then we again add up this offset or correction value to their respective chunks of data in parallel. Applying various types of scheduling, I can observe that this result is not dependent of type of scheduling used.

1. **Merge Sort**

Q. 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?

Answer:

Parallel Structure of Merge Sort:

1. Based on number of threads divide the iterations
2. Within the parallel implementation, you should keep track of implementing the variable globally.
3. Until we get down to single elements, divide the array equally.
4. Compute left most, middle and right most variables on each iteration.
5. Call merge function iteratively.
6. Merge method will compare first element of each two arrays passed to it and arrange numbers in ascending order.
7. Return resulting merge sorted array.

Q. Does the plot make sense? Why?

Yes, the plot makes sense.

Mergesort\_speedup\_n.pdf : Speedup is achieved. Maximum speedup achieved is 2.5 - 3. It can be observed that with increasing value of n, speedup increases with increasing number of threads. Maximum speedup achieved was for maximum value of n and number of threads. n=1000000000 and threads=16

Mergesort\_speedup\_thread.pdf : There are total 6 plots in this pdf. And we can infere form the graphs that, with increasing value of N and with increasing number of threads, we observe that speedup keeps increasing. Hence, parallelism is achieved.