Homework 2: Algorithm Analysis and OpenMP (Due 9/28, 11:59 pm)

For problems 3 and 4, please use the BMI cluster. For this assignment, it is not necessary to use SIMD instructions.

To allow mulitple threads, use the flag –cpus-per-task=p when submitting via srun or sbatch.

All write ups should be submitted via github classroom assignment 2. (https://classroom.github.com/a/mgfaLFBm). You can use word, markdown, or pdf for the write ups, and it is fine to include photos of drawings. Please submit a single write up document for all questions to the github repo.

For the sequential matrix multiplication and PI estimation code, you can use your code from assignment 0. If your assignment 0 is python base, you can use ChatGPT to translate to c/c++.

For this assignment, you can use compilation flag -O3 which will allow auto-vectorization and loop unrolling.

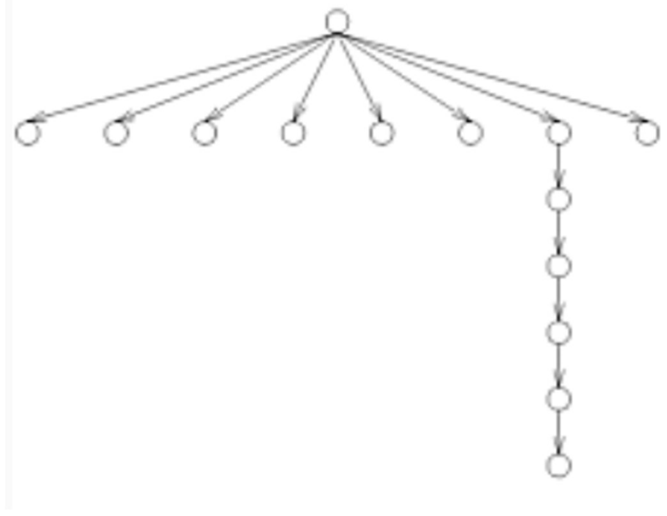
1 Work/Depth Model (10 pts)

1.1 Draw the DAG of algorithm to compute vector dot product for vectors of length 4 via sequential algorithm. What are the work and depth for this case? Generalize the derivation to vector of size n. What is the average available parallelism?

1.2 Draw the DAG of algorithm to compute vector dot product for vectors of length 4 by summing in parallel. What are the work and depth for this case, then generalize the derivation to vector of size n. What is the average available parallelism?

1.3 Extending 1.2, draw the DAG for a parallel square matrix-matrix multiplication with three rows and three columns. (Assume algorithm uses a simple double loop over the elements of the output array, and the vector dot-product routine from 1.2.) Derive the depth and work for the 4 × 4 case. Generalize your derivation to the case of multiplying two matrices of size n × n.

1.4 Given the following task DAG, what are the work, depth, and average available parallelism? What is the maximum achievable speed up? What is the minimum number of processes needed to achieve maximum speed up? What is the maximum speed up with a) 2, and b) 8 processors?



2 Parallel Algorithm Analysis (10 pts)

2.1 As presented in Week 5 Lecture 1, we can compute the cost or T_{all} by multiply the number of processors with the parallel time. Re-analyze the algorithm in problem 1.3 with matrix size NxN and processor count of p for cost, speed up, efficiency, and overhead (as expressions of N and p). Is the algorithm cost optimal?

2.2 Let N = 1024, plot the speed up of the algorithm in 2.1 as p increases from 1 to 1024 in powers of 2. Next fix p to 8, plot the speed up of the algorithm as N increases from 8 to 1024. Repeat the second plot for p = 32 and N increases from 32 to 4096.

2.3 Instead of the algorithm in 1.3, assume the algorithm uses sequential vector dot product and parallelized by the outer loop (over rows of the first matrix). Analyze the algorithm with matrix size NxN and processor count of p for cost, speed up, efficiency, and overhead (as expressions of N and p). Is the algorithm cost optimal?

2.4 Using the algorithm in 2.3, create the speed up plots in 2.2.

3 Compute π (10 points for code, 5 points for write-up)

3.1 Using your previous sequential Monte Carlo code to estimate PI. Parallelize it using OpenMP. Recall that random numbers generation needs to be parallelized. Compare the error with the value of $\pi = \cos^{-1}(-1.0)$.

3.2 What is the speedup? Plot the strong scaling (as a function of number of threads) and the weak scaling (as a function of the number of points) for p from 1 to 8.

4 Matrix Multiplication (10 points for code, 5 points for writeup)

4.1 Multiply two NxN random matrices. Check your results by comparing the outputs of a serial and a parallel version.

4.2 Plot the strong scaling (as a function of number of threads) and the weak scaling (as a function of the number of points) for p from 1 to 8. Use any optimization strategy that you can find, to run your code faster.

4.3 You are free to use the full OpenMP API, adjust your matrix multiplication strategy, and use any optimization strategy you can find. **In other words, write the fastest matrix multiplier**.

5 For Fun and Glory

You can earn bonus points for problems 2 and 3. You will need to provide me with the following:

5.1 **For Problem #3**, plot the time to compute vs. computed error. Competition: computes the most accurate PI in the least amount of time using OpenMP operations. **You can use other approximation strategies to calculate PI.**

5.2 **For Problem #4**, plot N (size of matrix) vs. Time. On your plot, record the N that you were able to compute in 10s. Competition: Compute the mat-mat product for the largest matrix N in 10s, but proposing and implementing a different memory access pattern (e.g. process a chunk of a row/column at a time).