Lab 3

Parallelization with OpenMP Directives

Part 1

1. Set Up Your Environment. Ensure you have access to the DenseMatMultOmp.logs and DenseMatMult2Omp.log files generated in the previous lab.

If you're using Python, make sure the following libraries are installed: pandas, matplotlib, numpy, re (Regular Expressions for text parsing)

You can use Google Colab (similar to jupyter notebook) for your ease as you do not need to worry about installing libraries.

2. Parse the Logs Read in the log files DenseMatMultOmp.logs and DenseMatMult2Omp.logs: The logs contain information about the number of threads used, the dimensions of the matrices, and the time taken for each matrix multiplication in microseconds. Extract relevant information such as: Number of threads. Time taken for each test case (convert microseconds to seconds for consistency). You can use regular expressions (regex) to extract this information or any other text parsing method of your choice.

3. Calculate the Average Time For each unique thread count (1, 2, 4, 8, 16), calculate the average time taken across all test cases. Perform this step separately for both logs (DenseMatMultOmp1 and DenseMatMult2). You can also choose to plot speedup/execution time w.r.t to one test case for each thread count scenario across both algorithms.

4. Calculate Speedup, it is defined as the ratio of time taken by the baseline (the slower version, DenseMatMultOmp1) to the time taken by the optimized version (DenseMatMult2Omp).

Formula: Speedup = Time taken by DenseMatMultOmp1/Time taken by DenseMatMult2Omp.

5. You can choose to plot Average Time vs. Number of Threads Create a line plot showing the average time (in seconds) vs. number of threads. Use different colors/markers for DenseMatMultOmp1 and DenseMatMult2Omp to distinguish between them.

6. You can also choose to plot Speedup vs. Number of Threads Create a line plot showing the speedup vs. number of threads. Set DenseMatMultOmp1 as the baseline, so its speedup should be 1 for all thread counts. Plot the speedup of DenseMatMult2Omp against this baseline.

Note: You can plot any combination of graphs from log files to analyze the performance visually as the main goal for this exercise is to learn analysis of various algorithms and effects of parallel coding practices. If you are not comfortable with the log file format, you can update Lab 2's code to generate logs in your favorable format using print statements of your choice.

Deliverables: Plotname.png (Can be Multiple)

Part 2

In this part of the lab, you will be working on parallelizing various serial codes using OpenMP. Each problem consists of a serial code implementation.

Your goal is to parallelize the serial logic using the appropriate OpenMP directives after the serial implementation section. Each problem has its own function that consists of a serial part (given) and a parallel part (need to be

implemented). You may reuse the serial code in most problems but in few cases you may need to change the order of operations or code your own logic or introduce synchronization mechanisms to resolve dependencies or data race conditions.

Few OpenMP Directives that might be useful:

- #pragma omp parallel
- #pragma omp for
- #pragma omp critical
- #pragma omp atomic
- #pragma omp reduction
- #pragma omp ordered
- #pragma omp barrier

Note: When you implement the parallel code for problems, it's not always guaranteed that your implementation will achieve better performance or speedup because in some cases, handling dependencies and ensuring correctness can cost you more time when you parallelize the code.

The goal of this lab is to get familiarized with OpenMP directives and What, Where, and When to use them in your code.

For each problem, you are provided with serial code logic. You will implement the parallel version using OpenMP directives as specified.

You can submit a batch job or request interactive resources using srun (recommended)

srun --partition=cmp --nodes 1 --ntasks 1 --cpus-per-task 8 --mem 4G --time 00:15:00 --pty /bin/bash

Now here you can compile the code and run it

g++ -fopenmp -o OpenMP_probs OpenMP_problems.cpp

./OpenMP_probs