CS 420, Parallel Programming, Fall 2019
MP5

Szaday, Justin Shah, Ishan Pankaj

Due Date: November 13th, 2019, at Midnight

1 Overview

As discussed in class, OpenMP can be used to parallelize existing applications. We will be applying this property of OpenMP in this MP, using it to parallelize a program from *Numerical Recipes* – a collection of algorithms across a very broad range of domains. We will be comparing the performance of the program with and without OpenMP, and test its parallel scaling. In addition, four credit-hour students will compare the parallel performance of two implementations of sparse matrix-vector multiplication from the textbook.

2 Getting Started

2.1 Skeleton Programs

Pull the skeleton for MP5 from the `_release` repository: https://github-dev.cs.illinois.edu/cs420-fa19/_release

3 Part A, Parallelizing a "Numerical Recipe"

3.1 Your Task

A *Padé approximant* is the "best" approximation of a function by a rational function, developed by French mathematician Henri Padé in 1890. We have provided you with a program that finds Padé approximants (`pade`), sourced from the book *Numerical Recipes in C*, which also uses the "recipes": LU decomposition (`ludcmp`), LU back-substitution (`lubksb`), and iterative improvement (`mprove`). Your task is to parallelize these functions using OpenMP pragmas, aiming for a 2x speedup for 4 threads for data size 2048. Hint: You can use VTune's hotspots analysis to identify loops that can benefit from parallelization.

Once you have successfully parallelized it and verified its correctness, you may use the small Python-based, benchmarking program we have included to test its parallel performance. See MP3 for reference about how to run it and what to expect from its output.

3.1.1 OpenMP Notes

Adding the "-qopenmp" flag to your CFLAGs during compilation will compile your program with OpenMP. There are two targets in your Makefile, one that compiles the program with

and without this flag. To change the number of threads used by OpenMP, you can set the `OMP_NUM_THREADS` environment variable.

3.1.2 Skeleton Notes

The arguments to the program are, optionally, the size (as an integer, default 2048) and `--verbose`, which prints 21 points of the Padé approximant and the function its approximating. The program produces "correct" results for up to around size 256, then starts including not-a-number's (`pade` is not meant to scale, we are using it in an unusual way in this regard); so, test your program's correctness using smaller data-sizes and do not worry about its results for the larger ones.

? How does parallelizing this program with OpenMP compare to using pthreads?

? Were there any loops in the program that you could not parallelize (e.g. due to size, dependencies, etc.)? If so, pick one and explain why not.

? Were you able to achieve the desired speedup? If not, why not? If so, how could you achieve a higher speedup?

? Briefly comment on the parallel scaling of your implementation for data-size 2048, including a copy of the output of the benchmarking program in your report.

4 Part B, Sparse Matrix-Vector Multiplication (4 Credits Only)

4.1 Your Task

The textbook provides two versions of sparse matrix-vector multiplication (sMVM), known as CRS and JDS, in Section 3.6. It discusses how to parallelize them and their expected parallel performance in Section 7.3. Your task is to implement each version in C and parallelize them using OpenMP, comparing their performance for a variety of thread counts and data-sizes to test the assumptions made in the textbook.

We have provided a skeleton program for this assignment, which loads matrices in both CRS and JDS representation and, additionally, measures the execution time and checks the correctness of your sMVM implementations (against a dense, reference version of MVM), failing when your implementations are incorrect. You can use the `--verbose` flag to print the results from the various implementations of MVM for debugging purposes. We have also provided an updated version of `randMatrix` that generates large, sparse matrices and vectors for this assignment.

? For 4-credit students... did the assumptions made in the textbook about the parallel performance of each version withstand testing? Comment about why or why not.

5 Submission Guidelines

5.1 Expectations For This Assignment

The graded portions of this assignment are your parallelized program from part A (and B for 4 cr. students) and answers to the short-answer questions.

5.1.1 Points Breakdown

- $4 * 10pts.$ — Each Short Answer Question
- $40pts.$ — Correctness of Parallelized *Numerical Recipe* Program
- $20pts.$ — Demonstrated Speedup for Parallelized *Numerical Recipe* Program
- $50pts.$ — Parallelized Mat-Vec Program and Answer to Question (4 cr. only)

5.2 Pushing Your Submission

Follow the git commands to commit and push your changes to the remote repo. Ensure that your files are visible on the GitHub web interface, your work will not be graded otherwise. Only the most recent commit before the deadline will be graded.