CSC/ECE 506: Architecture of Parallel Computers

Program 2: OpenMP Programming
Due: Tuesday, February 18, 2025

Pre-Requisites:

Read the entire program specification carefully before proceeding to start the implementation. This program utilizes the C programming language and the OpenMP API. Although you don't need to be an expert C programmer, basic knowledge of the language is recommended. OpenMP is a set of high-level APIs, with simple directives that are applied to the source code to enable work-sharing constructs.

Instructions:

You have been provided with 2 directories, Serial and Quicksort. Each directory contains a C Code and a Makefile. Carefully read and understand both the codes.

There are two independent tasks that you need to complete in this program. You will apply OpenMP APIs to modify the implementations of both tasks. The learning objective for this assignment is to compare and contrast how different parallelization options in OpenMP can affect performance.

Tool Setup:

We will be using HPC for this program. Log into login.hpc.ncsu.edu with your Unity id/password via ssh in a terminal window (e.g., via PuTTy on Windows) ssh unity_id@login.hpc.ncsu.edu. **This is the login node; you must not execute your code here.**

You should move to the shared storage `cd /share/csc506s25/`⟨your Unity ID⟩. Request an interactive node using `bsub -Is -n 32 -W 10 tcsh`. You can now execute your code.

Task 1: Using OpenMP for calculation *(40 points)*

The below equation is used to calculate Lagrange Interpolation. There is a C Code file provided in the directory Serial named `serial.c`

$$x = \sum_i \left\{ \left(\sum_j \left(\prod_{k \neq j}^{k} \frac{x - X[k]}{X[j] - X[k]} \right) \times M\left[i\right]\left[j\right] \right) \right.$$

You should attempt to compile this code and verify its correctness before proceeding with your deliverables. Build and run the program using the commands below:

```
make clean
make
./serial
```

1. *(5 points)* OpenMP has a defined function to get clock time in seconds. Look up this timing function and call it in the commented placeholders provided in the code. (Search for variables `start_time` and `end_time` in the code and place them appropriately.)
2. *(30 points)* Parallelize iterations of the nested **`for`** `i`, `j`, and k loops. First, you have to parallelize individual `i`, `j`, and k loops and then parallelize all the loops together (Search for how you can parallelize loops). **Make sure that the value of `sum_avg` in the output remains the same as the original every time you run your code.** Find out how this is done in OpenMP and work on it accordingly.
3. *(5 points)* Vary the number of threads (1, 2, 4, 8) for each parallelized iteration and report the timings that you get using the above timing functions in a table format, as below

To vary the number of threads you need to give the thread_count in the argument to the serial program.

```
./serial 1
./serial 2
./serial 4
./serial 8
```

	Threads = 1	Threads = 2	Threads = 4	Threads = 8
Parallelize `i` loop only				
Parallelize `j` loop only				
Parallelize `k` loop only				
Parallelize all loops				

Task 2: Using OpenMP for Sorting *(40 points)*

You can find the implementation of the Quicksort Algorithm in the Quicksort directory.
You should attempt to compile this code and verify its correctness before proceeding with your deliverables. Build and run the program using the commands below:

```
make clean
make
./sort
```

1. *(5 points)* OpenMP has a defined function to get the clock time in seconds. Look up this timing function and substitute it for the commented placeholders provided in the code. (Search for variables “start_time” and “end_time” in the code and place them appropriately.)
2. *(25 points)* Add an OpenMP parallel pragma in the provided placeholder and change scheduling policies to Static, Dynamic, and Guided and vary number of threads (1,2,4,8) and report timings in a table format below.

   To vary the number of threads you need to give the thread_count in the argument..

   ```
   ./sort 1
   ./sort 2
   ./sort 4
   ./sort 8
   ```

	Threads = 1	Threads = 2	Threads = 4	Threads = 8
Static scheduling				
Dynamic scheduling				
Guided scheduling				

3. *(10 points)* Performance optimization. Find out ways in which you can optimize the code to minimize the overall execution time of the program. Edit your code accordingly. You can use whatever optimization policy you can come up with. This question will be graded according to the best optimization achieved. The highest optimization will get 10 points. **Remember do not change the sorting logic in the code.**

Report *(20 points)*

Your report should be at most three pages.

1. Include the table for Task 1, Question 3.
2. Include the table for Task 2, Question 2.
3. Report the achieved optimized execution time.
4. *(10 points)* Explain what a scheduling policy in OpenMP actually means. What are the different types of scheduling policies in OpenMP and which is the default one?
5. *(10 points)* What is privatization and Reduction and what clauses in OpenMP can achieve them?

Submission

Name your submission directory as ⟨unityid⟩`_pgm2`, where ⟨unityid⟩ is your Unity ID. Compress and upload it. Your submission must include the following.

1. Serial directory that includes the `Makefile` and `serial.c` (it should contain the timing functions, and all the iterations (`i`, `j`, `k`) parallelized)
2. Quicksort directory that includes the `Makefile` and `sort.c` (it should contain the timing functions and your optimized code)
3. The report.