LAB 3: OpenMP (in C++)

Implementation Due**: 03 / 07 / 2017 - 11:55 pm**

Part A (Sobel algorithm):

Implement the same Sobel Edge Detection algorithm as you did for the first lab assignment using OpenMP. The testcases and requirements still remain the same, just the implementation needs to be modified for the OpenMP version of the program. Chunk Size for both parts below is passed from command line.

Part A1:
Use static loop scheduling to process the edge detection with OpenMP parallel for. Print the information about the threads and the starting point of chunks (Example: Thread 1 -> Processing Chunk starting at Row 50) that they processed. Store the thread Id and the starting points in a variable and print it after output is written to the PGM file.

Part A2:
Use dynamic loop scheduling process the edge detection with OpenMP threads. The OpenMP uses a chunk in a similar way to how we used it in Lab 1. Each chunk is of equal size (except for the last chunk, which may be smaller). Again, print the information about the threads and the starting point of chunks that they processed.

Part B (Analysis):

Compare the performance of your implementation (on Openlab) for Sobel algorithm from Lab 1 (C++ threads) with Part A2 of this lab. Both feature dynamic scheduling and variable chunk sizes. Be careful when timing your code, you want to time only the time for processing the image.
To time the C++ thread implementation's "dispatch_threads" function (from lab 1 skeleton), use *omp_get_wtime()* function, just like Part A skeleton. To make this timing function work you will have to add #include <omp.h> in the source file and -fopenmp flag for that in addition to -pthread flag while compiling. Print the timing before the program ends.
Repeat each measurement experiment 5 times and use the smallest time observed.

Report the performance for all 5 test cases with 2, 4 and 8 threads. Do it for chunk size of 25 and 70.

Submit via EEE Dropbox in 2 parts: *YOU **MUST** USE the file names below:*

1. Part A1 and A2 in the same **Implementation**.cpp file.
2. Part B as a PDF file name **Analysis.pdf**

Point Breakdown:
Part A Implementation 50pts, Part B Analysis 20pts.

OpenMP how-to

C work sharing construct:

#pragma omp parallel for schedule *(type [,chunk]),* private *(list),* shared *(list)*

Use schedule type STATIC for part A1 and DYNAMIC for part A2. Private variables in the list will be private to each thread.

Compiling an OpenMP program:

```
$ g++   -fopenmp   your_program_name.c
```

Useful OpenMP functions:

- Specify the number of threads for your program using omp_set_num_threads. But do this OUTSIDE the parallel sections. Note that the system may override this specification and give your program fewer threads!
- Use the OpenMP function omp_get_num_threads() to get the number of threads in use but call this function inside a parallel region.
- To get the identifier of a running thread use the OpenMP function omp_get_thread_num(). The function returns an integer. You can assign the returned integer to a variable thread_id and later use the variable to control the execution flow of your program, i.e.
  if (thread_id == 0) {...} else {...}.
- You may need to create variables private to an OpenMP thread (i.e. thread_id). First declare the thread_id variable and next to use the #pragma omp parallel for private(th_id) statement. In this way you do not need to worry about the overwriting of the variable thread_id.
- Do not use I/O statements inside the parallel region.

NOTE: The TA cannot solve the exercises for you, but he can answer your questions!

USEFUL LINKS:
http://en.wikipedia.org/wiki/OpenMP (very easy and useful)
https://computing.llnl.gov/tutorials/openMP/ (much more complex but complete)
http://openmp.org/mp-documents/omp-hands-on-SC08.pdf (very good tutorial)
http://openmp.org/mp-documents/OpenMP4.0.0.Examples.pdf (very detailed examples)