CS3210 – Parallel Computing (AY 2017/2018 Sem 1)

Assignment 1
Performance Improvements and Speedup Analysis (15 marks)

Individual Submission due on **Fri, 13 October 2017, 2pm**

This assignment, covering lectures 1 to 5, is designed to enhance your understanding of multithreaded programming, OpenMP, performance and speedup analysis.

In question 1, you are given a sequential implementation in C for blurring an image. You will work towards reducing the execution time by using multithreading programming (POSIX threads) and OpenMP.

Question 2 focuses on fixed-workload and fixed-time performance speedup.

All timing measurements submitted for this assignment must be collected on the computer nodes in our lab. You can develop your programs offline on your own notebook or PC but the results submitted must be measured on **node1 in our lab (Dell Optiplex tower system with Intel i7 8-core).**

[*Optional Performance Challenge:* For Question 1, besides the above, you are encouraged to explore other forms of optimization and bonus marks will be awarded. This is an opportunity to score more than full marks.]

Question 1. Performance Improvements with Threads and OpenPM (Total: 7 marks)

Compilation and Execution

Get the "**`blur_seq.c`**" from the web:

```
$wget www.comp.nus.edu.sg/~ccris/cs3210/a1.zip
$unzip a1.zip
```

Unzip the archive and Compile the sequential program:

```
$gcc -O3 blur_seq.c -o blur_seq –lm
```

Note:

- Use gcc version 4.8.4 (the version installed on node1 machines in the lab).
- Use –O3 flag during compilation to obtain maximum level of optimizations from the gcc compiler.

Run the sequential version with the following inputs: "image1.bmp", σ = 3 and Kernel size = 9 to obtain "out_image1.bmp" as output:

```
$./blur_seq image1.bmp 3 9 out_image1.bmp
```

Take note of the sequential timing T_{seq}.

The sequential program applies a Gaussian blur to a Bitmap (.bmp) image. The Gaussian blur is a type of image-blurring filter that uses a Gaussian function for calculating the transformation to apply to each pixel in the image. The equation of a Gaussian function in one dimension is:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

You may find additional details about the Gaussian blur here:

1. https://en.wikipedia.org/wiki/Gaussian_blur
2. https://en.wikipedia.org/wiki/Convolution

The sequential program we provided first reads the image from the file (each pixel is defined by three bytes representing Blue, Green and Red, respectively). Next it computes the Kernel values (G(x)), and applies the kernel to each pixel on each row of the image to obtain an intermediate image results. Later, it applies the same Kernel to each pixel on the columns of the intermediate result. The image result is written in Bitmap file result. Please note that the program uses a linear (1D) kernel, not matrix (2D) kernel.

Your Task

- Optimize the sequential version for better execution time using POSIX threads (up to 3 marks). Name your program **blur_threads.c**
- Optimize the sequential version for better execution time using OpenMP (up to 7 marks). Name your program **blur_omp.c**
- Derive the speedup achieved and explain your optimizations. **Marks will not be awarded if the performance results obtained are not explained**.
- You may change anything in the sequential implementation, as long as you apply the same kernel in the same way to the image. You may find that may other optimizations can be made to the code (not only related to parallelization).
- Output image obtained from your program should be identical with the output image produced by the sequential program for the same set of input arguments.

Note:

- Use an image size of **at least 512 x 512 pixels.**

Evaluation

Marks:

POSIX threads:

- Up to 2 marks are awarded if you reduce the execution time by correctly parallelizing parts of the sequential code using POSIX threads. The execution time should be consistently reduced in your new implementation.
- Up to 3 marks are awarded if you further reduce the execution time by applying Advanced Vector Extensions (AVX instructions). You might need to add one byte of padding for each 3 bytes of each pixel (BGR) before applying AVX. You might choose to apply the AVX only for some of the computations. Please find details here: https://software.intel.com/en-us/articles/iir-gaussian-blur-filter-implementation-using-intel-advanced-vector-extensions

OpenMP:

- Up to 6 marks are awarded if you reduce the execution time by correctly parallelizing parts of the sequential code using OpenMP. The execution time should be consistently reduced in your new implementation.
- Up to 7 marks are awarded if you further reduce the execution time by applying Advanced Vector Extensions (AVX instructions). You might need to add one byte of padding for each 3 bytes of each pixel (BGR) before applying AVX. You might choose to apply the AVX only for some of the computations. Please find details here: https://software.intel.com/en-us/articles/iir-gaussian-blur-filter-implementation-using-intel-advanced-vector-extensions
-

Bonus:

POSIX threads:

- 1 bonus mark for the two best submissions in terms of performance

OpenMP:

- 1 bonus mark for the two best submissions in terms of performance

Question 2. Performance Evaluation (5 marks)

The aim of this assignment is to carry out a simple performance analysis of a shared-memory program on a multicore system.

Compilation and Execution

The blur_seq.c, blur_threads.c and blur_omp.c should be compiled and executed in the same way as question 1.

You can control the processor core(s) used in the execution via the **numactl** command.

For example, to run the program *blur_omp* on core 3:

```
$ numactl --physcpubind=3 ./blur_omp …
```

To run the program **blur_omp** on four cores (from core 0 to core 3):

```
$ numactl --physcpubind=0-3 ./blur_omp …
```

Your Tasks

a. Run the thee programs on **node 1** in our lab (Dell Optiplex tower system with Intel i7 8-core), measure the program execution time for a **matrix size of 4096** and complete the table below:

Number of cores	Execution Time		Speedup		
	Threads	**OpenMP**	**using T_1(seq)**	**using T_1(threads)**	**using T_1(OMP)**
1	T_1(sequential), T_1(threads)	T_1(OMP)			
2	T_2	T_2			
4	T_4	T_4			
6	T_6	T_6			
8	T_8	T_8			

Note:

- **T_1(sequential)** is the execution time of **`blur_seq.c`** on one core.
- **T_1(parallel)** is the execution time of **`blur_omp.c`** with **64 threads** on **one core.**
- **T_n (n=2,4,6,8)** is the execution time of **`blur_omp.c`** with **64 threads** on **n cores**.
- When measuring the execution time make sure that you do not output any info at the terminal as this will considerably increase your execution time.

b. **Execution time:** Is there any difference between T_1(sequential), T_1(threads), and T_1(OMP)? Explain your answer for each pair of values.

c. **Speedup:** Comment on the speedups obtained using T_1(sequential) and T_1(OMP).

d. **Number of Threads**: For the shared-memory program, what should be an appropriate number of threads if we use eight cores? Devise an experiment to determine the number of threads. Explain your answer and assumptions if any.

e. **Overhead of Threading**: Devise an experiment to determine the average threading overhead in terms of overhead time per thread.

Deliverables and Submission

Prepare a **zip file** using your **student id (matric no), e.g.** A01234567X.zip, which contains:

`blur_threads.c` `blur_omp.c`	Your optimized version with appropriate comments, modularization and indentation. Please **remove all debug messages**.
`makefile` / `readme`	For compiling your `blur_threads.c` and `blur_omp.c`.
`assignemnt1_report.pdf`	Contains: 1. Details about your programs (how they work). **Include details on how to reproduce your result, e.g. inputs, execution time measurement etc.** 2. Any special consideration or implementation detail that you consider non-trivial. 3. Answers for Question 2 with supporting arguments and data. Provide additional information for us to reproduce your experiment, e.g. **input values (image size, σ, kernel size), how to measure execution time and to compute speedup, etc.**

1. This assignment is to be done on an individual basis. You can discuss the assignment with others as necessary but in the case of plagiarism both parties will be severely penalized.

2. The zip archive must be uploaded to IVLE in the workbin folder "Assignment1" by **Fri, 13 Oct, 2pm. Penalty of 50% per day for late submissions will be applied.**