ECSE 420 - Parallel Computing
Student: Daniel Macario
ID: 260503662

Programming Assignment 1

1. Binarization Algorithm

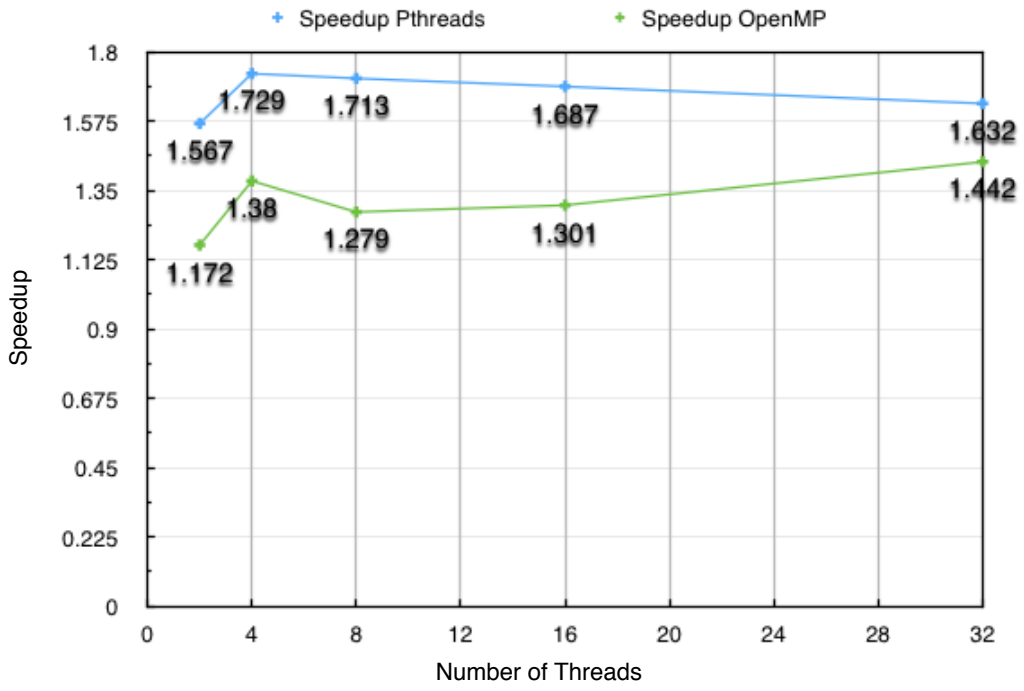

Diagram 1. Image binarization speedup through parallelization using Pthreads and OpenMP.

As shown in diagram 1 above, both the Pthreads and OpenMP parallelization algorithms resulted in a speedup greater than 1 for all five thread counts. The maximum speedup of the Pthreads implementation took place when the binarization algorithm was executed on 4 threads, which was 1.729 times faster than the sequential version. The OpenMP implementation peaked in performance when executed on 32 threads - execution was 1.442 times faster than the sequential version. The second fastest speedup of the algorithm in the OpenMP implementation occurred when it was executed on 4 threads, which was 1.38 times faster than the sequential version.

The tests were executed on an Intel core i7, which is a 64-bit dual core processor that utilizes Intel's proprietary hyper-threading technology which maps each physical core to two virtual cores in the operating system. This explains the two peaks observed when the algorithm was executed on 4 threads.

The size of the executables for the Pthreads and the OpenMP implementation are 105,512 and 105,352 bytes, respectively. The similarity in size is probably due to the fact that OpenMP is built on top of the threading library supplied by the operating system, which in the case of my computer, a macbook pro, is Pthreads. After compilation, both executables are leveraging the same threading library.

An important point to note is the difference in lines of code. The Pthreads implementation of the binarization algorithm is 102 lines of code, while the OpenMP implementation is only 58. OpenMP is seen as a higher-level threading library when compared to Pthreads, which offers a finer level of thread control at the expense of being more verbose.

Diagram 2. Photo before binarization.

Diagram 3. Photo after binarization.

2. Sobel Filter Algorithm

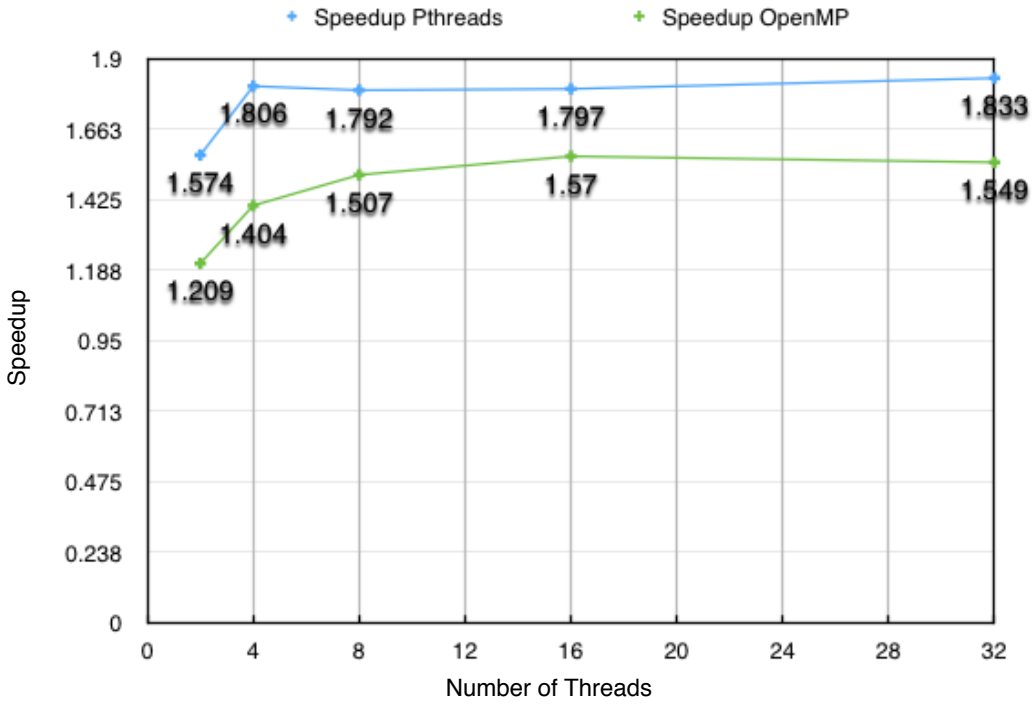

Diagram 4. Sobel filter speedup through parallelization using Pthreads and OpenMP.

Diagram 4 above demonstrates the speedup obtained from running a Sobel filter on five different thread counts. For all thread counts the speedup was greater than 1. The Pthreads parallelization peaked when executed on 32 threads, which was 1.833 times faster than the sequential version. However, the diagram shows that increasing thread counts had no significant impact in the speedup produced when executed using more than 4 threads. The speedup from the OpenMP version peaked when executed on 16 threads, which was 1.57 times faster than the sequential version. The same pattern discussed above is observed again, although it occurs when the algorithm is executed on 8 or more threads.

The diagram indicates that the speedup obtained from parallelizing the Sobel filter plateaus when executed on more than 4 or 8 threads, depending on the threading library used. Introducing more physical cores would overcome this limitation by increasing the level of real parallelism (not time-slicing) that can be achieved.

The Pthreads executable is 105,512 bytes, and the OpenMP executable is 105,352 bytes. The same discussion presented in part one applies here: OpenMP is built on top of the threading library provided by the operating system, which in my case is Pthreads. The two programs are both using the same threading library after compilation, which is likely the reason that their sizes are practically the same.

The Pthreads implementation was written in 136 lines of code, while the OpenMP implementation is 97 lines of code. As mentioned in part one, OpenMP is a higher-level threading library which allows for practical management of threads without the fine-grained control offered by Pthreads.

Diagram 5. Photo after applying the Sobel filter.