Analysis of the Execution Time Variation of OpenMp+MPI with Canny Edge Detection and Feature Detector

Marco Lin
University of the Pacific

Hight Performance Computing
m_lin23@u.pacific.edu

***Abstract*—The purpose of this report is to compare the difference of execution time between serial execution and parallel execution. In order to analyze this, we will implement Canny Edge Detection and Feature Detector first and, and then using open MP and open MPI to parallelize it.**

Keywords—C, Canny Edge Detector, open MP, parallel

I. Introduction

The purpose of this report is to analyze the speedup and parallel efficiency by designing a serial and parallel code. In Canny Edge Detector and feature detector, we have to implement convolution, Suppressed, and Hysteresis. In those function, the program needs to get and calculate the value of each pixels. This process takes up most all of execute time thus by using to parallelize it, we can effectively to observe the reducing of execution time. In addition, running our program on cluster which provide multiple processors allows us to observe the changing of execution time between different numbers of threads and different numbers of processors. (Numbers of threads from 2 to 32 and Numbers of processors from 2 to 32)

II. Implementation

A. Canny edge detector with Open MPI and Open MP

We used this function to detect the edge of image. In order to effectively reduce our execution time. We used Open MPI and OpenMP to implement our parallelization. In Open MPI process, we need to consider the right communication between each processor for sending and receiving the data.

In the Figures 1 to 15, it appears that the correlation between efficiency and processors and threads. When we used 4 processors to performance it, the efficiency up to 160 percent. The range of efficiency is from 90 (2 processors) to 300 (32 processors). And, Range of speed up is from 1.5 to 9.

B. Parallelization Methodology

By setting two time zones: end-to-end time and computation time to measure the execution time in serial code and parallel code. we parallelized the for loop most because for loops take up most of all execution time. The environment we used to run the program is a cluster which allows us to use multiple processors. The sizes of images we use were: 12800, 10240, 7680, 4096, 2048, to 1024. The value of Sigma were 1.25, 1.1, 0.6. The threads we test were 2, 4, 8, 16, 32. The processors were from 2 to 32.

III. Results and Analysis

We have the figures from 1 to 15 to show our Speed-up time and Efficiency results in each sigma, thread, and processors.

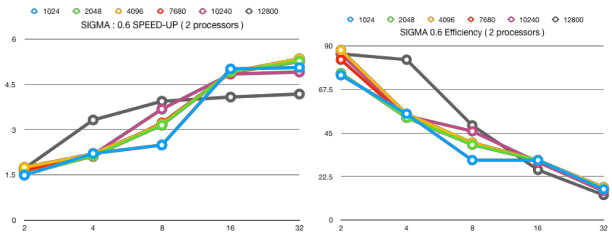

Figure 1.

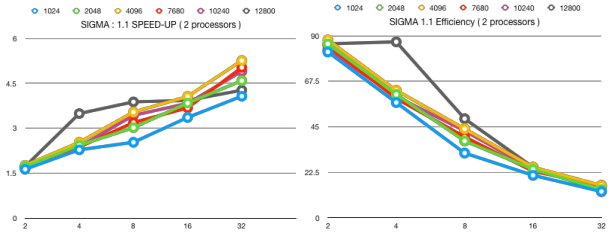

Figure 2.

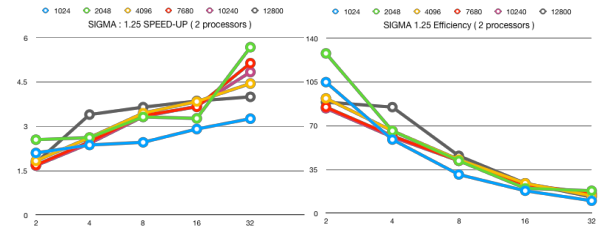

Figure 3.

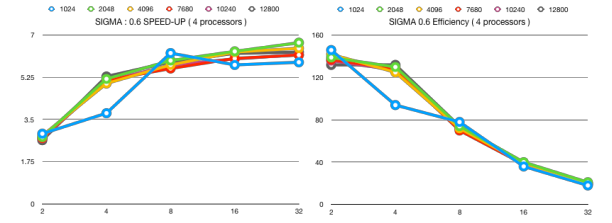

Figure 4.

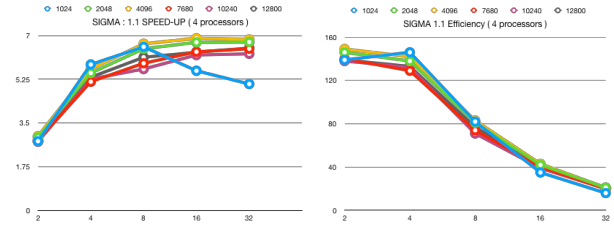

Figure 5.

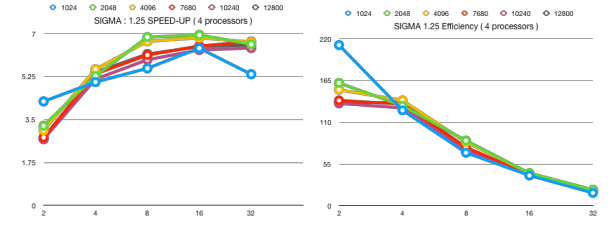

Figure 6.

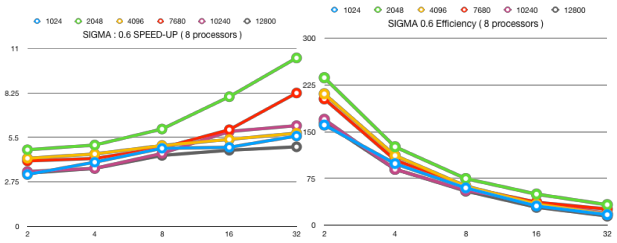

Figure 7.

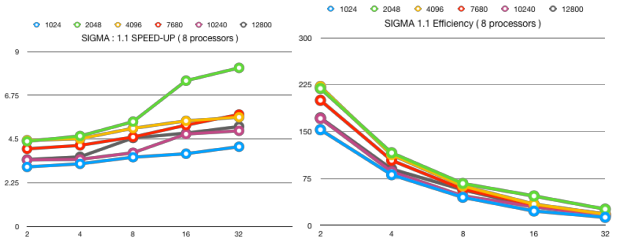

Figure 8.

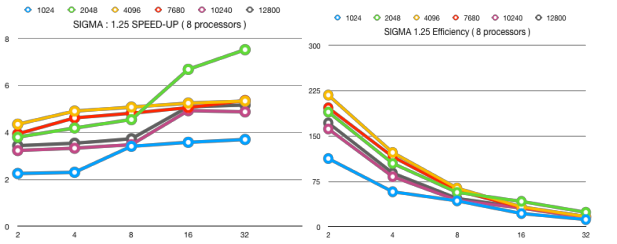

Figure 9.

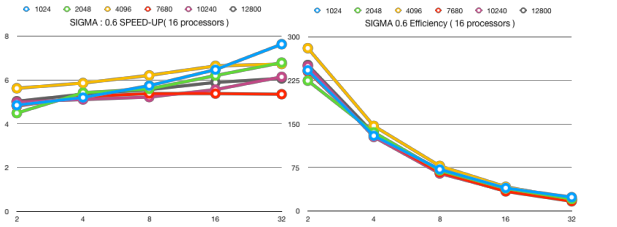

Figure 10.

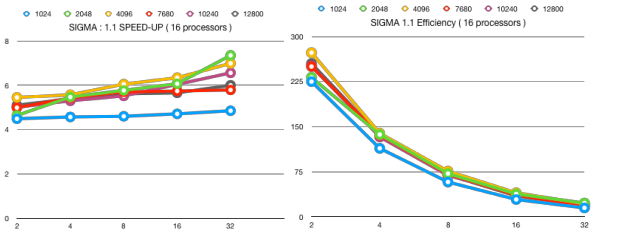

Figure 11.

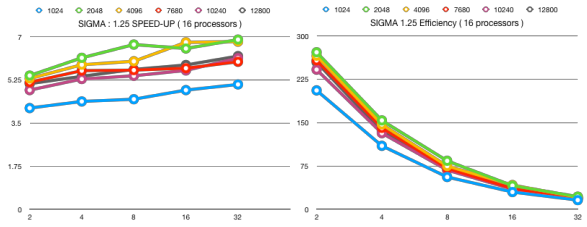

Figure 12.

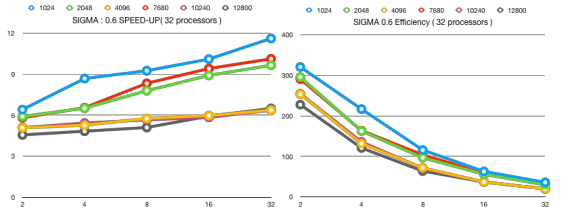

Figure 13.

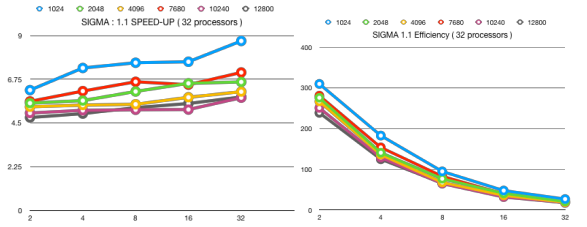

Figure 14.

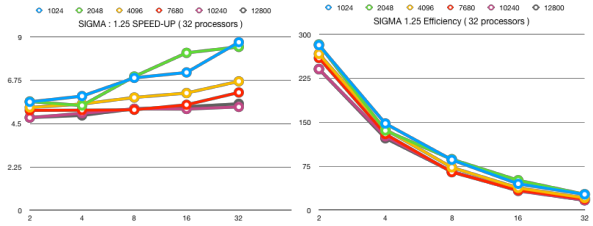

Figure 15.

The execution time are affected by sigma value, size of image, number of processors, and number of threads. According to the execution time we got, we show the Speed-up time and Efficiency with different processors and threads.

IV. Conclusion

Along with this project, we could understand the execution time will affected by not only the condition we set but also the condition of cluster. Moreover, the problem of segmentation is a critical issue we need to be careful. When I run my code on 2 processors, there was no problem. But, when I run it on 8 processors, the program showed me segmentation fault. One of the reasons caused that problem was I set the inappropriate bound in for loop. So, this project can help us to understand not only how parallelization is but also how segmentation is.