**Lab 1 Report**

**Testing Conditions:**

The programs for rectification, pooling and convolution were tested on a machine with the following specifications:

* 6 core Intel(R) Xeon(R) CPU E5-2640 0 @ 2.50GHz.
* x86\_64 architecture
* CPU Clock Frequency: 2500.00 MHz.
* Ubuntu 16.04.1 LTS (Xenial Xerus) operating system.

**Method:**

We tested each particular process using 1,2,4,8,16 and 32 threads. The function that executes the particular process (rectification, pooling or convolution) is called 100 times in order to clearly show the benefits of parallelization.

Previously, we called the function that loaded the old image and saved the new image 100 times as well. When this occurred we would see little change in the total run time of the program since most of the work done was loading and saving the image with was happening serially.

To test the time taken for each function we wrote the following:

time ./<executable\_name> <test\_image>.png <test\_result>.png <number of threads>

The output of the program would look something like the following:

real 0m1.874s

user 0m1.816s

sys 0m0.008s

We then recorded the value of the program execution time as the **real time**. We didn’t use the sum of the user and system time since this measurement will give you the sum of time that each thread spent in user and system mode, which will not give an accurate measurement of improved performance.

After executing the programs we would also run an equality check using the equality program provided and would validate that the output was correct.

**Limitations:**

Quoting from the man page for the time command:

*The elapsed time is not collected atomically with the execution of the program; as a result, in bizarre circumstances (if the* ***time*** *command gets stopped or swapped out in between when the program being timed exits and when* ***time*** *calculates how long it took to run), it could be much larger than the actual execution time.*

We attempted to remove this limitation of our measurement tool by executing each test multiple times. While this doesn’t necessarily remove the error in our measurement it at least improves our precision.

**Results:**

**Rectification**

Table 1 – Rectification Results

|  |  |  |  |
| --- | --- | --- | --- |
| Trial # | # of Threads | Real Time(seconds) | Equality? |
| 1 | 1 | 2.603 | Yes |
| 2 | 1 | 2.816 | Yes |
| 3 | 1 | 2.597 | Yes |
| 4 | 1 | 2.609 | Yes |
| 5 | 2 | 1.831 | Yes |
| 6 | 2 | 1.803 | Yes |
| 7 | 2 | 1.806 | Yes |
| 8 | 2 | 1.797 | Yes |
| 9 | 4 | 1.543 | Yes |
| 10 | 4 | 1.504 | Yes |
| 11 | 4 | 1.516 | Yes |
| 12 | 4 | 1.482 | Yes |
| 13 | 8 | 1.485 | Yes |
| 14 | 8 | 1.522 | Yes |
| 15 | 8 | 1.651 | Yes |
| 16 | 8 | 1.569 | Yes |
| 17 | 16 | 1.534 | Yes |
| 18 | 16 | 1.682 | Yes |
| 19 | 16 | 1.551 | Yes |
| 20 | 16 | 1.668 | Yes |
| 21 | 32 | 1.645 | Yes |
| 22 | 32 | 1.662 | Yes |
| 23 | 32 | 1.525 | Yes |
| 24 | 32 | 1.543 | Yes |

Figure 1 - Rectification

**Pooling**

**Convolution**

Table 2 - Convolution Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Trial # | | # of Threads | | Time(seconds) | | Equality? |
| 1 | | 1 | | 57.011 | | Yes |
| 2 | | 1 | | 57.026 | | Yes |
| 3 | | 1 | | 57.297 | | Yes |
| 4 | | 1 | | 57.129 | | Yes |
| 5 | | 2 | | 36.47 | | Yes |
| 6 | | 2 | | 36.326 | | Yes |
| 7 | | 2 | | 36.308 | | Yes |
| 8 | | 2 | | 36.707 | | Yes |
| 9 | | 4 | | 20.787 | | Yes |
| 10 | | 4 | | 20.829 | | Yes |
| 11 | | 4 | | 21.147 | | Yes |
| 12 | | 4 | | 21.278 | | Yes |
| 13 | | 8 | | 19.366 | | Yes |
| 14 | | 8 | | 19.339 | | Yes |
| 15 | | 8 | | 19.201 | | Yes |
| 16 | | 8 | | 19.02 | | Yes |
| 17 | | 16 | | 17.841 | | Yes |
| 18 | | 16 | | 17.923 | | Yes |
| 19 | | 16 | | 18.096 | | Yes |
| 20 | | 16 | | 18.911 | | Yes |
| 21 | | 32 | | 18.282 | | Yes |
| 22 | | 32 | | 18.531 | | Yes |
| 23 | | 32 | | 18.072 | | Yes |
| 24 | 32 | | 17.566 | | Yes | |

Figure - Convolution

**Analysis and Conclusion**

Looking at Figures 1, 2 and 3 you can see there is an asymptotic relationship between the improvement in the program’s performance and an increasing number of threads.

The performance seems to plateau at around 5-6 threads. A reasonable justification may be that since we are testing these programs on a system with 6 cores, any additional threads are not improving the parallelization of the program. We may still see improvements in the real time execution of the program because we are increasing the chance that the CPU context switches to our application and runs our process but in general we see this value plateau with only small increments in performance.