**Parallel Programming Skills Report:**

A screenshot of a computer

Description automatically generated

while following the directions provided in parallel programming tasks I ran into an error. while using the following command line at first “g++ -o dd\_omp dd\_omp.cpp -lm -fopenmp -ltbb -lrt” I got an “tbb” error which would not compile and was having trouble finding the tbb library. After some research I resolved this issue by manually installing tbb and setting flags and environment variables following this guide:

<https://www.theimpossiblecode.com/blog/intel-tbb-on-raspberry-pi/>

After installing the tbb libraries ran into an error again about “arm architecture version armv7-a”. which was solved by installing opencv and pip, using following guide:

<https://www.pyimagesearch.com/2018/09/19/pip-install-opencv/>

A screenshot of a cell phone

Description automatically generated

Running the program shown in the example (./dd\_method 1) sets max ligands to 1 with default value 120 to nLigands and nThreads

The Results:

|  |  |
| --- | --- |
| Implementation | Time(s) |
| dd\_serial | 129.972 |
| dd\_openmp | 128.876 |
| dd\_thread | 136.050 |

The run times looks pretty close to dd\_serial. While using one thread the run time is the fast. Including libraries and more complicated program structure is most likely the reason behind these differences in times.

Increasing number of threads in openmp1 (Threads from 2 to 5):

A screenshot of a computer

Description automatically generated

Increasing number of threads in cplusthreads (Threads from 2 to 5):

A screenshot of a cell phone

Description automatically generated

Results:

|  |  |  |  |
| --- | --- | --- | --- |
| Implementation | Time(s) 2 Threads | Time(s) 3 Threads | Time(s) 4 Threads |
| dd\_omp | 114.043 | 97.786 | 81.113 |
| dd\_thread | 78.679 | 55.158 | 42.094 |

The times start to drop with the addition of more threads, especially for the dd\_threads implementation. The time dd\_thread takes with 2 threads is faster than dd\_opm can do with 4. However, we start to see diminishing returns with the increase of threads in dd\_threads.

Questions:

1. Which approach is fastest?

* Depends on the number of threads you plan on using. With one thread you are better off using dd\_serial as it will save just a little bit of time since it doesn’t contain as much code and libraries. However, if you want the flexibility of using more threads dd\_threads using tbb is the fastest and is superior to dd\_omp.

1. Determine the number of lines in each file (use wc -l). How does the C++11 implementation compare to the OpenMP implementations?

A screenshot of a cell phone screen with text

Description automatically generated

* As shown in this image, the word count of dd\_omp.cpp is 194 and dd\_threads.cpp is 208. There is 13 word difference between the two, which is pretty good how much quicker dd\_threads is compared to dd\_omp.

1. Increase the number of threads to 5 threads. What is the runtime for each?

A screen shot of a smart phone

Description automatically generated

A screen shot of a social media post

Description automatically generated

|  |  |
| --- | --- |
| Implementation | Time(s) 5 Threads |
| dd\_omp | 75.270 |
| dd\_threads | 42.656 |

* It seems almost no change in time while running threads 5. This is because the raspberry pi is quad core, meaning it can handle 1 thread per core, which makes it run faster.

1. Increase the maximum ligand length to 7, and rerun each program. What is the run time for each?

* The DEFAULT\_max\_ligand was already set at 7 and is what I used for my test to find the run time. Compared to times for max\_ligand < 7 the runtime is much greater.

Runtime for max\_ligand 7

|  |  |
| --- | --- |
| Implementation | Time(s) |
| dd\_serial | 129.972 |
| dd\_openmp | 128.876 |
| dd\_thread | 136.050 |