**Michael Knight**

**Gitter-Geeks**

**Project: A5**

**Task 3: a) Parallel Programing Skills**

**What are the basic steps (show all steps) in building a parallel program? Show at least one example.**

Identifying sets of tasks that can run concurrently and can be processed concurrently, Assigning tasks evenly or balancing the load, Orchestration “synchronizing the processes with one another, and mapping the assigned processes to the processor. An example is a super computer.

**What is MapReduce?**

MapReduce is a programming model and an associated implementation for processing and generating big data sets with parallel programming

**What is map and what is reduced?**

Map or Lisp takes a input function and a sequence of values and then applies the function to each value in the sequence.

Reduce combines all the elements in the sequence using binary operations.

**Why MapReduce?**

Very cost-efficient, and time-efficient

**Show an example for MapReduce.**

Crawled documents, or web request logs.

**Explain in your own words how MapReduce model is executed?**

1. MapReduce library first splits up the input files into pieces that can be broken up. 2
2. While 1 copy goes to the master the rest are sent to the workers where the master assigns the work, the workers that are idle are picked.
3. Worker reads the contents of the task it was assigned and passes the information to the map function along with a key/value.
4. Periodically the pairs are saved and sent to the master.
5. A Reduce worker is notified about the locations by the master and reads the data and sorts it by the keys.
6. After a Reduce worker goes over all the sorted data for each key it was assigned it passes the key and values to the reduce function.
7. Once all tasks are finished the master wakes the user program and returns the user code.

**List and describe three examples that are expressed as MapReduce computations.**

Distributed Grep: A function that produces results that matches a given pattern.

Reverse Web-Link Graph: The map produces a <target, source> pair for each link to a target URL

Count of URL Access Frequency: Map processes logs of web page requests and produces <URL, “Total count> pair depending on how many values for the same URL were requested.

**When do we use OpenMP, MPI, and MapReduce(Hadoop), and why?**

OpenMP: When you want to use parallel programming, because its efficient, neat and powerful.

MPI: “Message Passing Interface” used to develop parallel scientific applications. Because they tightly synchronous code and well load balanced.

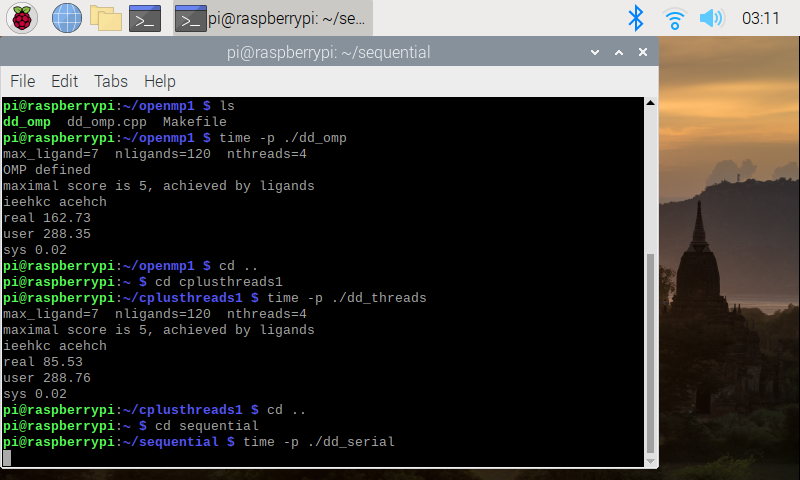
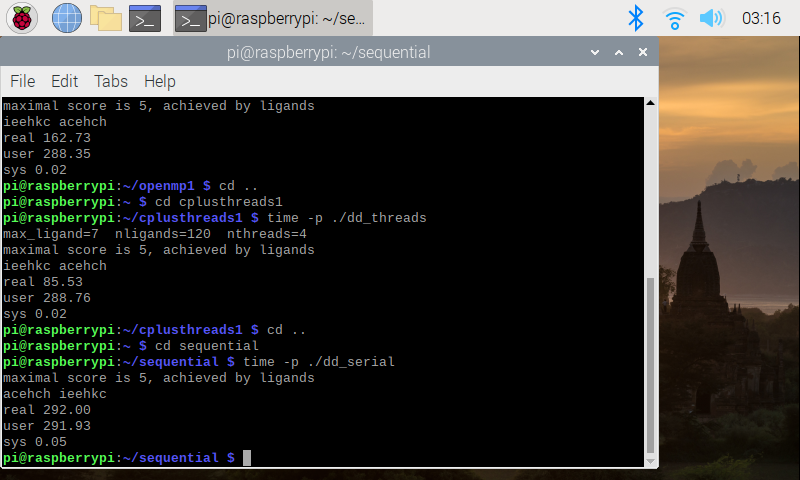
MapReduce(Hadoop): When you have large amounts of data to extract, transform, and load. Because MapReduce can be used to write tightly coupled scientific applications as well.

**In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**

A drug design is the implantation to generate ligands for a protein, but not only that but the best possible outcome. DNA problem is that it is proteins in our body that are shaped in unique ways depending on their functions.

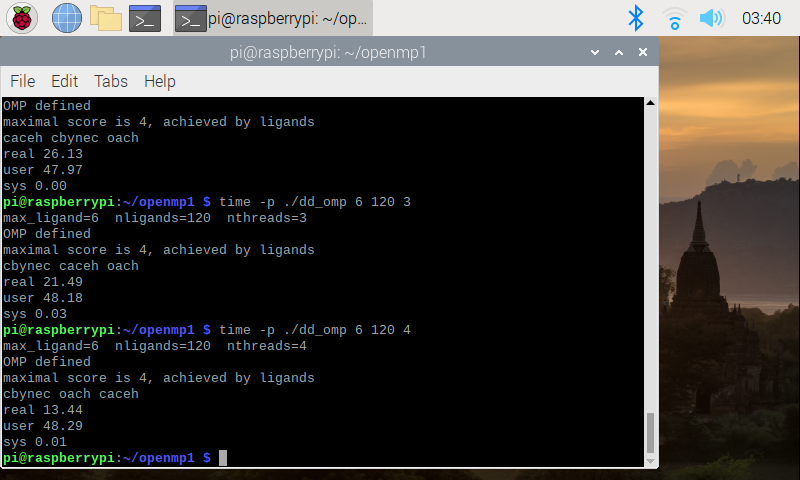
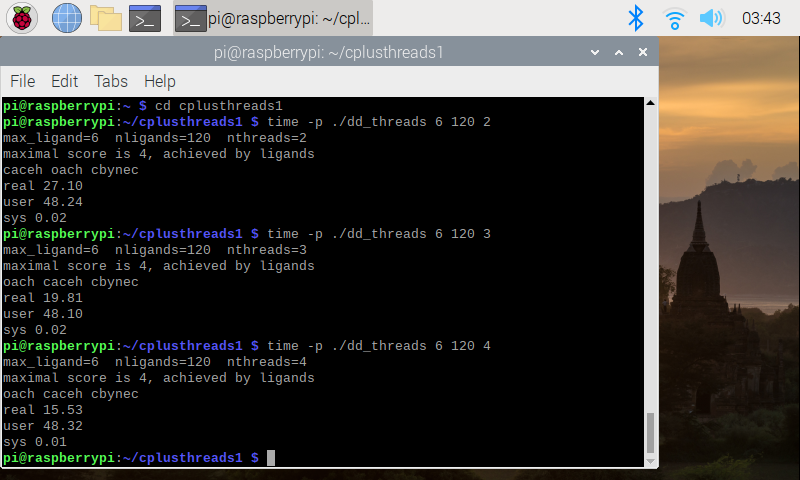
**b) Parallel Programming Basics:**

**Task 4:**

**First I ran the running time of each implementation and recorded its results**

|  |  |
| --- | --- |
| **dd\_serial** | **292.00** |
| **dd\_omp** | **162.16** |
| **dd\_threads** | **85.53** |

**Next I increased the number of threads for the implementation to run with and recorded it’s results:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Implementation** | **Time(s) 2 Threads** | **Time(s) 3 Threads** | **Time(s) 4 Threads** |
| **dd\_omp** | **4** | **21.49** | **13.44** |
| **dd\_threads** | **27.1** | **19.81** | **15.53** |

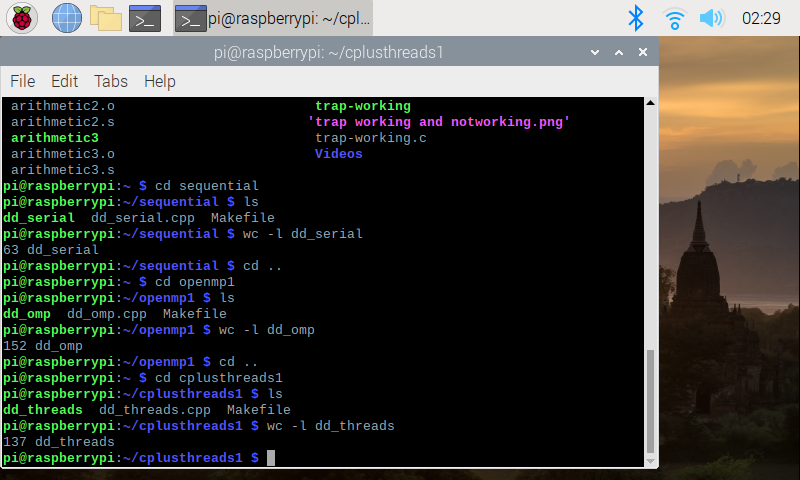
**2.3 Discussion Questions:**

**1. Which approach is the fastest?**

Overall the OpenMP implementation is fastest

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

**Discovered the number of lines each implementation had and recorded it.**

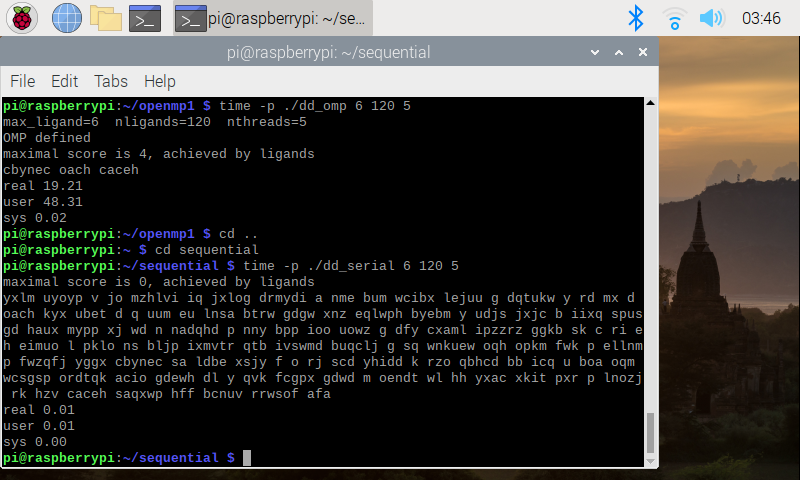
****

|  |  |
| --- | --- |
| **Implementation** | **Number of lines** |
| **dd\_serial** | **63** |
| **dd\_omp** | **152** |
| **dd\_threads** | **137** |

**C++ seems more simplified compaired to OpenMP, while OpenMP seems more complex**

**3. Increase the number of threads to 5 threads. What is the run time for each?**

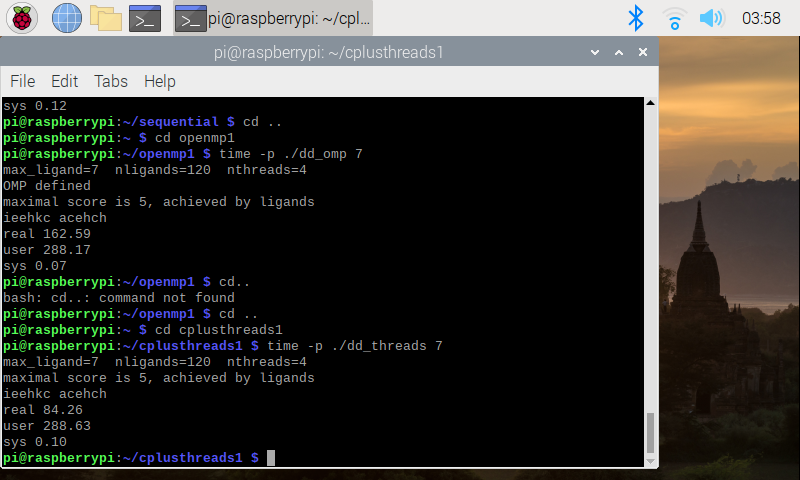
**As expected when running a sequential implementation with threads you’d receive an odd result**

****

|  |  |
| --- | --- |
| **Implementation** | **Time(s) 5 Threads** |
| **dd\_serial** | **0.01** |
| **dd\_omp** | **19.21** |
| **dd\_threads** | **14.82** |

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

**Last increased the max ligand to 7 and recorded each implementation’s results:**

****

|  |  |
| --- | --- |
| **Implementation** | **Time(s)** |
| **dd\_serial** | **287.58** |
| **dd\_omp** | **162.59** |
| **dd\_threads** | **84.26** |