|  |  |
| --- | --- |
| Developing Soft and Parallel Programming Skills Using Project-Based Learning, | (Fall 2019) ***THE 6-PACK:***  Team MEMBERS:  *jENNIFER VU*  *Abraham mammen*  *rachid bodson*  *bryan rudy gonzales*  *hazel santiago* |

**Planning and Scheduling (T-1):**

**Work Breakdown Structure**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Assignee**  **Name** | **Email** | **Task** | **Duration**  **(Hours)** | **Dependency** | **Due**  **date** | **Note** |
| Abraham Mammen  **(coordinator)** | amammen1@student.gsu.edu | -Create the work breakdown structure  -Proofread and submit the report electronically  -Parallel Programming participation | -1  -1  -3 | Reading and understanding the Parallel Programming Basics instructions | -12/2/2019  -12/6/2019  -11/21/2019 |  |
| Jennifer Chau Vu | jvu@student.gsu.edu | -Parallel programing lab report  -Parallel Programming participation | -3  -3 | -Reading and understanding the Parallel Programming Basics instructions | -12/2/2019  -11/21/2019 |  |
| Hazel Alexis Santiago | hsantiago1@student.gsu.edu | -Parallel Programming skills foundation  -Parallel Programming participation | -2  -3 | Reading and understanding the Parallel Programming Basics instructions | -11/3/2019  -11/21/2019 |  |
| Rachid Bodson | rbodson1@student.gsu.edu | -Parallel Programming skills foundation  -Parallel Programming participation | -2  -3 | Reading and understanding the Parallel Programming Basics instructions | -11/3/2019  -11/21/2019 |  |
| Lauren Taylor James | ljames26@student.gsu.edu | -Project report  -Parallel Programming participation  -Slack and github | -3  -2  -0.5 | - Screenshots from Slack & GitHub, and Lab Reports needed to complete report  - Reading and understanding the Parallel Programming Basics instructions | -12/4/2019  -12/3/2019  -12/3/2019 | Report must be ready 24 hours before due date. |
| Bryan Rudy Gonzales | bgonzales1@student.gsu.edu | -Video  -Parallel Programming participation | -5 | -Reading and understanding the Parallel Programming Basics instructions  -make sure that everyone ones available to meet for shooting | -12/3/2019 | Video should be ready 24 hours before due date |

**Parallel Programming Skills(T-3):**

**Foundation**

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

The ideal parallel programming computation task is one that can be decomposed into equal-size partitions, same processing is required for each element, no dependencies in computation, and no communication required between tasks. Basically, the steps are: Decomposition (identify tasks) → Assignment (balancing load and maximizing locality) → Orchestration (name and data are accessed, and synchronization is performed among processes) → Mapping (processes are assigned to processors). An example would be to have an array with equal-size subarrays. A technique would be the master/worker implementation. The master initializes the array and splits it up according to the number of available workers. It then sends each worker its subarray (of equal-size). The worker receives the subarray from the master. It then performs the processing on the subarray and returns the result to the master. The model uses the static load balancing implementation which is the same amount of work on identical machines being performed by all the parallel tasks. The load balancing refers to spreading of tasks among processors in a parallel system to avoid processors from being idle while other tasks are queued up for execution. A static load balancer allocates processes to processors at run time while taking no account of current network load.

1. **What is MapReduce?**

MapReduce is a mechanism developed within Google to process large amounts of raw data and distribute said data across thousands of machines, implying parallel computing. MapReduce also derives from the map and reduce combinators from a functional language such as Lisp.

1. **What is map and what is reduce?**

Map function takes an input pair and produces a set of intermediate key/value pair. The MapReduce library groups the together all intermediate values associated with the same intermediate key *I* and passes them to reduce function. Map invocations, in MapReduce model execution, are distributed across multiple machines. The input data is automatically partitioned into a set of M splits or shards. The shards are processed in parallel on different machines.

Reduce function accepts an intermediate key *I* and a set of values for that key. It merges together these values to form a possibly smaller set of values. Reduce invocations are distributed by the intermediate ley space into R pieces using a partition function (hash(key)). The number of partitions (R) and the partition function are specified by the user.

Both are written by a user of the MapReduce library.

1. **Why MapReduce?**

MapReduce is very cost-effective when implementing it to traditional databases when it comes to performing computations and storing data. It is much faster and more efficient.

1. **Show an example for MapReduce.**

An example of MapReduce is implementing it with Hadoop to operate on large amount of data by creating in the format of tuples to make things simple.

1. **Explain in your own words how MapReduce model is executed.**

In a MapReduce model execution, the MapReduce library in the user program first shards the input files in M pieces of 16-64 MB per piece. The user program is copied into clusters of machines and are initiated. One of the copies is the master; the rest are workers that are assigned tasks by the master. There are two types of tasks: M map and R reduce. The master identifies idle workers and assigns each either M or R tasks. The workers with M tasks read the contents of its corresponding input shard. It passes key pairs out of the input data and into a user-defined Map function. The intermediates key pairs produced in this Map function are buffered in memory. Periodically, the pairs are written on local disk, partitioned into reduce regions by the partitioning function. The locations of the pair on the local disk are passed onto the master. The master then assigns the locations to the reduce workers. The reduce workers use remote procedure calls to read the buffered pairs from the local disk of the map workers. Once all the data is read, the workers sort the data by intermediate key, therefore, all the same keys are clustered together. For each unique intermediate key, the reduce workers pass the key and its corresponding set of values to the user’s Reduce function. The output of the Reduce function is attached to a final output file for the reduction partition. Once the map and reduce workers complete their respective tasks, the master wakes up the user program. The MapReduce call is then returned back into user code.

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

Distributed Grep: The map function emits a line if it matches a given pattern. The reduce function is an identity function that copies the supplied intermediate data to the output.

Reverser Web-Link Graph: The map function outputs <target, source> pairs for each link to a target URL found in a page named “source.” The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>.

Inverted Index: The map function analyzes each document, and emits a sequence of <word, document ID> pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair. The set of all output pairs forms a simple inverted index. This computation can be augmented to keep track of word positions.

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

We use OpenMP when writing a multithreaded program because the compiler takes care of transforming the sequential code into parallel code according to the directives and also the threads will share the same memory address space.

We use MPI when we are trying to develop parallel scientific applications because it is a high-level abstraction for parallel programming.

We use MapReduce when we are dealing with big data processing because MapReduce handles data record by record without loading whole data into memory and in addition the program is executed in parallel over a cluster.

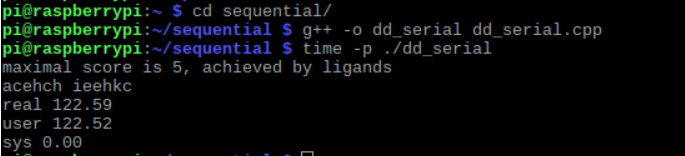
1. **In your own words, explain what a Drug Design and DNA problem is in no more than 150words.**

Drug design is the process of finding and testing new treatments based on biological targets. For the efficiency of the drug design, we use computer modeling techniques known as the Computational drug design. The Computational Drug Design is the designing of drug using computational capabilities such to make sure the effectiveness of the drug remains intact and can able to fight against diseases. Once a drug has been shown to be effective by an initial computer modeling technique, much more testing must be done before it can be given to human patients.DNA problem is referred to as the problems that are inherent to the DNA computers while performing operations related to DNA.

**Parallel Programming Basics: Drug Design and DNA in Parallel**

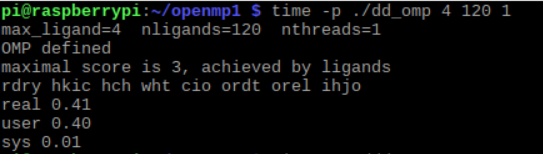
**Sequential Solution:**

We downloaded dd\_serial.cpp and Makefile and moved them to the sequential folder. We compiled the file using the command, **g++ -o dd\_serial dd\_serial.cpp**. We ran the program and measured the run time using the command, **time -p ./dd\_serial**.



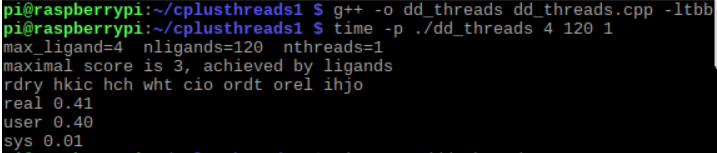
**OpenMP Solution:**

We downloaded dd\_omp.cpp and Makefile and moved them to the openmp1 folder. We compiled the file using the command, **g++ -o dd\_omp dd\_omp.cpp -ltbb -lpthread -fopenmp.** We ran and measured the run time using the command, **time -p ./dd\_omp 4 120 1.**



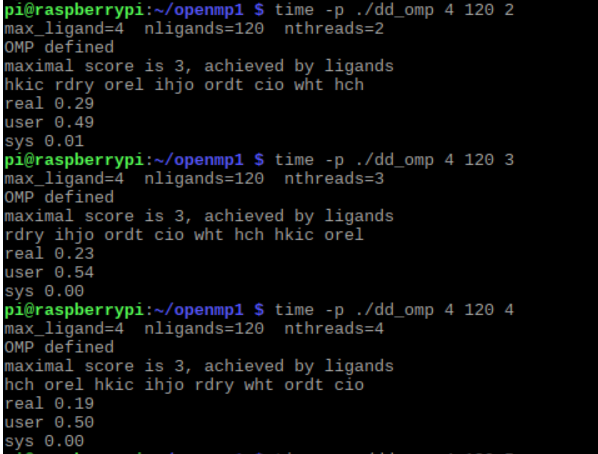
**C++ 11 Threads Solution:**

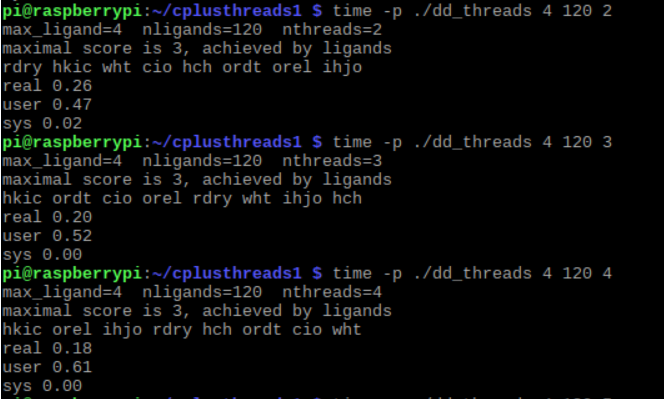
We downloaded dd\_threads.cpp and Makefile and moved them to the cplusthreads1 folder. We compiled the file using the command, **g++ -o dd\_threads dd\_threads.cpp -ltbb -lpthread. “-ltbb”** makes sure the tbb library is included since a header file from the tbb library is imported in dd\_threads.cpp. We ran the program and measured the run time using the command, **time -p ./dd\_threads 4 120 1.**



|  |  |
| --- | --- |
| Implementation | Time (s) |
| dd\_serial | 122.59 |
| dd\_omp | 0.41 |
| dd\_threads | 0.41 |

We ran the program and measured the run times using the commands, “**time -p ./dd\_omp 4 120 2”,** “**time -p ./dd\_omp 4 120 3”,** and“**time -p ./dd\_omp 4 120 4”.**



We ran the program and measured the run times using the commands, “**time -p ./dd\_threads 4 120 2”,** “**time -p ./dd\_threads 4 120 3”,** and“**time -p ./dd\_threads 4 120 4”.**

|  |  |  |  |
| --- | --- | --- | --- |
| Implementation | Time (s) 2 Threads | Time (s) 3 Threads | Time (s) 4 Threads |
| dd\_omp | 0.29 | 0.23 | 0.19 |
| dd\_threads | 0.26 | 0.20 | 0.18 |

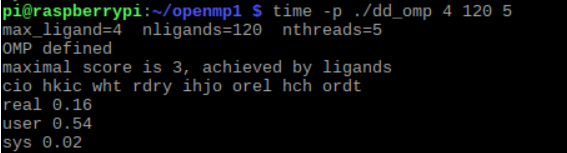
**Discussion Questions:**

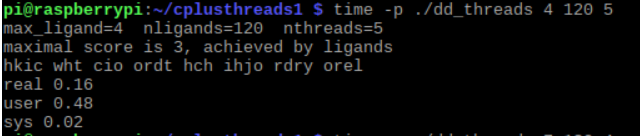
**1. Which approach is the fastest?** The C++ 11 threads solution is the fastest.

**2.** **Determine the number of lines in each file (use wc -l). How does the C++11 implementation compare to the OpenMP implementation?** The serial solution has 171 lines. The C++ 11 solution has 208 lines. The OpenMP solution has 194 lines. The C++ 11 implementation has more lines compared to the OpenMP implementation.

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

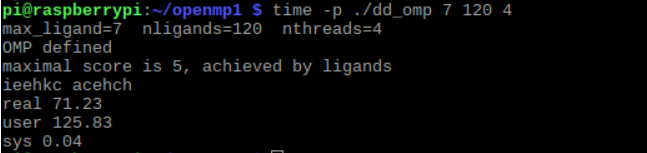
The run time for the omp solution is 0.16 s, and the run time for the C++ 11 solution is 0.16 s.

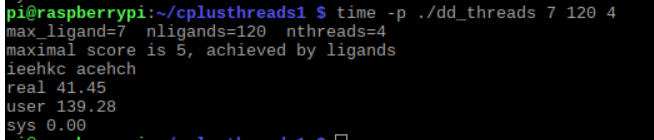




**4. Increase the maximum ligand length to 7, and rerun each program. What is the run time for each.**

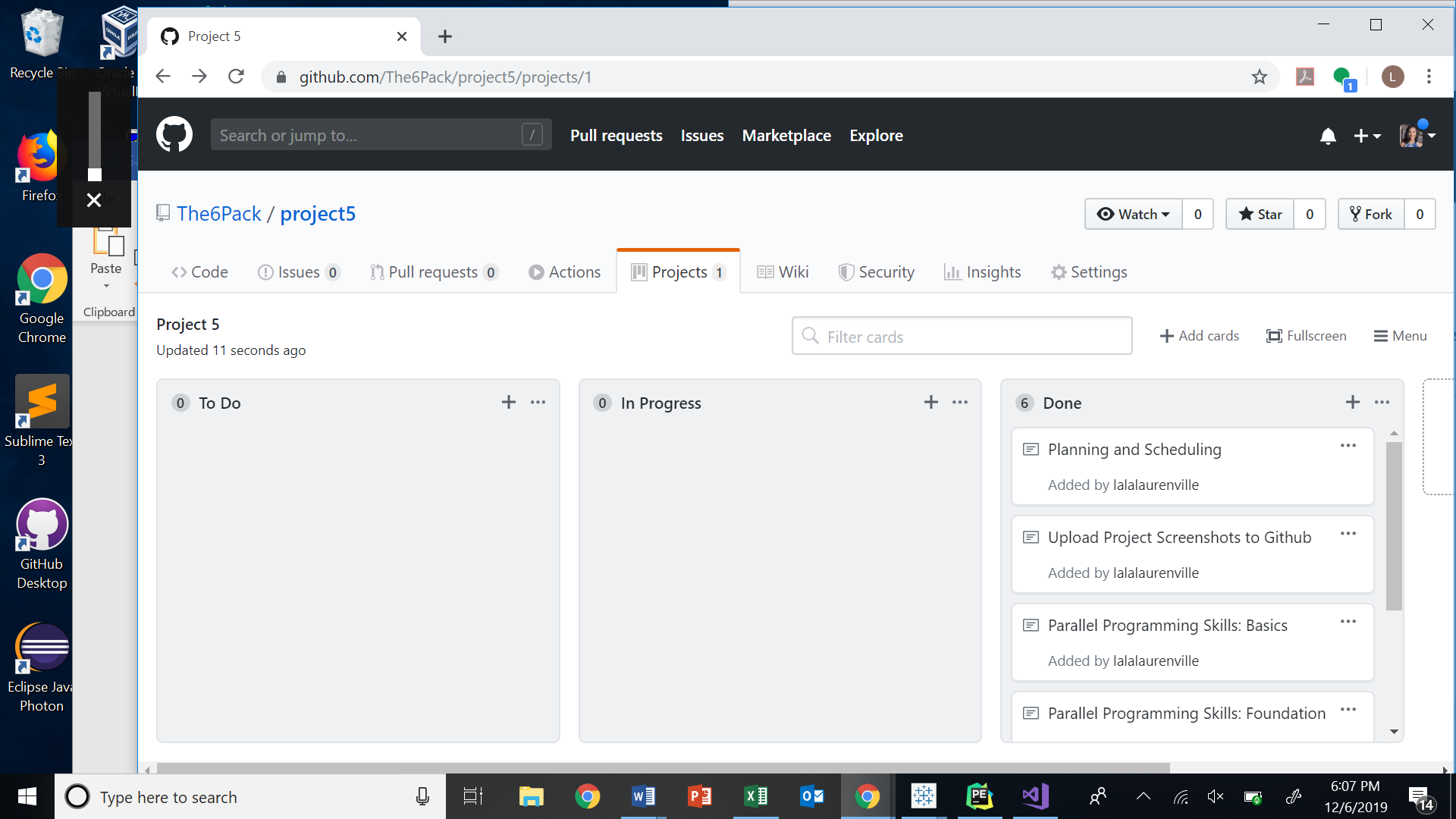
The run time for the omp solution is 71.23 s, and the run time for the C++ 11 solution is 41.45 s.



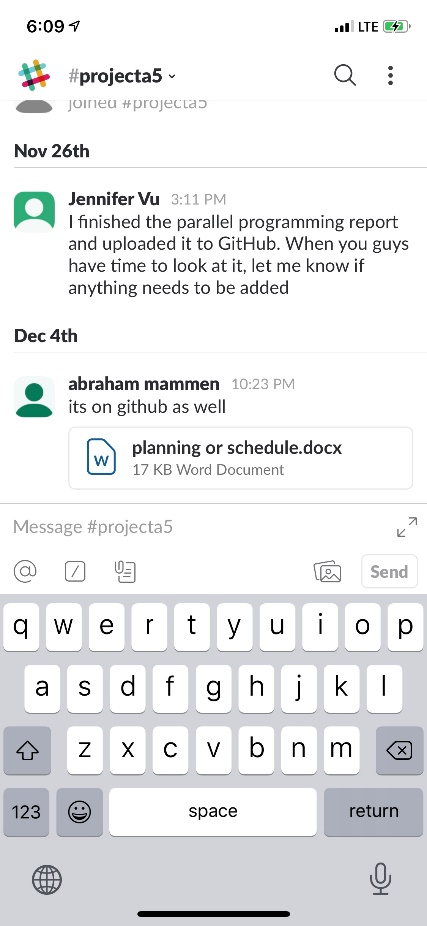


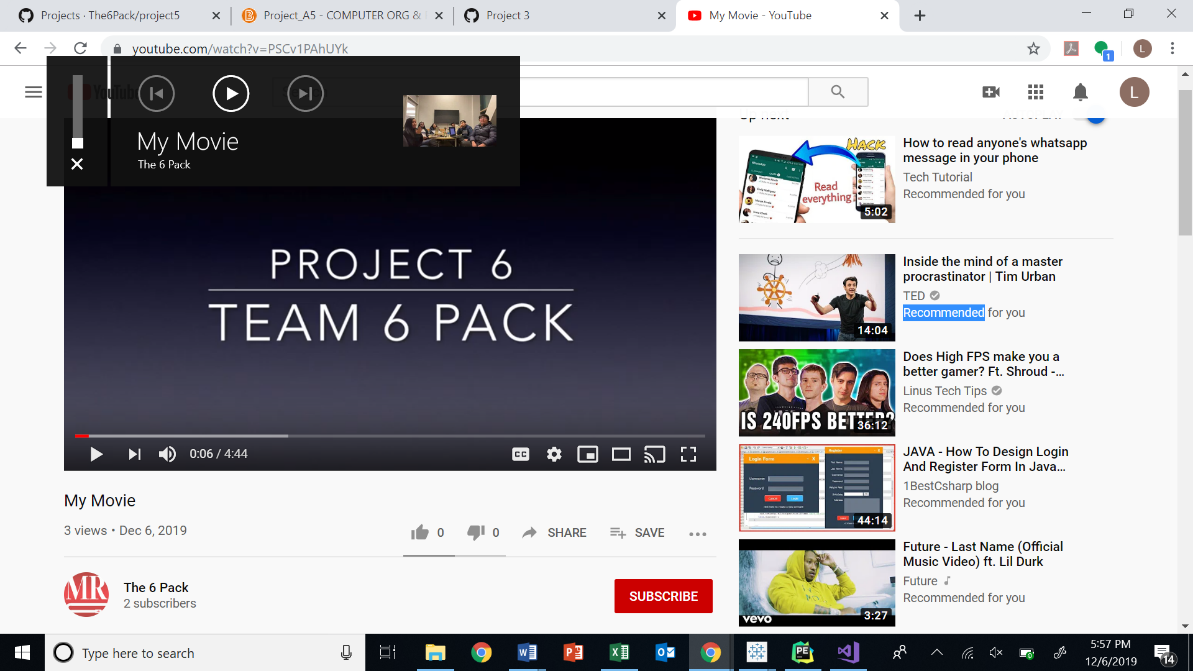
**Appendix:**

***GitHub-***https://github.com/The6Pack/project5



***Slack-*** the6pack2019.slack.com (Channel: #projecta5)



***Youtube***-

https://www.youtube.com/channel/UC7dMmLnGrmjZv3d8c-39qJg?view\_as=subscriber