Developing Soft and Parallel Programming Skills Using Project Based Learning

Fall 2019

Group Name: Quinary

Members: Dominick DiLeo, Meet Patel, Chowdhary Mobin, Akiva Ochova, Ivana Lanzani

Assignment Schedule

Asignee Name	Email	Task	Duration	Dependency	Due date	Note
Meet Patel	mpatel165@	Final report	2 hours	Everyone else being done	Friday December 6th	Complete on time
Dominick DiLeo	ddileo2@	Planning and coordinating. Dealing with github and schedule	1 hour	Hearing what everyone wants to do	Thursday December 5th	Complete on time
Chowdhury Mobin	cmobin1@	Questions for parallel part	2 hours	Reading the file given for the questions	Thursday December 5th	Complete on time
Akiva Ochoa	aochoa5@	Parallel programming	3 hours	Understanding the previous programming tasks	Thursday December 5th	Complete on time
Ivana Lanzani	ilanzani1@	Resetting the pi Uploading video	1 hour	The programming task being done	Friday December 6th	Complete on time

Introduction to Parallel Programming and MapReduce

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

   - Parallel programming had been developed to improve performance and efficiency. First, we must identify sets of tasks that can run concurrently or partitions of data which can be processed concurrently on a parallel program. A common implementation technique is master/worker. Master/Worker uses static load balancing. Which divides tasks among the processors in a parallel system to avoid processors being idle to increase efficiency and performance.

2. What is MapReduce?

   - MapReduce is a programming model. The map and reduce combinators from Lisp, a functional language originates this model.

3. What is map and what is reduce?

   - In Lips, map takes a function as an input and a sequence of values. After this, it applies the function to each value in the sequence. Map which is written by a user of the MapReduce library takes input pairs and produces a set of intermediate key values. Then, all the intermediate values associated with intermediate keys are grouped together by the MapReduce library passes them to reduce function.

     Reduce combines all the elements od the sequence taken by map using a binary operation. This is also written by the user which accepts an intermediate key and a set of values for that key. It forms a conceivably smaller set of values by merging those values together.

4. Why MapReduce?

   - It is an abstraction which allows to write scalable applications which uses parallel processing to process a huge amount of data. It was introduced by Google, so that their engineers could perform simple computation while hiding details of parallelization, load balancing, data distribution and fault tolerance. It processes a large amount of raw data and then distributes it across thousands of machines to be processed in a reasonable time. After Google, Yahoo, Netflix, Facebook and many more companies starting using MapReduce.

5. Show an example for MapReduce.

   - Source Code:

     ```
     map (String key, String value):
     for each word w in value:
     EmitIntermediate (w, "1");
     ```

```
Reduce (String key, Iterator values):
int result = 0;
for each v in values:
result += ParseInt (v);
Emit (AsString (result));
```

Here, key is document name and value are document contents, under map function. It emits each word plus an associated count of occurrence using the function map. After that, the reduce function sums together all that for a specific word.

6. Explain in your own words how MapReduce model is executed?

   - MapReduce is a programming model which is used to process huge amount of data. The programs are parallel in nature. The model has two phase first one is Map and the second one is Reduce. First, phase of this model splits the input data in sets. The input data after the split can be processed on different machines on parallel. On the second phase, the partitioning of the intermediate key space happens. Here, the number of partitions and the functions of partitions are specified by the user.

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

   - Three examples which are expressed as MapReduce computations are Distributed Grep, Reverse Web-Link Graph and Inverted Index.
     Distributed Grep: This map function emits a line if it matches a given pattern. The reduce function then copies the supplied intermediate data to output.
     Reverse Web-Link Graph: Here map function outputs pairs of target and source links to a specific URL found in a page named source. After that the reduce function takes all source URL's associated with the given target URL and emits the pairs.
     Inverted Index: Word and document ID pairs are emitted by the map function. The reduce function then accepts all the pairs of given words and sorts the document ID's and emits the pairs. All output pair forms inverted index.

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

   - In a code, OpenMP is used to introduce shared memory parallelism. It is an effective directive-based library which is powerful yet simple. It is used mostly to get the best performance. But it has its limitations, sometimes it causes performance issues.

     Message Passing Interface or MPI is used to develop scientific applications. It uses a distributed memory parallel model implementation. The codes are usually load balanced and highly synchronous. It can be used in codes which runs over multiple machines. User can also combine MPI and OpenMP to do hybrid programming.

Hadoop MapReduce gives user two constructs to apply over large amount of data. This is used when data is too large to organize. User can do some reduction over the results also by using this. If users have terabytes of data to extract, transform and load they should use Hadoop MapReduce. Users can also use this for scientific applications, but it will not be as good as MPI.

9. In your own words, explain what a Drug Design and DNA problem is.

   - Our DNA contains the instructions for making proteins in our bodies. Different proteins perform different tasks in a body. To identify which protein does which task we need to see the protein shape. Small molecules called ligands found in protein is used to design drugs. Some ligands will fit, and some will not. The one which fit will be used to produce a desired shape change to design a drug. A collection of ligands is used and tested against the protein to check if they are binding in a useful way. They are scored on binding properties and the best are marked and identified which will make good drug candidates.

     Here, MapReduce strategy is used to solve the problem by implementing a master worker pattern. First, we use a function which many ligands will be tested against the given protein. Then Map function will be used to compute binding score for pairs. Then Reduce function will identify the highest scoring ligands.

Parallel Programming Coding Part / Report

For this programming assignment I had to observe the differences between the efficiency of sequential, OpenMP, and C++11 implementations for theoretical medicine creation.

To start, I downloaded the given programs for comparing ligand strings. Later extracting the files for each implementation. The first task required running the default cases for each implementation. So, for the first case I entered the sequential directory to compile the file using make. After an executable was created I ran the program using time -p ./dd_serial in order to record the time it took for the program to complete. Resulting in 128 seconds for it to complete.

```
pi@raspberrypi:~/assignment5/sequential $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 128.69
user 128.51
sys 0.00
```

To finish the table for default times I repeated the process of compiling the respective files using make and printing the time each took.

OpenMP:

```
pi@raspberrypi:~/assignment5/openMP $ time -p ./dd_omp
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 80.11
user 141.53
sys 0.11
```

And threads:

```
pi@raspberrypi:~/assignment5/threads $ time -p ./dd_threads
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 40.72
user 138.07
sys 0.03
```

The next task was to record the times for openMP and threads using different thread counts. To do this had to append the specific numerical arguments I wanted for the program's ligand count, max length , and threads. I noticed that with less threads each took more time to complete, yet the C++11 implementation still ran faster than the openMP version:

openMP using 2 threads:

```
pi@raspberrypi:~/assignment5/openMP $ time -p ./dd_omp 7 120 2
max_ligand=7  nligands=120  nthreads=2
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 111.71
user 138.89
sys 0.01
```

threads using a thread count of 2:

```
pi@raspberrypi:~/assignment5/threads $ time -p ./dd_threads 7 120 2
max_ligand=7  nligands=120  nthreads=2
maximal score is 5, achieved by ligands
ieehkc acehch
real 69.92
user 127.01
sys 0.02
```

From the discussion questions, I've observed C++11 was able to complete the same tasks as openMP swifter, with only a few more lines code used. With this upon observing a larger case with 9 ligands, the difference between them was starker with C++11 completing 10 minutes ahead.

Initial Data Collection

Base Time Comparison:

Implementation Used	Time
dd_serial	128.69
dd_OpenMP	80.11
dd_threads	40.72

Thread Count Comparison:

	Time(s) 2 Threads	Time(s) 3 Threads	Time(s) 4 Threads
dd_OpenMP	111.71	96.23	69.33
dd_threads	69.92	53.46	41.87

Discussion Question

1. The thread C++11 approach resulted in faster completion.

2. dd_OpenMP = 193 lines | dd_threads = 207 lines. C++11 had only 14 more lines of code than OpenMP.

3. 5 Threads Test:

Implementation Used	Time(s) 5 Threads
dd_OpenMP	64.21
dd_threads	37.44

4. 9 Ligand Test

Implementation Used	Time(s) 9 Ligands
dd_OpenMP	2672.16
dd_threads	2075.03

Appendix

GitHub:

https://github.com/Quinary-GSU/Assignment1

Slack:

https://app.slack.com/client/TN3HREW9X/GNRJ40RT9

Parallel Coding Part 1:

https://github.com/Quinary-GSU/Assignments-1-to-4/blob/master/Assignment5_openMP_Thread_Test.png

Parallel Coding Part 2:

https://github.com/Quinary-GSU/Assignments-1-to-4/blob/master/Assignment5_threads_Thread_test.png

Parallel Coding Part 3:

https://github.com/Quinary-GSU/Assignments-1-to-4/blob/master/Assignment5_Serial.png

Parallel Coding Part 4:

https://github.com/Quinary-GSU/Assignments-1-to-4/blob/master/Assignemnt5_5_Threads.png

Parallel Coding Part 5:

https://github.com/Quinary-GSU/Assignments/blob/master/Assignment5_7_Ligand.png

YouTube:

https://www.youtube.com/watch?v=3ciUfuxnch8&feature=youtu.be