Matthew Maloof
Project 5 Task 3 - Parallel Programming Skills

Foundation:

- (56p) Read this paper "Introduction to Parallel Programming and MapReduce" and answer the following questions:

---

o (15p) What are the basic steps (show all steps) in building a parallel program? Show at least one example.

The basic steps for building a parallel program involves

1. Decomposition (Breaking up computation into tasks to be divided among different processes)
2. Assignment (Specifying mechanism to divide work up among different processes)
3. Orchestration (Involves naming data, structuring communication, synchronization, and organizing data structures / scheduling tasks temporally)
4. Mapping (Parallel program is made after organization, mapping involves two questions:
   -Which processes will run on the same processor if necessary?
   -Which process runs on which processor?
   or mapping to a network topology)

An example of parallel processing involves scientific simulation software.

o (5p) What is MapReduce?

MapReduce is a framework that is suitable for processing huge amounts of data (terabytes and petabytes). This is done by generating big data sets with a parallel, distributed algorithm on a cluster.

o (10p) What is map and what is reduce?

Map is used to read a stream of data and parse it into intermediate pairs / another set of data. Reduce takes the output from the map and combines these individual elements into smaller tuples (key/value) pairs.

o (5p) Why MapReduce?

MapReduce is used because it's a very cost-effective process when implemented into the traditional databases when it comes to performing computations and storing data. MapReduce is more efficient than OpenMP

o (5p) Show an example for MapReduce.

MapReduce is used with Hadoop to operate on the large amount of data being used. This is done by creating tuples to make things simpler. An example would be a search tool in Microsoft

Word to find a specific character or sequence and display the count in that document, with map counting the appearances and reduce being the calculator expressing the total sum.

o (10p) Explain in your own words how MapReduce model is executed?

MapReduce model is executed in three stages - Map Stage, Shuffle Stage, and Reduce Stage. Map Stage is the mapper's job to process the input data. This data is usually in the form of a file and is stored in the HDFS (Hadoop File System). This input file is passed to the mapper function line by line. This mapper processes this data and creates several smaller simpler chunks of data.

The Reduce Stage combines the Shuffle and Reduce stages together. The reducer's job processes the data that is outputted from the mapper, then it produces a new set of output stored in the HDFS.

After these stages, Hadoop sends the Map and Reduce tasks out to the appropriate servers in the specific cluster. This framework manages all the details regarding data-passing, where most of the computing occurs on nodes with data on local discs. This allows for reduced network traffic. After completion of the above, this cluster collects and reduces the data and forms an appropriate result, which is when sent back to the Hadoop server.

o (6p) List and describe three examples that are expressed as MapReduce computations.

- Reverse Web-Link Graph: This uses MapReduce computations by producing the targets and sources for each link and turns them into a target URL found in a source page. Then the Reduce function puts all the source URLs together to create the target and source pairs.
- URL Access Frequency Number: This also uses Mapreduce computations by using the map function to process web requests to provide the URLs and counts when the reduce function adds these values of the same URLs and yields the total count.
- Distributed grep: Lastly, this uses the map function to find patterns in a given input line and the reduce function copies that similar data/pattern found to the output.

- (6p) When do we use OpenMP, MPI and, MapReduce (Hadoop), and why?

OpenMP is used to introduce shared memory parallelism to code. This is because OpenMP is an efficient directive based library where you can just put a directive like '#pragma parallel for' before the for loop to split it into multiple threads.

MPI is used to develop parallel scientific applications and develop any parallel code that runs over multiple machines. The reason it's good for scientific applications is because these applications are usually tightly synchronous code and well load balanced. Mixing MPI and OpenMP to use threads within machines is called hybrid programming.

MapReduce (Hadoop) is used over MPI when you have terabytes of data that you want to use EPL (extract, transform, and load) like operations. An important feature Hadoop has is fault tolerance. Performance-wise, it's not a good idea to write tightly coupled scientific applications with MapReduce. It is a great idea to use MapReduce to scale data processing over multiple computing nodes.

- (14p) In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.

Drug design is a very lengthy and complex process that has a high risk of uncertainty towards if the drug will work or not. These issues also play into account with unknown pathophysiology in different nervous system disorders which make identifying the target difficult. There is no certain way to retrieve validated diagnostic and therapeutic biomarkers which would help to detect and measure the biological states. DNA problem is used where problems exist towards the DNA computers performing the operations related to DNA effectively and efficiently. Creating this drug design software would have to generate the ligands in a trial and error fashion to fit into a protein, and assign a score as to how well it fits. Then, it will sort the scores and pick the highest scoring ligands to use/test.

Part B: Programming

I downloaded all the files and placed them in their respective folders then made each serial openMP and threads executables as shown in Figure 1.

```
pi@raspberrypi:~/Project/serial $ g++ -o dd_serial dd_serial.cpp
pi@raspberrypi:~/Project/serial $ cd ~/Project/openMP
pi@raspberrypi:~/Project/openMP $ g++ -o dd_omp dd_omp.cpp -lm -fopenmp -ltbb -lrt
pi@raspberrypi:~/Project/openMP $ cd ~/Project/threads
pi@raspberrypi:~/Project/threads $ g++ -o dd_threads dd_threads.cpp -lm -std=c++11 -pthread -ltbb -lrt
pi@raspberrypi:~/Project/threads $ make
make: 'dd_threads' is up to date.
pi@raspberrypi:~/Project/threads $ cd ~/Project/openMP
pi@raspberrypi:~/Project/openMP $ make
make: 'dd_omp' is up to date.
pi@raspberrypi:~/Project/openMP $ cd ..
pi@raspberrypi:~/Project $ cd serial
pi@raspberrypi:~/Project/serial $ make
make: 'dd_serial' is up to date.
pi@raspberrypi:~/Project/serial $
```

Figure 1

Then, I ran each implementation to compare the real time it takes to see the efficiency (Figure 2).

```
pi@raspberrypi:~/Project/serial $ time -p ./dd_serial
maximal score is 5, achieved by ligands
acehch ieehkc
real 143.84
user 143.83
sys 0.00
```

```
pi@raspberrypi:~/Project/openMP $ time -p ./dd_omp
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 81.20
user 143.87
sys 0.03
pi@raspberrypi:~/Project/openMP $ cd ~/Project/threads
pi@raspberrypi:~/Project/threads $ time -p ./dd_threads
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 42.31
user 143.85
sys 0.03
```

Figure 2

Then I ran dd_omp and dd_threads with 1 max ligand as requested (Figure 3).

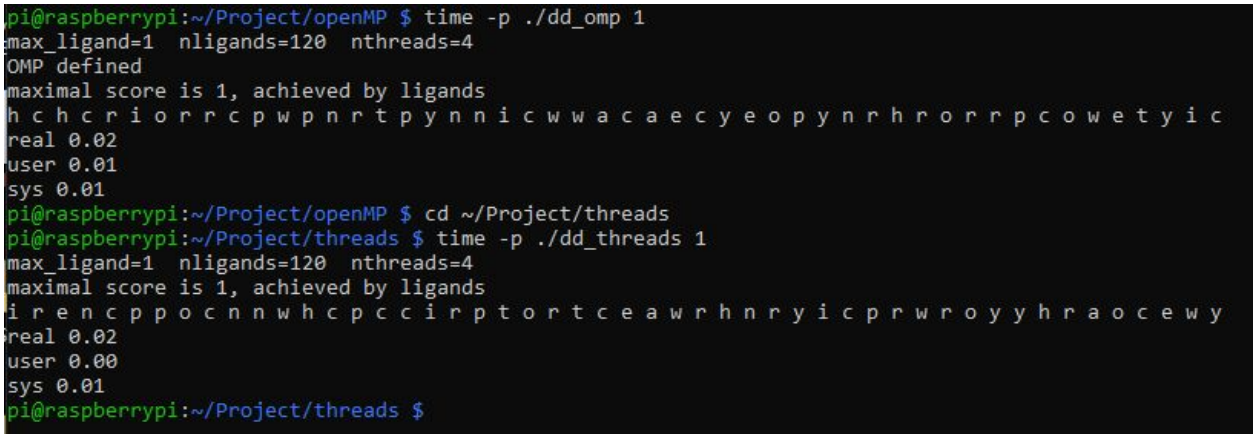

Figure 3

Implementation	Time(s)
dd_serial	143.84
dd_omp	81.20
dd_threads	42.31
dd_omp 1 max thread	0.02
dd_threads 1 max thread	0.02

Then, I ran dd_omp and dd_threads with 2, 3, and 4 max threads and reported the results (Figures 4 and 5).

```
pi@raspberrypi:~/Project/openMP $ time -p ./dd_omp 7, 120, 2
max_ligand=7  nligands=120  nthreads=2
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 116.71
user 144.26
sys 0.12
pi@raspberrypi:~/Project/openMP $ time -p ./dd_omp 7, 120, 3
max_ligand=7  nligands=120  nthreads=3
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 98.62
user 144.01
sys 0.02
pi@raspberrypi:~/Project/openMP $ time -p ./dd_omp 7, 120, 4
max_ligand=7  nligands=120  nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 80.99
user 143.98
sys 0.01
```

Figure 4

```
pi@raspberrypi:~/Project $ cd threads
pi@raspberrypi:~/Project/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 79.65
user 144.03
sys 0.00
pi@raspberrypi:~/Project/threads $ time -p ./dd_threads 7, 120, 3
max_ligand=7  nligands=120  nthreads=3
maximal score is 5, achieved by ligands
acehch ieehkc
real 55.22
user 143.46
sys 0.02
pi@raspberrypi:~/Project/threads $ time -p ./dd_threads 7, 120, 4
max_ligand=7  nligands=120  nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 42.11
user 143.86
sys 0.00
pi@raspberrypi:~/Project/threads $
```

Figure 5

Implementation	Time (s) 2 Threads	Time (s) 3 Threads	Time (s) 4 Threads
dd_omp	116.71	98.62	80.99
dd_threads	79.65	55.22	42.11

Questions :

1. The approach that is the fastest is dd_threads which uses C++11 threads to solve the problem over OpenMP threads. The larger max threads input, the faster it performs.
2. Typing wc -l to measure the lines of code for each program yields Serial being 170 lines, openMP being 193 lines, and Threads being 207 lines (Figure 6).

```
pi@raspberrypi:~/Project/threads $ cd ~/Project/serial
pi@raspberrypi:~/Project/serial $ wc -l dd_serial.cpp
170 dd_serial.cpp
pi@raspberrypi:~/Project/serial $ cd ~/Project/openMP
pi@raspberrypi:~/Project/openMP $ wc -l dd_omp.cpp
193 dd_omp.cpp
pi@raspberrypi:~/Project/openMP $ cd ~/Project/threads
pi@raspberrypi:~/Project/threads $ wc -l dd_threads.cpp
207 dd_threads.cpp
pi@raspberrypi:~/Project/threads $
```

Figure 6

3. I increased the number of threads to 5 and the run time is shown in Figure 7.

```
pi@raspberrypi:~/Project/openMP $ time -p ./dd_omp 7, 120, 5
max_ligand=7  nligands=120  nthreads=5
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 74.66
user 143.69
sys 0.00
pi@raspberrypi:~/Project/openMP $ cd ..
pi@raspberrypi:~/Project $ cd threads
pi@raspberrypi:~/Project/threads $ time -p ./dd_threads 7, 120, 5
max_ligand=7  nligands=120  nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 44.45
user 150.37
sys 0.06
pi@raspberrypi:~/Project/threads $
```

Figure 7

Implementations	Time (s) 5 Threads
dd_omp	74.66
dd_threads	44.45

4. The programs have been run already with a max ligand length of 7 (default value) (refer to figure 2).

Implementations	Time (s) 5 Threads
dd_serial (max_ligand = 7)	143.84
dd_omp (max_ligand = 7)	81.20
dd_threads (max_ligand = 7)	42.31