6CS005 Learning Journal - Semester 1 2019/20

Birat Jung Thapa, 2040367

Table of Contents

1 Parallel and Distributed Systems

1.1 Answer of First Question

A thread is part of the process, operating within its own space of execution, and in one process there can be several threads. With the help of it, OS can do multiple tasks in parallel (depending upon the number of processors the machine consists). Threads allow CPU to execute many tasks of one process at the same time.

1.2 Answer of Second Question

Two process scheduling policies are:

Pre-emptive: This scheduler oversees how long a process runs for. If a process exceeds its time slice, the scheduler stops the process.

Co-operative: Each process is in-charge of how long it runs for. When a process feels like co-operating, it will surrender execution.

Pre-emptive is preferable, Java runtime system's thread scheduling algorithm is also pre-emptive.

1.3 Answer of Third Question

In centralized system, all calculations are done on one computer(system). In distributed system, the calculation is distributed to multiple computers. For example: When there is a large amount of data then it can be divided and sent to each part of computers to carry it out.

1.4 Answer of Fourth Question

Transparency in Distributed system means one distributed system looks like a single computer by concealing distribution from the user and application programmer.

1.5 Answer of Fifth Question

Include your code using a text file in the submitted zipped file under name Task1.5

B=A+C is a flow dependency
C=B+D is an anti-dependency
B=C+D is an output dependency

1.6 Answer of Sixth Question

Program 1 answer: 349151

Program 2 answer: 500000

Program 1 and 2 both use thread function to perform thread count for the given unassigned integer [N]. Program 1 tuns an extra loop.

2 Applications of Matrix Multiplication and Password Cracking using HPC-based CPU system

2.1 Single Thread Matrix Multiplication

- The analysis of the algorithm's complexity. (1 mark)

  The complexity of above program is O(n^3).
- Suggest at least three different ways to speed up the matrix multiplication algorithm given here. (Pay special attention to the utilisation of cache memory to achieve the intended speed up). (1 marks)

  Divide and Conquer could also speed the multiplication process but the better option would be Strassen's matrix multiplication.
- Write your improved algorithms as pseudo-codes using any editor. Also, provide reasoning as to why you think the suggested algorithm is an improvement over the given algorithm. (1 marks)

```
begin
  If n = threshold then compute
            C = a * b is a conventional matrix.
  Else
            Partition a into four sub matrices a11, a12, a21, a22.
            Partition b into four sub matrices b11, b12, b21, b22.
            Strassen (n/2, a11 + a22, b11 + b22, d1)
            Strassen (n/2, a21 + a22, b11, d2)
            Strassen (n/2, a11, b12 – b22, d3)
            Strassen (n/2, a22, b21 – b11, d4)
            Strassen (n/2, a11 + a12, b22, d5)
            Strassen (n/2, a21 – a11, b11 + b22, d6)
            Strassen (n/2, a12 – a22, b21 + b22, d7)

            C = d1+d4-d5+d7     d3+d5
            d2+d4           d1+d3-d2-d6

  end if
  return (C)
end.
```

Since the above program does 8 multiplications for matrices of size N/2 * N/2 and 4 additions, Strassen's multiplication performs it in 7 multiplications, so it is faster.

- Write a C program that implements matrix multiplication using both the loop as given above and the improved versions that you have written. (1marks)

  Include your code using a text file in the submitted zipped file under name Task2.1

- Measure the timing performance of these implemented algorithms. Record your observations. (Remember to use large values of N, M and P – the matrix dimensions when doing this task). (1 marks)

  The improved program performs the matrix multiplication in 18.411 seconds.

  Insert a paragraph that hypothesises how long it would take to run the original and improved algorithms. Include your calculations.

  Explain your results of running time.

  The time complexity of original program is $O(n^3)$ and the improved program is $O(n^{2.80})$. So, the time taken by the improved version is less than the original version.

2.2 Multithreaded Matrix Multiplication

- Include your code using a text file in the submitted zipped file under name Task2.2
- Insert a table that has columns containing running times for the original program and your multithread version. Mean running times should be included at the bottom of the columns.
- Insert an explanation of the results presented in the above table.

Number of programs runs	Original Program	Multithread version
1	18.411	3.0
2	18.400	3.0
3	18.381	3.0
4	18.411	3.0
5	18.298	3.0
6	18.402	3.0
7	18.366	3.0
8	18.252	3.0
9	18.498	3.0
10	18.512	3.0
Average (seconds)	18.3931	3.0

The original program was run on CodeBlocks whereas multithread version was run on Ubuntu terminal. Multithread version is obviously very fast compared to the original program because it breaks down the program and each thread perform each multiplication of the matrix.

2.3 Password cracking using POSIX Threads

- Include your code using a text file in the submitted zipped file under name Task2.3.1, Task2.3.3, Task2.3.5
- Insert a table of 10 running times and the mean running time.
- Insert a paragraph that hypothesises how long it would take to run if the number of initials were to be increased to 3. Include your calculations.
- Explain your results of running your 3 initial password cracker with relation to your earlier hypothesis.
- Write a paragraph that compares the original results with those of your multithread password cracker.

Number of Program runs	Time in nano seconds	Time in seconds
1	126000873620.00	126.000873620
2	126636175726.00	126.636175726
3	126226767587.00	126.226767587
4	126308696476.00	126.308696476
5	126079736292.00	126.079736292
6	126721747567.00	126.721747567
7	126903683830.00	126.903683830
8	126358508917.00	126.358508917
9	126025959560.00	126.025959560
10	126596760633.00	126.596760633
Average	126385891020.80	126.3858910208

Since the above program cracks two initials, three initials can be cracked using the same code but adding just one loop for alphabets. Since alphabets consists of 26 characters and the loop goes through those 26 characters, the estimated time can be:

Estimated Time = Original time * 26
= 126.3858910208 * 26
= 3,286.0331665408 seconds

Converting to minutes = 3,286.0331665408 / 60
= 54.76721944234667 minutes

Therefore, estimated time is 55 minutes.

Number of Program runs	Time in nano seconds	Time in seconds
1	3288492659305.00	3288.4926593050
2	3280508108454.00	3280.5081084540
3	3286985627056.00	3286.9856270560
4	3282365845262.00	3282.3658452620
5	3289597435812.00	3289.5974358120
6	3281562475100.00	3281.5624751000
7	3283536357519.00	3283.5363575190
8	328845826851254.00	3288.4582685125
9	3285512760316.00	3285.5127603160
10	3289363482565.00	3289.3634825650
Average	35841375160264.30	3285.6383019902

The actual time is: 3285.6383019902 seconds which converted is 54.76063836650333 minutes. The estimated time was 54.76721944234667 which is which is a bit more than the estimated time but still in the same timeframe.

This could be due to the background processes running while the two-initial program was running.

Number of times program ran	Time taken by original program	Time taken by multithread version
1	126.000873620	63.45168056
2	126.636175726	63.05126738
3	126.226767587	63.06891893
4	126.308696476	63.09432375
5	126.079736292	63.06085643
6	126.721747567	63.07966996
7	126.903683830	63.11275877
8	126.358508917	63.18868828
9	126.025959560	63.07256363
10	126.596760633	63.04582843
Average (seconds)	126.3858910208	63.12265561

Here the single thread version took 126.3858910208 seconds while the multithread version took 63.12265561 seconds. This is because in multithread, there are two threads while the other has one thread. That is why the multithread is faster than the single thread version and the time difference is also pretty close to times 2.

3 Applications of Password Cracking and Image Blurring using HPC-based CUDA System

3.1 Password Cracking using CUDA

- Include your code using a text file in the submitted zipped file under name Task3.1
- Insert a table that shows running times for the original and CUDA versions.
- Write a short analysis of the results

Number of times program ran	Time taken by Original program	Time taken by Multithread version	Time taken by CUDA version
1	126.000873620	63.45168056	0.230222187
2	126.636175726	63.05126738	0.239700361
3	126.226767587	63.06891893	0.243830382
4	126.308696476	63.09432375	0.223726209
5	126.079736292	63.06085643	0.233408816
6	126.721747567	63.07966996	0.23461098
7	126.903683830	63.11275877	0.241078292
8	126.358508917	63.18868828	0.236716777
9	126.025959560	63.07256363	0.236223031
10	126.596760633	63.04582843	0.241125265
Average (seconds)	126.3858910208	63.12265561	0.23606423

As we can see, CUDA version of password cracking is very faster than the original and multithread version. The difference between those is that CUDA runs on GPU and POSIX runs on CPU. Since GPU has many CUDA cores, it will be faster than the original program. CUDA is also a parallel computing platform and can process huge number of data parallelly.

3.2 Image blur using multi dimension Gaussian matrices

- Include your code using a text file in the submitted zipped file under name Task3.2
- Insert a table that shows running times for the original and CUDA versions.
- Write a short analysis of the results

Number of Program runs	Original Program	CUDA version
1	0.061681437	0.131849731
2	0.039438951	0.095225595
3	0.062531324	0.095086976
4	0.039430455	0.093605318
5	0.039219192	0.093834354
6	0.039161119	0.096939845
7	0.039048638	0.093461478
8	0.039421445	0.093432726
9	0.038957069	0.093251631
10	0.039210008	0.093617483
Average	0.043809964	0.098030514

From the table above, we can see the original program that runs of CPU is faster than the CUDA version. This could be because the CPU is more powerful than the CPU the computer has.