6CS005 Learning Journal - Semester 1 2020/21

Subin Shrestha, 2039281

**Table of Contents**

[1 Parallel and Distributed Systems 2](#_Toc59795530)

[1.1 Answer of First Question 2](#_Toc59795531)

[1.2 Answer of Second Question 2](#_Toc59795532)

[1.3 Answer of Third Question 2](#_Toc59795533)

[1.4 Answer of Fourth Question 3](#_Toc59795534)

[1.5 Answer of Fifth Question 3](#_Toc59795535)

[1.6 Answer of Sixth Question 5](#_Toc59795536)

[2 Applications of Matrix Multiplication and Password Cracking using HPC-based CPU system 5](#_Toc59795537)

[2.1 Single Thread Matrix Multiplication 5](#_Toc59795538)

[2.2 Multithreaded Matrix Multiplication 6](#_Toc59795539)

[2.3 Password cracking using POSIX Threads 6](#_Toc59795540)

[3 Applications of Password Cracking and Image Blurring using HPC-based CUDA System 6](#_Toc59795541)

[3.1 Password Cracking using CUDA 6](#_Toc59795542)

[3.2 Image blur using multi dimension Gaussian matrices 6](#_Toc59795543)

# Parallel and Distributed Systems

## Answer of First Question

Threads are the smallest unit of computational process executed by the CPU.

Computing a single task at a time takes a long time to compute. Threads are designed to compute independent processes. This enables CPU to execute multiple independent threads simultaneously saving time. Thus, CPU were designed so that CPUs could perform computation parallelly.

## Answer of Second Question

The two types of Process scheduling policies are Pre-emptive and Co-operative.

**Pre-emptive**: In pre-emptive process scheduling the scheduler decides how long each process runs for and allocates time for each process. If a process exceeds its allocated time, it is stopped by the scheduler.

**Co-operative**: In Co-operative process scheduling the process decides how long it runs for. Only when process has completed its execution, it stops and allows other process to execute.

Pre-emptive is preferred as if there is a flaw in design of a process and it does not stop, the process will take valuable computational/CPU resources. In pre-emptive scheduling all the process will have access to CPU resources and will result in high CPU utilization.

###################### JAVA QUES LEFT

## Answer of Third Question

**Centralized System**: Centralized System are systems where all the programs are executed at a same computer. All the users always share the same resources and has single point of control and failure and security is easier to maintain.

**Distributed System**: Distributed systems are systems where all the programs are executed on multiple independent autonomous interconnected computers. Resources are spread across multiple computers and has multiple point of failure. Distributed System has high fault tolerance as even if some part of the system fails the whole system continues to function. Resources in distributed systems are effectively utilized and has significantly more computational resources than centralized systems. Maintaining security and difficulty in developing software for distributed systems are a major problem in distributed systems.

## Answer of Fourth Question

Transparency in Distributed systems means to hide from the users that their processes are running on multiple independent computers. Users perceive that computation is being done on a single centralized system rather than on multiple interconnected systems. This is important in Distributed Systems as it is easier and intuitive for users to use single processor systems.

Types of Transparency in Distributed Systems:

Access Transparency: Local and Remote resources are accessed using identical operations.

Local Transparency: Resources are accessed without knowing their physical location.

Migration Transparency: Location of resources can be moved without any effect in the process.

Replication Transparency: Users do not know how many copies/backups of data exit.

Concurrency Transparency: Multiple users can use same resource concurrently without interference between them.

Failure Transparency: Users do not know if the failure of resource occurs.

Performance Transparency: Resources are allocated and utilized to improve performance.

Scaling Transparency: System can scale be scaled up or down without changing system structure or application programs.

## Answer of Fifth Question

B = A + C ------ 1

B = C + D ------ 2

C = B + D ------ 3

Identification of dependency

3 has flow dependency with 2 and 1 as value of B must be calculate before 3 can be executed.

1, 2 and 3 have anti dependency as 1 and 2 requires value of C is calculated which is at 3rd step. So, 1 and 2 must wait for 3 to be computed to calculate the value of C and use it in 1 and 2.

1 and 2 have output dependency as the value of B changes after each step. So, changing the order of 1 and 2 will affect the final output value of B. This will also affect the value of C as the value of C depends on value of B.

Part 3

Only reordering the statements does not produce same values for B and C as in original statements as the value of B truly depends on C and the value of C truly depends on B.

Part 3

C = B\_old + D --- 1

B\_new = A + C --- 2

B = C + D --- 3

This solves arrangement of statements removes anti dependencies and output dependency.

This Also enables statement 2 and 3 to be executed in parallel.

Anti-dependency is solved as no value is updated after it is required. Output dependency is solved as no value is updated once it is computed.

Flow Dependency cannot be solved as flow dependencies cannot be removed without Changing the purpose of the program.

* Code is included in the file: 2039281\_Task1\_5.c

## Answer of Sixth Question

Code on left (first code) gives random values as output each time it is run.

Outputs on running the code 5 times are:

188444, 208898 ,193965, 217172, 242647

Counter is a global variable and thread\_func updates it. The thread uses the current value of counter in the thread when the thread is created. Since the threads are created while other threads are running the value of counter is changing. So, when the threads are joined the value of counter is different.

Code on the right (second code) gives a constant output of 500000 each time it is run.

Counter is a global variable and thread\_func updates it. The thread uses the current value of counter in the thread when the thread is created. The threads are created only when the previous thread is joined, meaning new thread is created the previous thread is already completed. Since the thread\_func adds 100000 to counter, after running for 5 times the value is 500000

* The codes are included as task1\_6A.c and task1\_6B.c in Extra code folder.

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

## Single Thread Matrix Multiplication

* The analysis of the algorithm’s complexity. (1 mark)

Algorithm as 3 loops each running N, M and P times. Considering each calculation takes unit time:

The complexity of Algorithm is O(N\*M\*P).

If the matrices are square matrices with size n. Then,

The complexity of Algorithm is O(n3)

* 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)
  + The Three ways to improve are:
    - Using cache memory to prevent the access of C[i][j] value at every step. This saves access time.
    - Using matrix vector multiplication.
    - Using multithreading to compute multiple results at once.
* 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)

Int A[N][P], B[P][M], C[N][M];

int cache;

for (int i = 0;i < N; i++)

{

for (int j = 0; j < M; j++)

{

cache = 0;

for (int k = 0; k < P; k++)

{

cache = cache + A[i][k] \* B[k][j];

}

c[i][j] = cache;

}

}

* 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)

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.

## 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.

## 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.

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

## 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

## 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