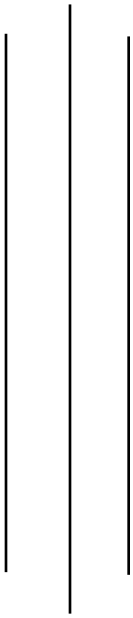

Programme Name: **BCS HONS**

Course Code: **CSC 2624**

Course Name: **Distributed And Parallel Computing**

Open Book Examination

Date of Submission: **9/20/2021**

Submitted By:

Student Name**: Dipesh Tha Shrestha**

IUKL ID: **041902900028**

Semester**: Fourth Semester**

Intake**: September 2019**

Submitted To:

Faculty Name**: Manoj Gautam**

Department**: LMS**

1. **Most programming problems have several parallel solutions. The best solution may differ from that suggested by existing sequential algorithms Explain The design methodology describe by foster to design a machine independent parallel program.**

Answer:

A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem. Parallel computing has become an important subject in the field of computer
science and has proven to be critical when researching high performance solutions. The evolution of computer architectures (multi-core and many-core) towards a higher number of cores can only confirm that parallelism is the method of choice f or speeding up an algorithm.

Without some logical methodology, designing a parallel program from scratch is difficult. It is far preferable to use a tried-and-true methodology that is both general and simple to follow. Otherwise, you won't be able to learn from other people's mistakes. Ian Foster proposed such a methodology in 1995, which became known as Foster's design methodology. Foster's strategy is well suited for computational physics because it handles data-parallel problems in a natural way. At the same time, Foster's strategy also works well for designing massive parallel GPU-based algorithms. In order to apply this strategy, it is necessary to know how the massive parallelism programming model works for mapping the computational resources to a data-parallel problem and how to overcome the technical restrictions when programming the GPU. As shown in the diagram, it is a four-stage design process whose input is the problem statement. The four stages are listed below, along with brief descriptions.

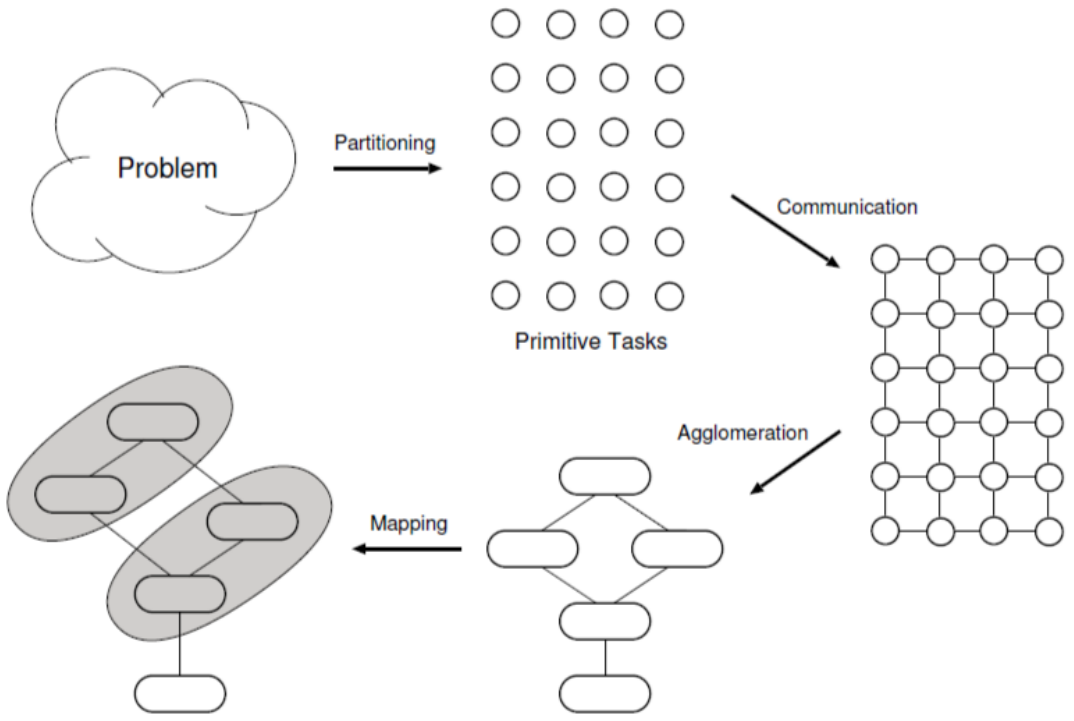

Fig: Foster's parallel algorithm design methodology.

- Partitioning

Partitioning's goal is to uncover as much parallelism as possible. The goal in this stage is to find all of it; the remaining stages typically reduce the amount of parallelism, so the goal in this stage is to find it all. Data and computation are two potential sources of parallelism, resulting in two complementary methods for extracting parallelism.

- Communication

The process of determining how tasks will communicate with each other, distinguishing between local communication and global communication. Within the tasks, there are two types of communication: local and global. When a task requires values from a small number of other tasks, it is referred to as local communication. When a large number of tasks must contribute data to perform a computation, this is referred to as global communication.

- Agglomeration

The process of grouping tasks into larger tasks to improve performance or simplify programming. The third stage, agglomeration, revisits the decisions made during the partitioning and communication phases. Agglomeration is the process of combining two or more tasks into one larger task in order to reduce the number of tasks and increase their size. Its goal is to boost performance and make programming easier. Agglomeration is an optimization problem; the goals are frequently in conflict, and trade-offs must be made.

- Mapping

The final stage of Foster's methodology is mapping, which involves assigning each task to a processor. Of course, on uniprocessors or shared-memory computers with operating systems that provide automatic task scheduling, this mapping problem does not exist. As a result, we'll assume the target architecture is a distributed-memory parallel computer, also known as a private memory multicomputer. The goal of mapping algorithms, from the perspective of a parallel programmer, is to reduce total execution time, plain and simple. This is typically accomplished by reducing inter processor communication while increasing processor utilization.

Therefore, this above Explain The design methodology describe by foster to design a machine independent parallel program.

2. **OpenMP allows you to parallelize your code with minimal effort. Write an OpenMp program to print the message given bellow by different threads.**

Parent: 4 Thread running
Thread # 2: running
Thread # 3: running
Thread # l: running

Answer:

An OpenMp program to print the message given bellow by different threads is given below:
Code:

```
#include<iostream>
#include<omp.h>
using namespace std;

int main()
{
    cout<<"Parent: 4 Thread running"<<"\n";
    #pragma omp parallen num_threads(3)
    {
        #pragma omp thread1
        {
            cout<<"Thread # 2:running"<<"\n";
        }
         #pragma omp thread2
        {
            cout<<"Thread # 3:running"<<"\n";
        }
         #pragma omp thread3
        {
            cout<<"Thread # 1:running"<<"\n";
        }

    }


}
```

Output:

```
PS C:\Fourth Semester\dISTRIBUTED AND PARALLEL CO
UTING\Final Examination\" ; if ($?) { g++ openrmp
Parent: 4 Thread running
Thread # 2:running
Thread # 3:running
Thread # 1:running
PS C:\Fourth Semester\dISTRIBUTED AND PARALLEL CO
```

3. **The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language. Write a complete java RMI program to compute the roots of a quadratic equation $ax^2 + bx + c$ such that the equation coefficients a, b and c are passed to the remote String computeRoot(double a, double b, double c) method from the client side.**

(Note: Interface name should be QuadraticComputation)

Answer:
As we know, The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton.

```
import java.rmi.*;
public interface Adder extends Remote{
public int quadratic(double a, double b, double c)throws RemoteException;
}
```

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int quadratic(double a, double b, double c){return ax 2 + bx + c;}
}
```

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
    quadratic stub=new quadraticRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
    quadratic stub=(quadratic)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.quadratic(3,4,4));
}catch(Exception e){}
}
}
```

```
import java.util.Scanner;
public class QuadraticEquationExample1
    {
    public static void main(String[] Strings)
    {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the value of a: ");
    double a = input.nextDouble();
    System.out.print("Enter the value of b: ");
    double b = input.nextDouble();
System.out.print("Enter the value of c: ");
    double c = input.nextDouble();
double d= b * b - 4.0 * a * c;
    if (d> 0.0)
    {
    double r1 = (-b + Math.pow(d, 0.5)) / (2.0 * a);
    double r2 = (-b - Math.pow(d, 0.5)) / (2.0 * a);
    System.out.println("The roots are " + r1 + " and " + r2);
    }
    else if (d == 0.0)
{
    double r1 = -b / (2.0 * a);
    System.out.println("The root is " + r1);
    }
    else    {
    System.out.println("Roots are not real.");
    }
    }
    }
```

4. **A socket is one end-point of a two-way communication link between two programs running on the network. Write a TCP client and a server socket program to compute the power of a number send by the client to the server (Note: server must listen on port 3000)**

Answer:

As we know, we need 2 folders (client and server) to run the above program. In given folder it will have its own files which will help them to connect with each other. Folder client will have 3 files named as Client.java, Number.java and RemoteCalcObject.java whereas Folder server will have Server.java, Number.java and NumberImpl.java. Therefore, Code for above program is given below:

Folder Server

Server.java

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server {
    public static void main(String[] args) {
        try {
            NumberImpl n1 = new NumberImpl(4);
            Number stub1 = (Number) UnicastRemoteObject.exportObject(n1, 0);
            Registry registry = LocateRegistry.getRegistry("127.0.0.1", 3000);
            registry.bind("number", stub1);
        } catch (Exception e) {
            System.out.println("Error :" + e);
        }
```

```
    }
}
```

Number.java

```
import java.rmi.*;

public interface Number extends Remote {
    public double getNum() throws RemoteException;
}
```

NumberImpl.java

```
import java.rmi.*;
import java.rmi.server.*;

public class NumberImpl implements Number{

    double numm;

    NumberImpl(double newnumm) throws RemoteException{
        this.numm = newnumm;
    }
    public double getNum() throws RemoteException{
        return this.numm;
    }
}
```

Folder Client

Client.java

```
import java.rmi.*;
import java.rmi.registry.*;

public class Client {
    public static void main(String[] args) throws RemoteException, NotBoundExcept
ion {
     try {
        Registry remoteRegistry = LocateRegistry.getRegistry("127.0.0.1", 3000);
```

```
         Number numm = (Number) remoteRegistry.lookup("number");
         RemoteCalcObject remoteCalcObject = new RemoteCalcObject();
         double finalnum = remoteCalcObject.computerPower(numm.getNum());
          System.out.println("The power of " + numm.getNum() + " by 2 is " + fina
lnum);
        } catch (Exception e) {
            System.out.println("Clinet error occoured " + e.toString());
        }

    }
}
```

Number.java

```
import java.rmi.*;

public interface Number extends Remote {
    public double getNum() throws RemoteException;
}
```

RemoteCalcObject.java

```
import java.lang.Math;

class RemoteCalcObject {
    RemoteCalcObject() {

    }

    public double computerPower(double num) {
        return Math.pow(num, 2);
    }
}
```

Output:

```
C:\Users\dell\Desktop\Final>javac Server.java

C:\Users\dell\Desktop\Final>java Server
Server is listening at port: 3000
```

```
C:\Users\dell\Desktop\Final>javac Client.java

C:\Users\dell\Desktop\Final>java Client
Enter number:

7
Enter power number:

2
number is 7
the powerof given number is: 49.0
```

5. **Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Explain the cache design issues from computer architecture prespective.**

Answer:

Cache memory, also known as a cache store or RAM cache, is a section of memory that uses high-speed static RAM (SRAM) rather than the slower and less expensive dynamic RAM (DRAM) used for main memory. Because most programs access the same data or instructions repeatedly, memory caching is effective. The cache memory's primary location in the computer structure was between the main memory and the CPU core, as depicted in figure 1 as a simple and minimal cache memory configuration. It is similar to the architecture found in early systems that used CPU caches.

The cache design issues from computer architecture prespective are given below:

- **Cache Size**

The larger the cache, the larger the number of gates involved in addressing the cache. The available chip and board area also limit cache size. The more cache a system has, the more likely it is to register a hit on memory access because fewer memory locations are forced to share the same cache line. Although an increase

in cache size will increase the hit ratio, a continuous increase in cache size will not yield an equivalent increase of the hit ratio.

Line Size

The line size is another design element. When a block of data is retrieved and stored in the cache, it includes not only the desired word, but also a number of adjacent words. Because of the principle of locality, which states that data in the vicinity of a referenced word is likely to be referenced in the near future, the hit ratio will initially increase as the block size increases from very small to larger sizes. More useful data is brought into the cache as the block size grows. However, as the block grows larger, the probability of using newly fetched information becomes less than the probability of reusing the information that must be replaced, the hit ratio will begin to decline.

Mapping Function

The mapping function determines which cache location a replacement block of data will occupy when it is scanned into the cache. Two constraints have an impact on the mapping performance planning. After one block has been scanned in, another could be replaced. We'd like to do it in the simplest way possible to reduce the chances of having to replace a block that will be needed in the near future. The more flexible the mapping function, the more scopes we'll need to design a replacement algorithmic rule to maximize the hit magnitude relationship. Second, the more versatile the mapping is, the more advanced the electronic equipment required to check the cache to see if a given block is present.

Replacement Algorithm

When a new block is added to the cache after it has been filled, one of the existing blocks must be replaced. There is only one possible line for each block in direct mapping, and there is no choice. Direct mapping is the only option because each block only maps to a single line. That line should be replaced. A replacement algorithm is required for the associative and set-associative techniques. Such an algorithm must be implemented in hardware to achieve high speed.

Write Policy

If the contents of a cache block are changed, it must be written back to main memory before being exchanged. When the memory write operation takes place,

the written policy dictates. The writing will occur whenever the block is updated at one extreme. At the other end of the spectrum, writing occurs only when the block is replaced. While the latter policy reduces memory write operations, it leaves the main memory in a state of obsolescence. This may cause issues with multiprocessor operation and direct I/O hardware module operation.

Thank You