Module-3

5 a. Describe the relationship between a process and a thread. (08 Marks)
b. Explain with an example, Domain Decomposition. Write Aneka code to create matrix product class. (08 Marks)

OR

6 a. What is task computing? Describe parameter sweep application with an example. (10 Marks)
b. List and explain Aneka ready – to – use task libraries. (06 Marks)

5A)

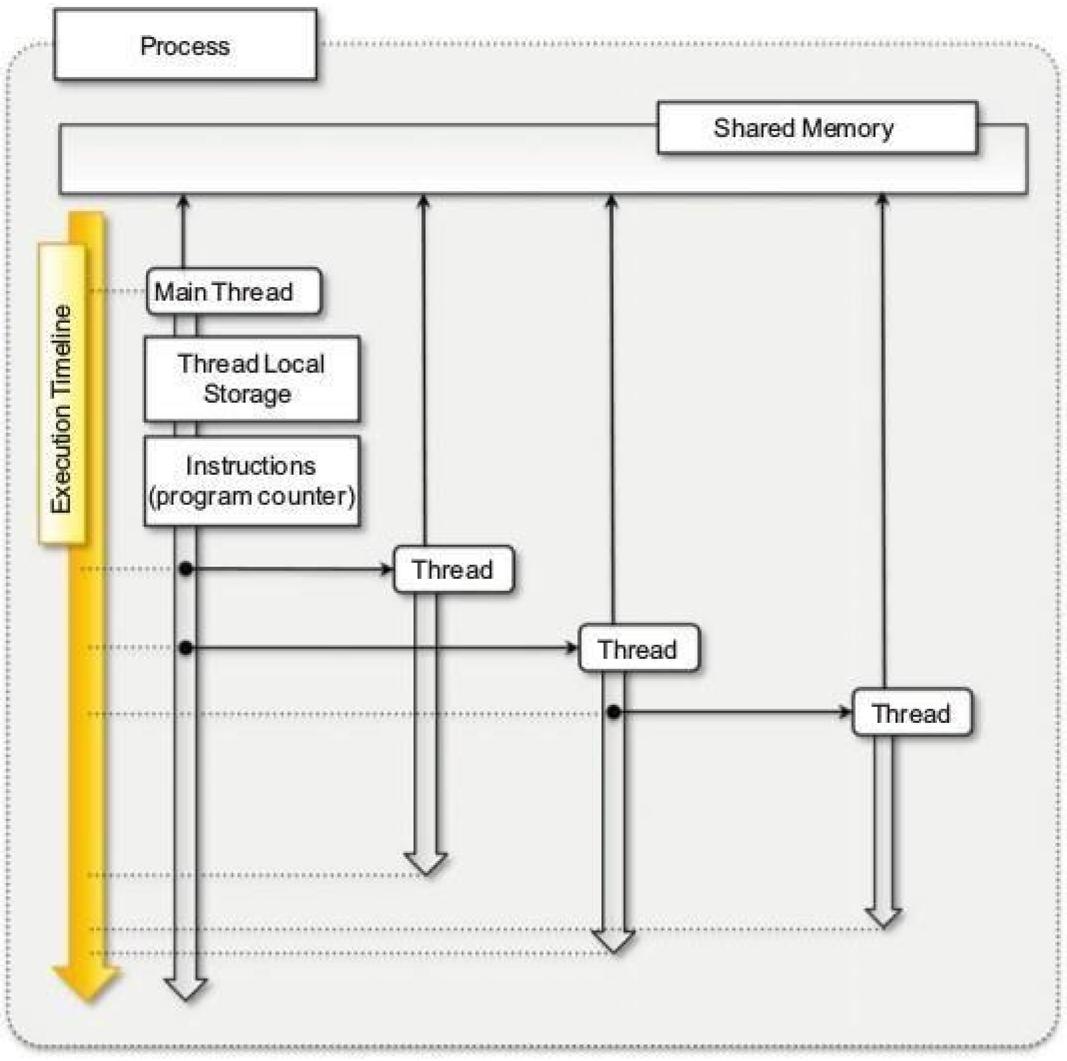


FIGURE 6.2

The relationship between processes and threads.

- **Figure 6.2** provides an overview of the relation between threads and processes and a simplified representation of the runtime execution of a multithreaded application.
- A running program is identified by a process, which contains at least one thread, also called the main thread. Such a thread is implicitly created by the compiler or the runtime environment executing the program.
- This thread is likely to last for the entire lifetime of the process and be the origin of other threads, which in general exhibit a shorter duration.
- As main threads, these threads can spawn other threads. There is no difference between the main thread and other threads created during the process lifetime.
- Each of them has its own local storage and a sequence of instructions to execute, and they all share the memory space allocated for the entire process.
- The execution of the process is considered terminated when all the threads are completed.

Thread APIs

1 POSIX Threads

2 Threading support in java and .NET

1 POSIX Threads
Portable Operating System Interface for Unix (POSIX) is a set of standards related to the application programming interfaces for a portable development of applications over the Unix operating system flavors. Standard POSIX 1.c (IEEE Std 1003.1c-1995) addresses the implementation of threads and the functionalities that should be available for application programmers to develop portable multithreaded applications.

2 Threading support in java and .NET
Languages such as Java and C# provide a rich set of functionalities for multithreaded programming by using an object-oriented approach. Both Java and .NET execute code on top of a virtual machine, the APIs exposed by the libraries refer to managed or logical threads. These are mapped to physical threads.

- Both Java and .NET provide class Thread with the common operations on threads: start, stop, suspend, resume, abort, sleep, join, and interrupt.
- Start and stop/abort are used to control the lifetime of the thread instance.

5B) **1. Domain decomposition**

Domain decomposition is the process of identifying patterns of functionally repetitive, but independent, computation on data. This is the most common type of decomposition in the case of throughput computing, and it relates to the identification of repetitive calculations required for solving a problem. The master-slave model is a quite common organization for these scenarios:

- • The system is divided into two major code segments.
- • One code segment contains the decomposition and coordination logic.
- • Another code segment contains the repetitive computation to perform.
- • A master thread executes the first code segment.
- • As a result of the master thread execution, as many slave threads as needed are created to execute the repetitive computation.
- • The collection of the results from each of the slave threads and an eventual composition of the final result are performed by the master thread.

Embarrassingly parallel problems constitute the easiest case for parallelization because there is no need to synchronize different threads that do not share any data. Embarrassingly parallel problems are quite common; they are based on the strong assumption that at each of the iterations of the decomposition method, it is possible to isolate an independent unit of work. This is what makes it possible to obtain a high computing throughput.

inherently sequential
If the values of all the iterations are dependent on some of the values obtained in the previous iterations, the problem is said to be **inherently sequential**. **Figure 6.3** provides a schematic representation of the decomposition of embarrassingly parallel and inherently sequential problems. The matrix product computes each element of the resulting matrix as a linear combination of the corresponding row and column of the first and second input matrices, respectively. The formula that applies

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik}B_{kj}$$

for each of the resulting matrix elements is the following:
Two conditions hold in order to perform a matrix product:

- • Input matrices must contain values of a comparable nature for which the scalar product is defined.
- • The number of columns in the first matrix must match the number of rows of the second matrix

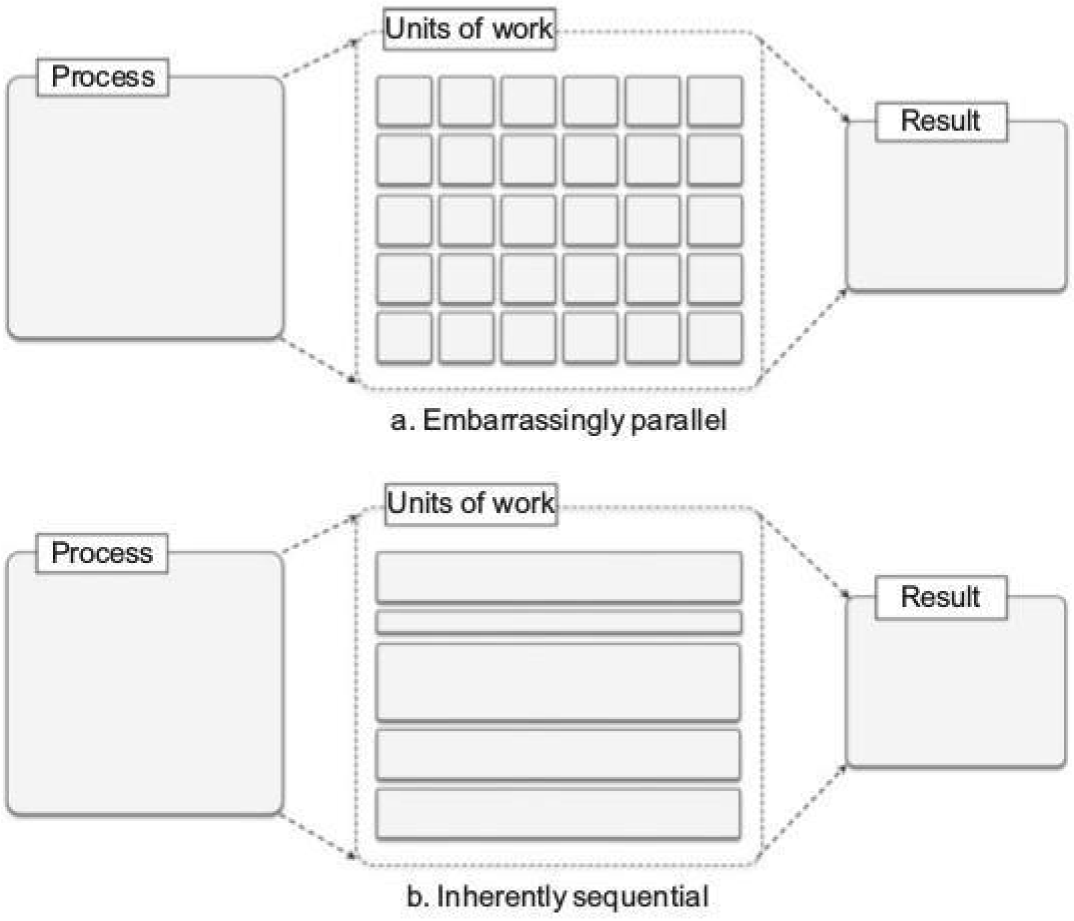


FIGURE 6.3

Domain decomposition techniques.

•*6.2.3.1 Domain decomposition(cond)*

•***Scalar Product***

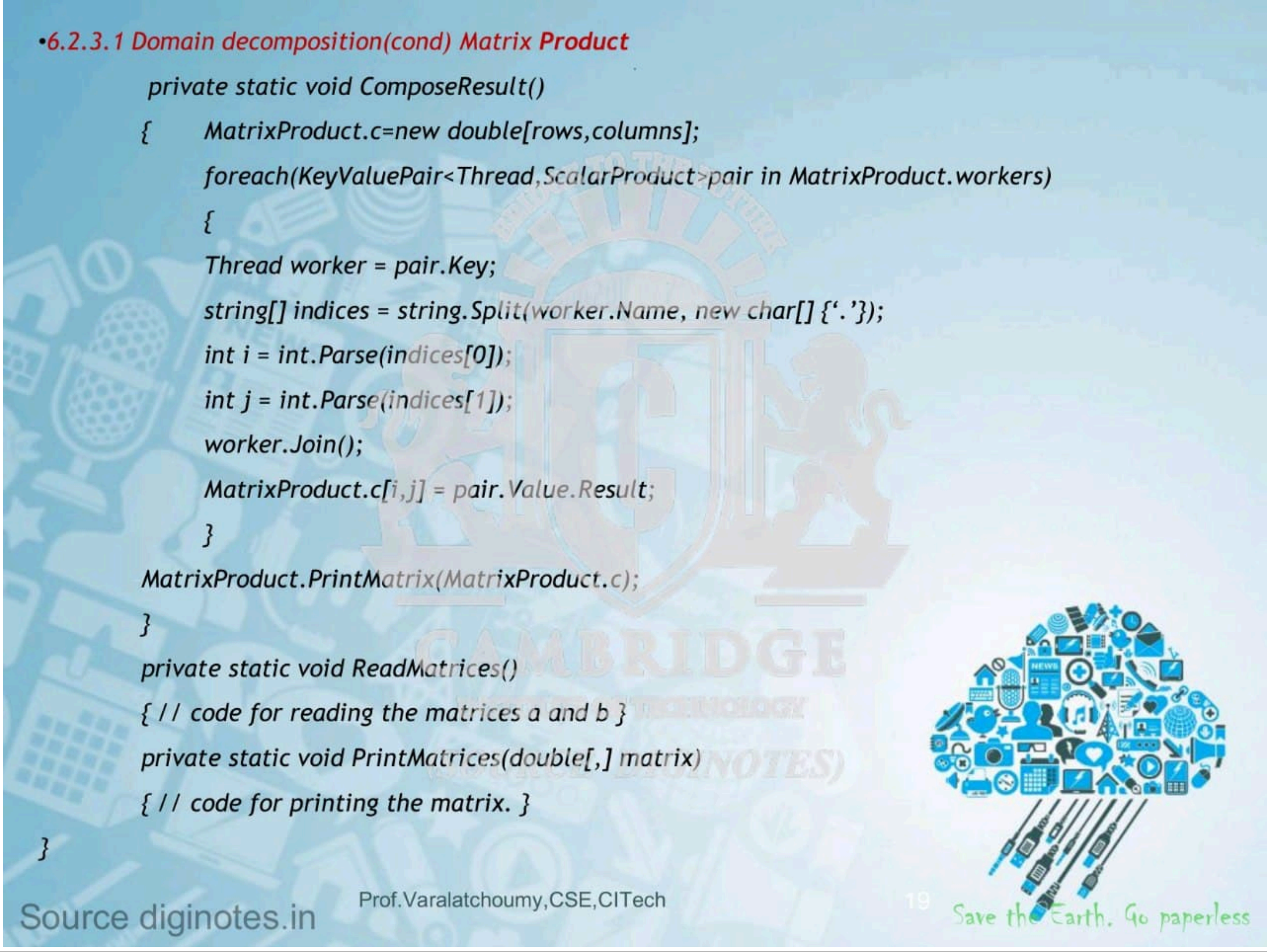
```
public class ScalarProduct
{
            private double result;
            private double[] row, column;
            public ScalarProduct(double[] row, double[] column)
            {
            this.row = row;
            this.column = column;
            }
            public void Multiply()
            {
            this.result = 0;
                    for(int i=0; i<this.row.Length; i++)
                    {
                    this.result += this.row[i] * this.column[i];
                    }
             }}
```

•6.2.3.1 Domain decomposition(cond) Matrix **Product**

```
using System; using System.Threading; using System.Collections.Generic;
public class MatrixProduct
{
        private static double[,]a, b;
        private static double[,] c;
/// Dictionary mapping the thread instances to the corresponding ScalarProduct instances that are run
 inside.
        private static IDictionary<Thread, ScalarProduct>workers.
public static void Main(string[] args)
{
        MatrixProduct.ReadMatrices();
        MatrixProduct.ExecuteThreads();
        MatrixProduct.ComposeResult();
}
private static void ExecuteThreads()
{
MatrixProduct.workers = newList<Thread>();
int rows = MatrixProduct.a.Length;
```

•6.2.3.1 Domain decomposition(cond) Matrix **Product**

```
        int columns = MatrixProduct.b.Length;
        for(int i=0; i<rows; i++)
        for(int j=0; j<columns; j++)
        {
        double[] row = MatrixProduct.a[i];
        double[] column = new double[column];
        for(int k=0; k<column; k++)
        {
        column[j] = MatrixProduct.b[j][i];
        }
        }
        ScalarProduct scalar = newScalarProduct(row, column);
        Thread worker = newThread(newThreadStart(scalar.Multiply));
        worker.Name =string.Format("{0}.{1}",row,column); worker.Start();
        MatrixProduct.workers.Add(worker, scalar);
        }
        }
```



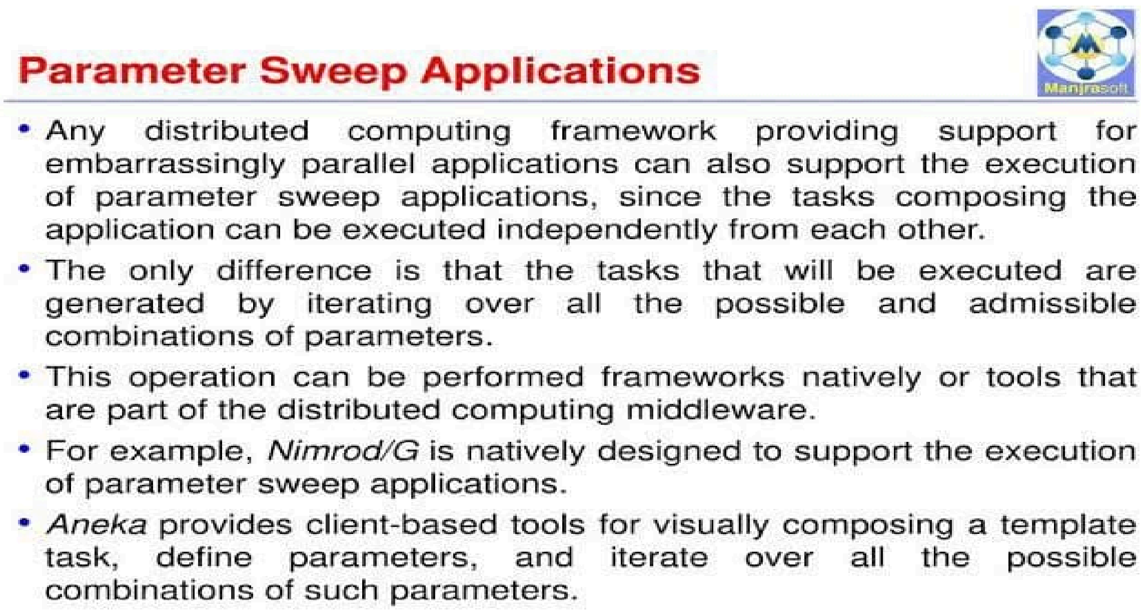
•6.2.3.1 Domain decomposition(cond) Matrix Product

```
          private static void ComposeResult()
         {      MatrixProduct.c=new double[rows,columns];
                foreach(KeyValuePair<Thread,ScalarProduct>pair in MatrixProduct.workers)
                {
                Thread worker = pair.Key;
                string[] indices = string.Split(worker.Name, new char[] {'.'});
                int i = int.Parse(indices[0]);
                int j = int.Parse(indices[1]);
                worker.Join();
                MatrixProduct.c[i,j] = pair.Value.Result;
                }
         MatrixProduct.PrintMatrix(MatrixProduct.c);
         }
         private static void ReadMatrices()
         { // code for reading the matrices a and b }
         private static void PrintMatrices(double[,] matrix)
         { // code for printing the matrix. }
}
```

6A)

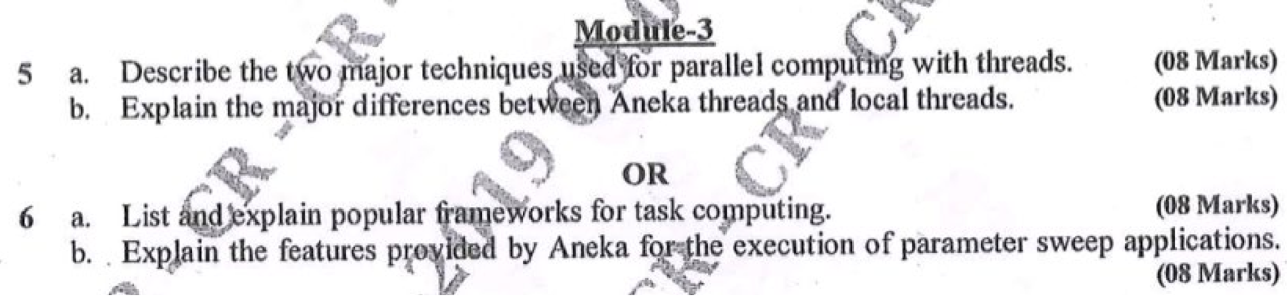
Task computing is a computation meant to fill the gap between tasks (what the user wants to be done) and services (functionalities that are available to the user). Task computing seeks to redefine how users interact with and use computing environments.

4. Parameter Sweep Model. This model is a specialized form of Task Model for applications that can be described by a template task whose instances are created by generating different combinations of parameters.



Parameter Sweep Applications

- Any distributed computing framework providing support for embarrassingly parallel applications can also support the execution of parameter sweep applications, since the tasks composing the application can be executed independently from each other.
- The only difference is that the tasks that will be executed are generated by iterating over all the possible and admissible combinations of parameters.
- This operation can be performed frameworks natively or tools that are part of the distributed computing middleware.
- For example, *Nimrod/G* is natively designed to support the execution of parameter sweep applications.
- *Aneka* provides client-based tools for visually composing a template task, define parameters, and iterate over all the possible combinations of such parameters.



Module-3

5 a. Describe the two major techniques used for parallel computing with threads. (08 Marks)
b. Explain the major differences between Aneka threads and local threads. (08 Marks)

OR

6 a. List and explain popular frameworks for task computing. (08 Marks)
b. Explain the features provided by Aneka for the execution of parameter sweep applications. (08 Marks)

5A) **6.2.2 Techniques for parallel computation with threads**
Developing parallel applications requires an understanding of the problem and its logical structure. Decomposition is a useful technique that aids in understanding whether a problem is divided into components (or tasks) that can be executed concurrently. it allows the breaking down into independent units of work that can be executed concurrently with the support provided by threads.
1 Domain decomposition
2 Functional decomposition

3 Computation vs. Communication

5B) **1. Domain decomposition**

Domain decomposition is the process of identifying patterns of functionally repetitive, but independent, computation on data. This is the most common type of decomposition in the case of throughput computing, and it relates to the identification of repetitive calculations required for solving a problem. The master-slave model is a quite common organization for these scenarios:

- • The system is divided into two major code segments.
- • One code segment contains the decomposition and coordination logic.
- • Another code segment contains the repetitive computation to perform.
- • A master thread executes the first code segment.
- • As a result of the master thread execution, as many slave threads as needed are created to execute the repetitive computation.
- • The collection of the results from each of the slave threads and an eventual composition of the final result are performed by the master thread.

Embarrassingly parallel problems constitute the easiest case for parallelization because there is no need to synchronize different threads that do not share any data. Embarrassingly parallel problems are quite common; they are based on the strong assumption that at each of the iterations of the decomposition method, it is possible to isolate an independent unit of work. This is what makes it possible to obtain a high computing throughput.

inherently sequential
If the values of all the iterations are dependent on some of the values obtained in the previous iterations, the problem is said to be **inherently sequential**. **Figure 6.3** provides a schematic representation of the decomposition of embarrassingly parallel and inherently sequential problems. The matrix product computes each element of the resulting matrix as a linear combination of the corresponding row and column of the first and second input matrices, respectively. The formula that applies

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}$$

for each of the resulting matrix elements is the following:

Two conditions hold in order to perform a matrix product:

- • Input matrices must contain values of a comparable nature for which the scalar product is defined.
- • The number of columns in the first matrix must match the number of rows of the second matrix

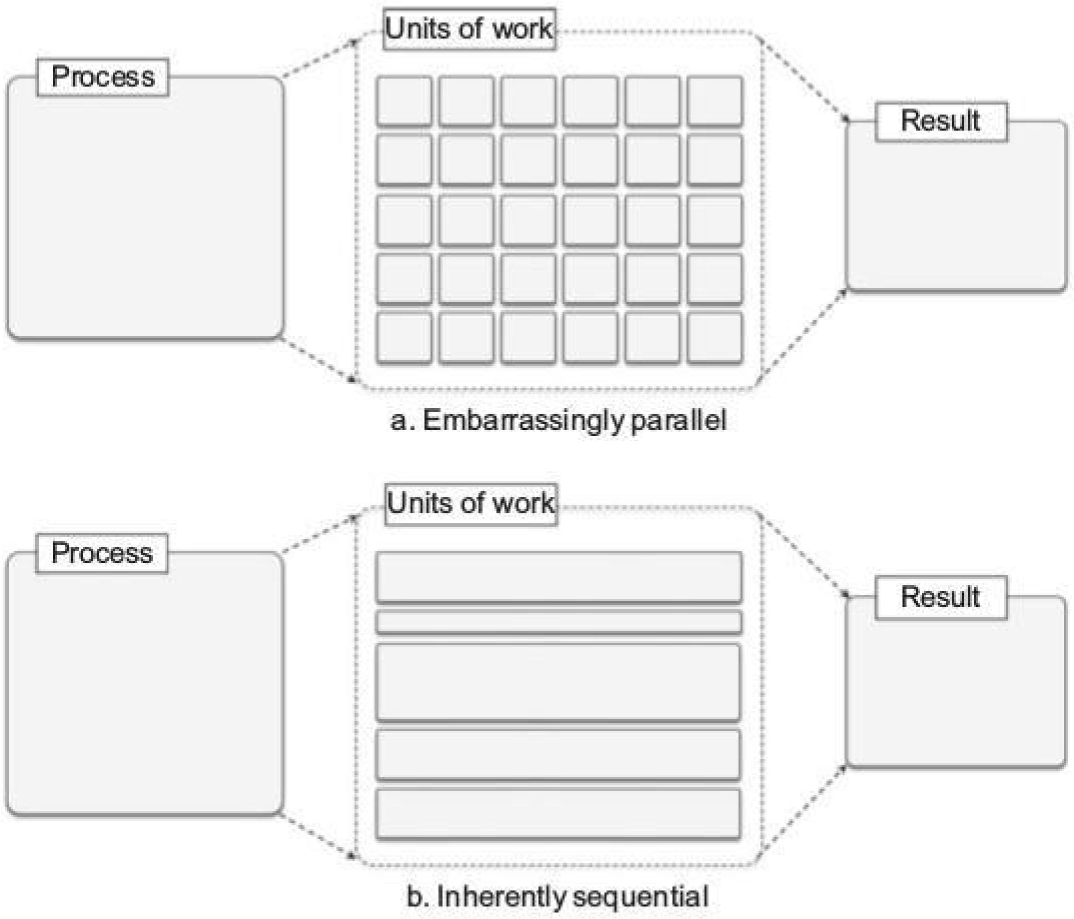


FIGURE 6.3

Domain decomposition techniques.

Functional decomposition

Functional decomposition is the process of identifying functionally distinct but independent computations. The focus here is on the type of computation rather than on the data manipulated by the computation.

This kind of decomposition is less common and does not lead to the creation of a large number of threads, since the different computations that are performed by a single program are limited.

Functional decomposition leads to a natural decomposition of the problem in separate units of work. **Figure 6.5** provides a pictorial view of how decomposition operates and allows parallelization.

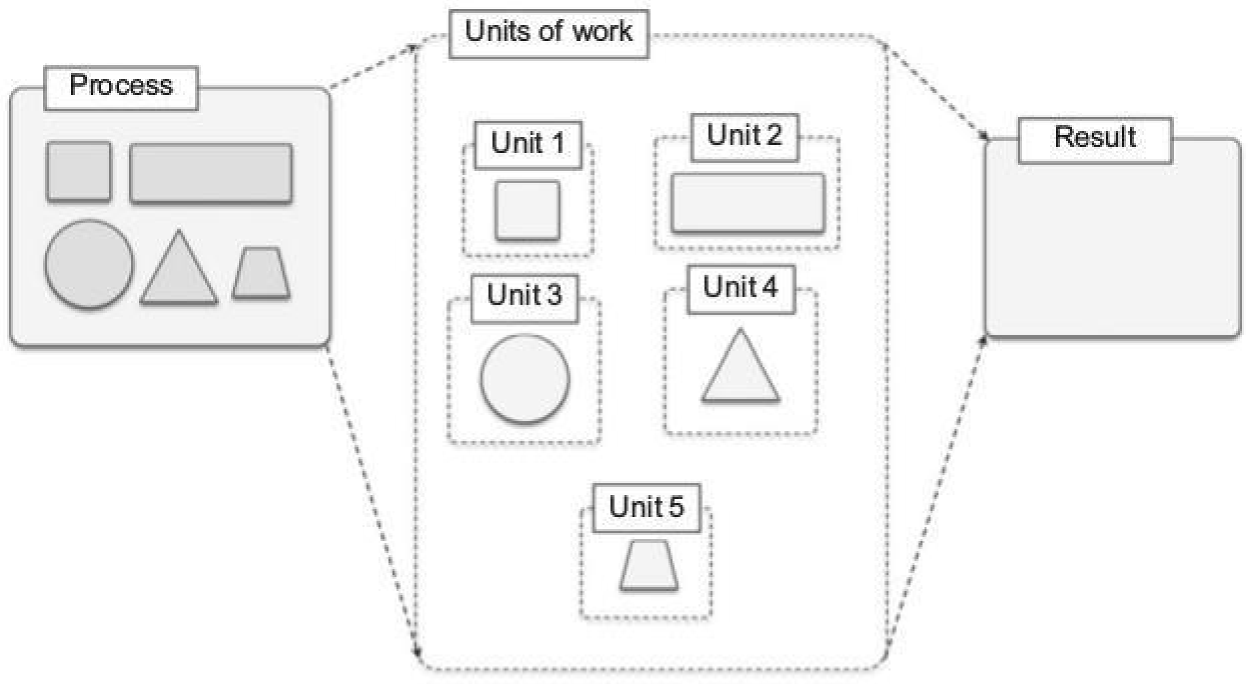


FIGURE 6.5

Functional decomposition.

Computation vs. Communication

It is very important to carefully evaluate the communication patterns among the components that have been identified during problem decomposition. The two decomposition methods presented in this section and the corresponding sample applications are based on the assumption that the computations are independent. This means that:

• The input values required by one computation do not depend on the output values generated by another computation.

• The different units of work generated as a result of the decomposition do not need to interact (i.e., exchange data) with each other.

These two assumptions strongly simplify the implementation and allow achieving a high degree of parallelism and a high throughput.

5B) Explain the major difference between Aneka threads and local threads

To efficiently run on a distributed infrastructure, Aneka threads have certain limitations compared to local threads.

These limitations relate to the communication and synchronization strategies.

1 Interface compatibility

2 Thread life cycle

3 Thread synchronization

4 Thread priorities

5 Type serialization

- The Aneka Thread Model uses the same abstraction of for defining a sequence of instructions that can be remotely executed in parallel with other instructions.
- Hence, within the Thread Model an application is a collection of remotely executable threads.
- The Thread Model allows developers to virtualize the execution of a local multi-threaded application (developed with the .NET threading APIs) in an almost complete transparent manner.

- This model represents the right solution when developers want to port the execution of a .NET multi-threaded application on Aneka and still use the same way of controlling the execution of application flow, which is based on synchronization between threads

LOCAL THREAD

- The basic control operations for local threads such as Start and Abort have a direct mapping, whereas operations that involve the temporary interruption of the thread execution have not been supported.
- The reasons for such a design decision are twofold. **Firs**t, the use of the Suspend/Resume operations is generally a deprecated practice, even for local threads, since Suspend abruptly interrupts the execution state of the thread.
- **Second**, thread suspension in a distributed environment leads to an ineffective use of the infrastructure, where resources are shared among different tenants and applications.
- Sleep operation is not supported. Therefore, there is no need to support the Interrupt operation, which forcibly resumes the thread from a waiting or a sleeping state.

6(a) List and explain various frameworks for task computing [8 marks].

- Message Passing Interface (MPI) is a specification for developing parallel programs that communicate by exchanging messages.
- Compared to other models of task computing, MPI introduces the constraint of communication that involves MPI tasks that need to run at the same time. \
- MPI has originated as an attempt to create common ground from the several distributed shared memory and message-passing infrastructures available for distributed computing.
- Nowadays, MPI has become a de facto standard for developing portable and efficient message passing HPC applications. Interface specifications have been defined and implemented for C/C11 and Fortran.
- To create an MPI application it is necessary to define the code for the MPI process that will be executed in parallel.
- This program has, in general, the structure described in Figure below. The section of code that is executed in parallel is clearly identified by two operations that set up the MPI environment and shut it down, respectively. In the code section defined within these two operations, it is possible to use all the MPI functions to send or receive messages in either asynchronous or synchronous mode.
- **In SaaS,** multiple users are provided access to the application software hosted on the server by the

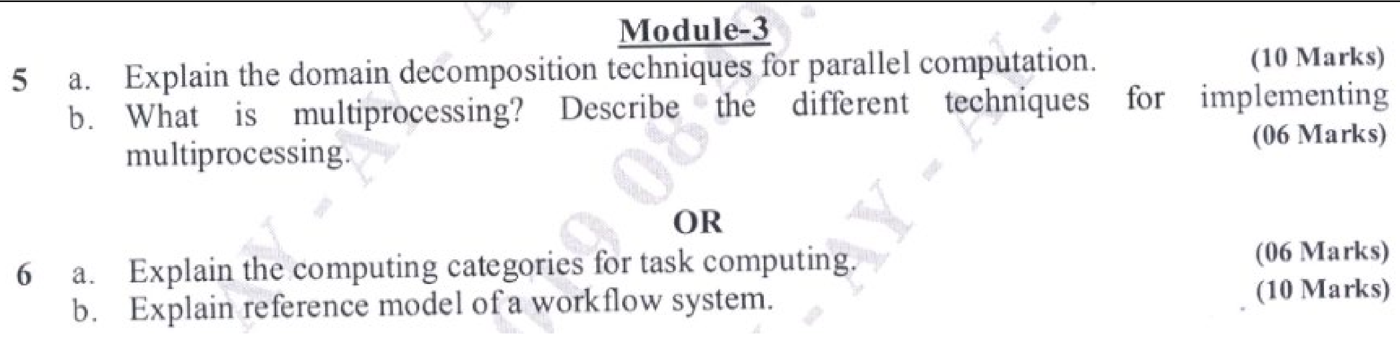
service provider. Users can access and interact with the cloud applications via the Internet, using interfaces such as web browsers, without the need to install any applications on their own systems. In Saas, software is provided as a service via the Internet and the service is priced on a pay-per-use basis. In Saas model, users do not manage or monitor the infrastructure components such as network, platform, operating system and storage devices.

- **Platform as a Service, PaaS**: The service provider delivers users a computing platform where they can develop and run their own applications using programming languages, software databases, services and tools provided by the service provider and also provides supplementary services.
- In PaaS model, users are not authorized to control or manage the servers, operating systems, storage spaces and other components that make up the platform infrastructure.
- Users' authority is limited to adjustments related to the software transferred to the cloud and configuration settings of the platform the software runs on.
- **Infrastructure as a Service, IaaS**
- In IaaS model, users can configure processing, storage, networks and other fundamental computing resources required for running applications and install the operating system and applications required.

- Users are not fully authorized to manage and control the physical infrastructure. However, users can control the system at the level of storage and operating system and manage specific network components. IaaS model is referred to as "Hardware as a service, HaaS" in some sources.

6(b) Explain the feature provided by Aneka for parameter sweep applications [8 Marks]
Aneka integrates support for parameter-sweeping applications on top of the task model by means of a collection of client components that allow developers to quickly prototype applications through either programming APIs or graphical user interfaces (GUIs). The set of abstractions and tools supporting the development of parameter sweep applications constitutes the Parameter Sweep Model (PSM).
The PSM is organized into several namespaces under the common root Aneka.PSM. More precisely:

- Aneka.PSM.Core (Aneka.PSM.Core.dll) contains the base classes for defining a template task and the client components managing the generation of tasks, given the set of parameters.
- Aneka.PSM.Workbench (Aneka.PSM.Workbench.exe) and Aneka.PSM.Wizard (Aneka.PSM. Wizard.dll) contain the user interface support for designing and monitoring parameter sweep applications. Mostly they contain the classes and components required by the Design Explorer, which is the main GUI for developing parameter sweep applications.
- Aneka.PSM.Console (Aneka.PSM.Console.exe) contains the components and classes supporting the execution of parameter sweep applications in console mode. These namespaces define the support for developing and controlling parameter sweep applications on top of Aneka.



Module-3

5 a. Explain the domain decomposition techniques for parallel computation. (10 Marks)
b. What is multiprocessing? Describe the different techniques for implementing multiprocessing. (06 Marks)

OR

6 a. Explain the computing categories for task computing. (06 Marks)
b. Explain reference model of a workflow system. (10 Marks)

5A)REPEATED

5B) WHAT IS MULTI PROCESSING DIFF TECNIQUE FOR IMPLEMENTING MULTIPROCESSING

Multiprocessing is the execution of multiple programs in a single machine, whereas multithreading relates to the possibility of multiple instruction streams within the same program.

Parallelism has been a technique for improving the performance of computers since the early 1960s. In particular, multiprocessing, which is the use of multiple processing units within a single machine, has gained a good deal of interest and gave birth to several parallel architectures.
Asymmetric multiprocessing involves the concurrent use of different processing units that are specialized to perform different functions. **Symmetric multiprocessing** features the use of similar or identical processing units to share the computation load.

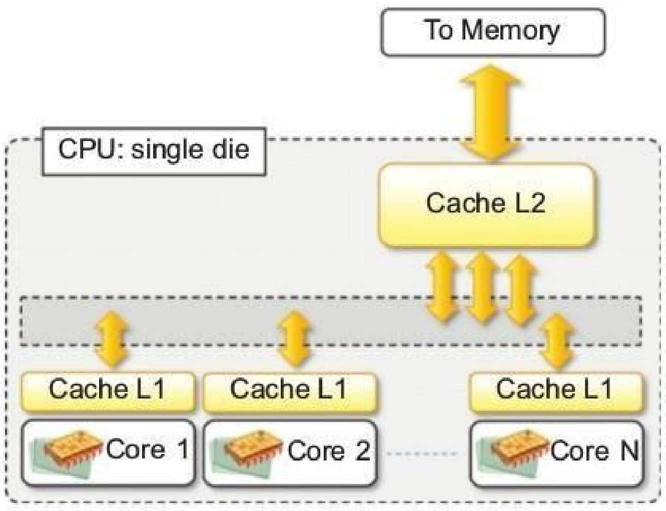


FIGURE 6.1

Multicore processor.

6A6B NOT AVAILABLE