JEPPIAAR
ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS6801 MULTI-CORE ARCHITECTURES AND PROGRAMMING

SEM:08

YEAR:04

BATCH 2015-2019

Vision of Institution

To build Jeppiaar Engineering College as an Institution of Academic Excellence in Technical education and Management education and to become a World Class University.

Mission of Institution

M1	To excel in teaching and **learning, research and innovation** by promoting the principles of scientific analysis and creative thinking
M2	To participate in the production, **development and dissemination of knowledge** and interact with **national and international communities**
M3	To equip students with **values, ethics and life skills** needed to enrich their lives and enable them to meaningfully contribute to the **progress of society**
M4	To prepare students **for higher studies and lifelong learning**, enrich them with the **practical and entrepreneurial skills** necessary to excel as future professionals and contribute to **Nation's economy**

Program Outcomes (POs)

PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Vision of Department

To emerge as a globally prominent department, developing ethical computer professionals, innovators and entrepreneurs with academic excellence through quality education and research**.**

Mission of Department

M1	To create **computer professionals** with an ability to identify and **formulate the engineering problems** and also to provide **innovative solutions** through **effective teaching learning process.**
M2	To **strengthen the core-competence** in computer science and engineering and to create an ability to **interact** effectively with industries.
M3	To produce engineers with good professional skills, **ethical values** and life skills for the **betterment of the society.**
M4	To encourage students towards **continuous and higher level learning** on technological advancements and provide a platform for **employment and self-employment.**

Program Educational Objectives (PEOs)

PEO1	**To address the real time complex engineering problems using innovative approach with strong core computing skills.**
PEO2	**To apply core-analytical knowledge and appropriate techniques and provide solutions to real time challenges of national and global society**
PEO3	**Apply ethical knowledge for professional excellence and leadership for the betterment of the society.**
PEO4	**Develop life-long learning skills needed for better employment and entrepreneurship**

Program Specific Outcomes (PSOs)

Students will be able to

PSO1	An ability to understand the core concepts of computer science and engineering and to enrich problem solving skills to analyze, design and implement software and hardware based systems of varying complexity.
PSO2	To interpret real-time problems with analytical skills and to arrive at cost effective and optimal solution using advanced tools and techniques.
PSO3	An understanding of social awareness and professional ethics with practical proficiency in the broad area of programming concepts by lifelong learning to inculcate employment and entrepreneurship skills.

BLOOM TAXANOMY LEVELS(BTL)

BTL1: Remembering
BTL 2: Understanding.,
BTL 3: Applying.,
BTL 4: Analyzing.,
BTL 5: Evaluating.,
BTL 6: Creating.,

JEPPIAAR ENGINEERING COLLEGE
DEPARTMENT OF CSE
QUESTION BANK

CS6801 MULTI-CORE ARCHITECTURES AND PROGRAMMING L T P C3 0 0 3

OBJECTIVES:

The student should be made to:

- Understand the challenges in parallel and multi-threaded programming.
- Learn about the various parallel programming paradigms, and solutions.

UNIT I MULTI-CORE PROCESSORS 9

Single core to Multi-core architectures – SIMD and MIMD systems – Interconnection networks -Symmetric and Distributed Shared Memory Architectures – Cache coherence - Performance Issues –Parallel program design.

UNIT II PARALLEL PROGRAM CHALLENGES 9

Performance – Scalability – Synchronization and data sharing – Data races – Synchronization primitives (mutexes, locks, semaphores, barriers) – deadlocks and livelocks – communication between threads (condition variables, signals, message queues and pipes).

UNIT III SHARED MEMORY PROGRAMMING WITH OpenMP 9

OpenMP Execution Model – Memory Model – OpenMP Directives – Work-sharing Constructs – Library functions – Handling Data and Functional Parallelism – Handling Loops - Performance
Considerations.

UNIT IV DISTRIBUTED MEMORY PROGRAMMING WITH MPI 9

MPI program execution – MPI constructs – libraries – MPI send and receive – Point-to-point and Collective communication – MPI derived datatypes – Performance evaluation

UNIT V PARALLEL PROGRAM DEVELOPMENT 9

Case studies - n-Body solvers – Tree Search – OpenMP and MPI implementations and comparison.

TOTAL: 45 PERIODS

OUTCOMES:

At the end of the course, the student should be able to:

- Program Parallel Processors.
- Develop programs using OpenMP and MPI.
- Compare and contrast programming for serial processors and programming for parallel processors.

TEXT BOOKS:
1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011.
2. Darryl Gove, “Multicore Application Programming for Windows, Linux, and Oracle Solaris”, Pearson, 2011 (unit 2)

REFERENCES:
1. Michael J Quinn, “Parallel programming in C with MPI and OpenMP”, Tata McGraw Hill

COURSE OUTCOME

C409.1	**Illustrate** the challenges in parallel and multi threaded programming
C409.2	**Explain** the various parallel programming paradigms and solutions**.**
C409.3	**Develop** shared memory programs using OpenMP
C409.4	**Develop** Distributed memory programs using MPI
C409.5	**Compare** and contrast programming for serial processors and parallel processors.

S.No	**UNIT**	REF.BOOK	**PAGE.NO**
1	**UNIT I**	1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011.	**1-8**
2	**UNIT II**	2. Darryl Gove, “Multicore Application Programming for Windows, Linux, and Oracle Solaris”, Pearson, 2011 (unit 2)	**8-14**
3	**UNIT III**	1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011.	**14-22**
4	**UNIT IV**	1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011.	**22-28**
5	**UNIT V**	1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011.	**28-36**

UNIT –I

MULTI-CORE PROCESSORS

Single core to Multi-core architectures – SIMD and MIMD systems – Interconnection networks - Symmetric and Distributed Shared Memory Architectures – Cache coherence - Performance Issues –Parallel program design.

<table>
<tr><th>Q. No.</th><th>Questions</th><th>CO</th><th>Bloom's Level</th></tr>
<tr><td>1.</td><td>What is Single core processors?
Single core processors have only one processor in die to process instructions.</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>2.</td><td>What are the Problems of Single Core Processors:
As we try to increase the clock speed of this processor , the amount of heat produced by the chip also increases. It is a big hindrance in the way of single core processors to continue evolving</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>3.</td><td>What is Multicore processor?
A multi-core processor is a single computing component with two or more independent actual processing units (called "cores"), which are units that read and execute program instructions. The multiple cores are embedded in the same die. The multicore processor may looks like a single processor but actuall y it contains two (dual - core), three (tri - core), four (quad - core), six(hexa-core), eight(octa-core)or ten (deca-core) cores.Some processor even have 22 or 32 cores..</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>4.</td><td>What are the Problems with multicore processors.
According to Amdahl's law , the performance of parallel computing is limited by its serial components. So, increasing the number of cores may not be the best solution .There is need to increase the clock speed of individual cores.</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>5.</td><td>Comparison Of Single-Core processor And Multi-Core Processor.
<table><tr><th>Parameter</th><th>Single-Core Processor</th><th>Multi-Core Processor</th></tr><tr><td>Number of cores on a die</td><td>Single</td><td>Multiple</td></tr><tr><td>Instruction Execution</td><td>Can execute Single instruction at a time</td><td>Can execute multiple instructions by using multiple cores</td></tr></table></td><td>C409.1</td><td>BTL1</td></tr>
</table>

	<table><tr><td>Gain</td><td>Speed up every program or software being executed</td><td>Speed up the programs which are designed for multi-core processors</td></tr><tr><td>Performance</td><td>Dependent on the clock frequency of the core</td><td>Dependent on the frequency, number of cores and program to be executed</td></tr><tr><td>Examples</td><td>Processor launched before 2005 like 80386,486, AMD 29000, AMD K6, Pentium I,II,III etc.</td><td>Processor launched after 2005 like Core -2-Duo,Athlon 64 X2,I3,I5 and I7 etc</td></tr></table>		
6	**What is meant by Single instruction, multiple data (SIMD)** **Single instruction, multiple data** (**SIMD**), is a class of parallel computers in Flynn's taxonomy. It describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously. Thus, such machines exploit data level parallelism, but not concurrency: there are simultaneous (parallel) computations, but only a single process (instruction) at a given moment	C409.1	BTL1
7	**What is meant by multiple instruction, multiple data (MIMD)** In computing, **MIMD** (multiple instruction, multiple data) is a technique employed to achieve parallelism. Machines using MIMD have a number of processors that function asynchronously and independently. At any time, different processors may be executing different instructions on different pieces of data. MIMD architectures may be used in a number of application areas such as computer-aided design/computer-aided manufacturing, simulation, modeling, and as communication switches. MIMD machines can be of either shared memory or distributed memory categories.	C409.1	BTL1
8	**Define Multistage interconnection networks.** Multistage **interconnection networks** (MINs) are a class of high-speed computer **networks** usually composed of processing elements (PEs) on one end of the **network** and memory elements (MEs) on the other end, connected by switching elements (SEs).	C409.1	BTL1
9	**What is meant by Routing?** -How does a message get from source to destination. -Static or adaptive	C409.1	BTL1

10	**What is Network interface?** -Connects endpoints (e.g. cores) to network. -Decouples computation/communication	C409.1	BTL1
11	**.What is Centralized shared-memory multiprocessor** It share a single centralized memory, interconnect processors and memory by a bus • also known as "uniform memory access" (UMA) or "symmetric (shared-memory) multiprocessor" (SMP) – A symmetric relationship to all processors. – A uniform memory access time from any processor.	C409.1	BTL1
12	**What is the concept of Caching in shared-memory machines.** • private data: used by a single processor – When a private item is cached, its location is ***migrated*** to the cache – Since no other processor uses the data, the program behavior is identical to that in a uniprocessor • shared data: used by multiple processor – When shared data are cached, the shared value may be ***replicated*** in multiple caches.	C409.1	BTL1
13	**What is Cache Coherence** . • migration: a data item can be moved to a local cache and used there in a transparent fashion • replication for shared data that are being simultaneously read both are critical to performance in accessing shared data.	C409.1	BTL1
14	**What is meant by Snooping Solution (Snoopy Bus).** – Send all requests for data to all processors – Processors snoop to see if they have a copy and respond accordingly – Requires broadcast, since caching information is at processors – Works well with bus (natural broadcast medium) – Dominates for small scale machines (most of the market)	C409.1	BTL1
15	**What is meant by Directory-Based Schemes.** – Directory keeps track of what is being shared in a centralized place (logically) – Distributed memory => distributed directory for scalability (avoids bottlenecks) – Send point-to-point requests to processors via network – Scales better than Snooping – Actually existed BEFORE Snooping-based schemes	C409.1	BTL1
16	**When a memory system is coherent ?** A memory system is coherent if: • P writes to X; no other processor writes to X; P reads X and receives the value previously written by P • P1 writes to X; no other processor writes to X; sufficient time lapses; P2 reads X and receives value written by P1 • Two writes to the same location by two processors are seen in the	C409.1	BTL1

<table>
<tr><td></td><td>same order by all processors – write serialization
• The memory consistency model defines “time elapsed” before the effect of a processor is seen by others.</td><td></td><td></td></tr>
<tr><td>17</td><td>What is meant by a distributed-memory system?

A distributed-memory system (often called a multicomputer) consist of multiple independent processing nodes with local memory modules which is connected by a general interconnection network. Software DSM systems can be implemented in an operating system, or as a programming library and can be thought of as extensions of the underlying virtual memory architecture.</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>18</td><td>What the difference between Message passing vs. DSM

<table>
<tr><th>Message passing</th><th>Distributed shared memory</th></tr>
<tr><td>Variables have to be marshalled</td><td>Variables are shared directly</td></tr>
<tr><td>Cost of communication is obvious</td><td>Cost of communication is invisible</td></tr>
<tr><td>Processes are protected by having private address space</td><td>Processes could cause error by altering data</td></tr>
<tr><td>Processes should execute at the same time</td><td>Executing the processes may happen with non-overlapping lifetimes</td></tr>
</table></td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>19</td><td>What are the Advantages of DSM. (Apr/May 2018)

System scalable
Hides the message passing
Can handle complex and large data bases without replication or sending the data to processes
DSM is usually cheaper than using multiprocessor system
No memory access bottleneck, as no single bus
DSM provides large virtual memory space
DSM programs portable as they use common DSM programming interface
Shields programmer from sending or receive primitives

DSM can (possibly) improve performance by speeding up data access</td><td>C409.1</td><td>BTL1</td></tr>
</table>

<table>
<tr><td>20</td><td>What the Issues in implementing DSM software
• Data is replicated or cached
• Reduce delays
• Semantics for concurrent access must be clearly specified
• DSM is controlled by memory management software, operating system, language run-time system
• Locating remote data
• Granularity: how much data is transferred in a single operation</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>21</td><td>What are the Disadvantages of DSM (Apr/May 2018)
• Could cause a performance penalty
• Should provide for protection against simultaneous access to shared data such as lock
• Performance of irregular problems could be difficult</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>22</td><td>What are the Methods of achieving DSM.
There are usually two methods of achieving distributed shared memory:
• hardware, such as cache coherence circuits and network interfaces;
• software.We can use this method in different ways such as modifying the operating system kernel.</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>23</td><td>What is meant by Consistency models..
Memory system tries to behave based on certain rules in the system, which is called system's consistency model.</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>24</td><td>Define Vector Instruction?(Apr/May2017)
A vector processor or array processor is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors, compared to scalar processors, whose instructions operate on single data items.</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>25</td><td>What is meant by Snooping cache coherence? (Apr/May 2017)
Also referred to as a bus-snooping protocol, a protocol for maintaining cache coherency in symmetric multiprocessing environments. In a snooping system, all caches on the bus monitor (or snoop) the bus to determine if they have a copy of the block of data that is requested on the bus.</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>26</td><td>Compare Symmetric memory architecture and distributed memory architecture. (Nov/Dec 2017)
<table><tr><td>Sno</td><td>Symmetric memory architecture</td><td>distributed memory architecture.</td></tr></table></td><td>C409.1</td><td>BTL1</td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td>1</td><td>sharedmemory cache-coherent multiprocessor systems. The systems communicated with each other and with shared main memory over a shared bus.</td><td>distributed memory refers to a multiprocessor computer system in which each processor has its own private memory. Computational tasks can only operate on local data, and if remote data is required, the computational task must communicate with one or more remote processors</td></tr><tr><td>2</td><td>any access from any processor to main memory would have equal latency</td><td>any access from any processor to main memory would have different latency</td></tr></table></td><td></td><td></td></tr>
<tr><td>27</td><td>What are multiprocessor systems and give their advantages? (Nov/Dec 2017)

Multiprocessor systems also known as parallel systems or tightly coupled systems are systems that have more than one processor in close communication, sharing the computer bus, the clock and sometimes memory & peripheral devices.
Their main advantages are
1Increased throughput
2 Economy of scale
3 Increased reliability</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>28</td><td>Define Channel?
A single logical connection between routers/switches</td><td>C409.1</td><td>BTL1</td></tr>
<tr><td>29</td><td>List the pros and cons of distributed system (Apr/May 2018)
System scalable
Hides the message passing
Can handle complex and large data bases without replication or sending the data to processes
DSM is usually cheaper than using multiprocessor system
No memory access bottleneck, as no single bus
DSM provides large virtual memory space
DSM programs portable as they use common DSM programming interface
Shields programmer from sending or receive primitives
DSM can (possibly) improve performance by speeding up data access
Could cause a performance penalty
Should provide for protection against simultaneous access to shared data such as lock</td><td>C409.1</td><td>BTL1</td></tr>
</table>

	Performance of irregular problems could be difficult		
30	**Define the symmetric shared memory (Apr/May 2018, Nov/Dec 2018)** **Symmetric Shared Memory Architecture** consists of several processors with a single physical **memory shared** by all processors through a **shared** bus	C409.1	BTL1
31	**List out the advantages of multicore CPU** • The largest boost in performance will likely be noticed in improved response time while running CPU intensive processes, like anti-virus scans, ripping/burning media. • Assuming that the die can fit into the package, physically, the multi-core CPU designs require much less printed Circuit Board(PCB) space than multichip SMP designs. • Also, a dual core processor uses slightly less power than two coupled single core processors, principally because of the decreased power required to drive signals external to the chip	C409.1	BTL1
32	**Define Vector Registers** These are registers capable of storing a vector of operands and operating simultaneously on their contents. The vector length is fixed by the system, and can range from 4 to 128 64-bit elements. Vectorized and pipelined functional units.	C409.1	BTL1
33	**Define Latency and Bandwidth** • The latency is the time that elapses between the source's beginning to transmit the data and the destination's starting to receive the first byte. • The bandwidth is the rate at which the destination receives data after it has started to receive the first byte. So if the latency of an interconnect is l seconds and the bandwidth is b bytes per second, then the time it takes to transmit a message of n bytes is • **Message transmission time= l+n/b**	C409.1	BTL1
34	**List out the approaches in cache coherence** • Snooping cache coherence • Directory-based cache coherence.	C409.1	BTL1
35	**List the steps involved in Parallel Program design** 1.Partitioning	C409.1	BTL1

	2.Aggregating 3.Communication 4.Mapping		
36	**Define Directory based cache coherence** It is protocols attempt to solve this problem through the use of a data structure called a **directory**. The directory stores the status of each cache line. Typically, this data structure is distributed; in our example, each core/memory pair might be responsible for storing the part of the structure that specifies the status of the cache lines in its local memory	C409.1	BTL1
37	**Define Parallel Overhead** **Tparallel = Tserial/p + Toverhead.**	C409.1	BTL1
38	**Define Scalability** The number of processes/threads that are used by the program. If we can find a corresponding rate of increase in the problem size so that the program always has efficiency E, then the program is scalable.	C409.1	BTL1
39	**Define Amdhal's Law (Nov/Dec 2018)** Amdahl made an observation that's become known as ***Amdahl's law***. It says, roughly, that unless virtually all of a serial program is parallelized, the possible speedup is going to be very limited—regardless of the number of cores available.	C409.1	BTL1
40	**Define Speedup and Efficiency** The Serial run-time *T*serial and our parallel run-time *T*parallel, then the best we can hope for is *T*parallel = *T*serial/*p*. Parallel program has **linear speedup**. So if we define the **speedup** of a parallel program to be linear speedup has $S = p$, which is unusual. Furthermore, as *p* increases,we expect *S* to become a smaller and smaller fraction of the ideal, linear speedup *p*.	C409.1	BTL1

41	**How to parallelize the serial program** • For the first step we might identify two types of tasks: finding the bin to which an element of data belongs and incrementing the appropriate entry in bin counts. • For the second step, there must be a communication between the computation of the appropriate bin and incrementing an element of bin counts.	C409.1	BTL1
42	**List out the different distributed memory interconnects** Distributed-memory interconnects are often divided into two groups: **Direct interconnects** and **Indirect interconnects**	C409.1	BTL1
43	**Define Direct Interconnects** In a direct interconnect each switch is directly connected to processor memory pair, and the switches are connected to each other. As before, the ***circles*** are ***switches***, the ***squares*** are ***processors***, and the ***lines*** are ***bidirectional links***. A ***ring*** is superior to a simple bus since it allows multiple simultaneous communications.	C409.1	BTL1
44	**Define Indirect Interconnects** • They provide an alternative to direct interconnects. In an indirect interconnect the switches may not be directly connected to a processor. • They're often shown with unidirectional links and a collection of processors, each of which has an outgoing and an incoming link, and a switching network.	C409.1	BTL1
45	**Define Ideal Direct interconnect** The ideal direct interconnect is a ***fully connected network*** in which each switch is directly connected to every other switch	C409.1	BTL1
46	**Define Hypercube** • It is a highly connected direct interconnect that has been used in actual systems. Hypercubes are built inductively: • A one-dimensional hypercube is a fullyconnected system with two processors. • A two-dimensional hypercube is built from two one-dimensional hypercubes by joining "corresponding" switches	C409.1	BTL1
47	**Define Interleaved memory** The memory system consists of multiple "banks" of memory, which can be accessed more or less independently. After accessing one bank, there will be	C409.1	BTL1

	a delay before it can be reassessed, but a different bank can be accessed much sooner. So if the elements of a vector are distributed across multiple banks, there can be little to no delay in loading/storing successive elements.		
48	**Define Strided memory** In strided memory access, the program accesses elements of a vector located at fixed intervals.For example, accessing the first element, the fifth element, the ninth element, and so on, would be strided access with a stride of four	C409.1	BTL1
49	**Define Graphics Processor Pipeline** Real-time graphics application programming interfaces, or APIs, use points,lines,and triangles to internally represent the surface of an object. They use a **graphics processing pipeline** to convert the internal representation into an array of pixels that can be sent to a computer screen	C409.1	BTL1
50	**List out the two principal types of MIMD system** • Shared Memory System • Distributed Memory system	C409.1	BTL1

PART B

Q. No.	Questions	CO	Bloom's Level
1	**Explain Single core to Multi-core Architectures .** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:15-20	C409.1	BTL5
2	**Explain SIMD and MIMD systems (Apr/May2017,Nov/Dec 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:29-34	C409.1	BTL5
3	**Explain about Interconnection networks? (Apr/May 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:35-44	C409.1	BTL5
4	**Explain with neat diagram Symmetric Shared Memory Architectures** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:35-42	C409.1	BTL5

5	**Explain with neat diagram Distributed Shared Memory Architectures (Nov/Dec 2018)** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011 Page No:43-45	C409.1	BTL5
6	**Explain Cache coherence in Symmetric Shared and Distributed Shared Memory Architectures** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011 Page No: 50-55	C409.1	BTL5
7	**Explain the performance issues of multicore processor. .(Nov/Dec 2017)** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011 Page No:58-64	C409.1	BTL5
8	**Define cache coherence problem. What are the 2 main approaches to cache coherence? Describe working of snooping cache coherence and explain describe directory based coherence. (Nov/Dec 2017)** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011 Page No:43-45	C409.1	BTL5
9	**Explain parallel program design (Apr/May 2017,Nov/Dec 2018)** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011 Page No:65-69	C409.1	BTL5
10	**State and explain Amdahl’s law Outline the steps in designing and building parallel program. Give example (Apr/May 2018)** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011	C409.1	BTL5
11	**Elaborate the classification of computer architecture in parallel computing system (Apr/May 2018)** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011	C409.1	BTL5
12	**Explain Directory Based cache coherence protocol** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011	C409.1	BTL4

13	**Generalize the snooping protocol briefly** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011	C409.1	BTL6
14	**Summarize the Parallelizing the serial program** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011	C409.1	BTL5
15	**Explain the Shared memory interconnect** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011	C409.1	BTL3
16	**Highlight the limitations of single core processors and outline how multicore architecture overcome the limitations** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011	C409.1	BTL3

UNIT II
PARALLEL PROGRAM CHALLENGES

Performance – Scalability – Synchronization and data sharing – Data races – Synchronization primitives (mutexes, locks, semaphores, barriers) – deadlocks and livelocks – communication between threads (condition variables, signals, message queues and pipes).

PART A

Q. No.	Questions	CO	Bloom's Level
1	**What is data race?** A data race occurs when multiple threads use the same data item and one or more of those threads are updating it.	C409.2	BTL1
2	**How to avoid data races .** One way to avoid data race is by utilizing proper synchronization between threads.	C409.2	BTL1
3	**Hoe to Avoid Data Races.** Although it can be hard to identify data races, avoiding them can be very simple: Make sure that only one thread can update the variable at a time. The easiest way to do this is to place a *synchronization lock* around all accesses to that variable and ensure that before referencing the variable, the thread must acquire the lock.	C409.2	BTL1
4	**What is the use of Synchronization Primitives? List out the various synchronization primitive in parallel programming (Nov/Dec 2018)** Synchronization is used to coordinate the activity of multiple threads. There are various situations where it is necessary; this might be to ensure that shared resources are not accessed by multiple threads simultaneously or that all work on those resources is complete before new work starts. Process Synchronization Memory Synchronization Thread Synchronization	C409.2	BTL1
5	**What is the simplest form of synchronization?** The simplest form of synchronization is a mutually exclusive (*mutex*) lock. Only one thread at a time can acquire a mutex lock, so they can be placed around a data structure to ensure that the data structure is modified by only one thread at a time.	C409.2	BTL1
6	**How to Place Mutex Locks Around Accesses to Variables.** int counter; mutex_lock mutex;	C409.2	BTL1

	void Increment()
	{
	acquire(&mutex);
	counter++;
	release(&mutex);}
	void Decrement()
	{
	acquire(&mutex);
	counter--;
	release(&mutex);}		

7	**What is meant by *contended* mutex**. If multiple threads are attempting to acquire the same mutex at the same time, then only one thread will succeed, and the other threads will have to wait. This situation is known as a *contended* mutex.	C409.2	BTL1
8	**What is critical region.** The region of code between the acquisition and release of a mutex lock is called a *critical section*, or *critical region*. Code in this region will be executed by only one thread at a time.	C409.2	BTL1

9	**Develop the code for Placing a Mutex Lock Around a Region of Code**
	
	void * threadSafeMalloc(size_t size)
	{
	acquire(&mallocMutex);
	void * memory = malloc(size);
	release(&mallocMutex);
	return memory;
	}
		C409.2	BTL6

10	**What is meant by Spin Locks.** *Spin locks* are essentially mutex locks. The thread waiting to acquire a spin lock will keep trying to acquire the lock without sleeping	C409.2	BTL1
11	**Compare spin lock and mutex lock. (Apr/May 2018)** The difference between a mutex lock and a spin lock is that a thread waiting to acquire a spin lock will keep trying to acquire the lock without sleeping .In comparison; a mutex lock may sleep if it is unable to acquire the lock.	C409.2	BTL2
12	**Write the advantage of spin locks.** The advantage of using spin locks is that they will acquire the lock as soon as it is released, whereas a mutex lock will need to be woken by the operating system before it can get the lock	C409.2	BTL1
13	**What is the disadvantage of spin locks.** The disadvantage is that a spin lock will spin on a virtual CPU monopolizing that resource. In comparison, a mutex lock will sleep and free the virtual CPU for another thread to use.	C409.2	BTL1

14	**What is Semaphores?** *Semaphores* are counters that can be either incremented or decremented. They can be used in situations where there is a finite limit to a resource and a mechanism is needed to impose that limit. Semaphores will also signal or wake up threads that are waiting on them to use available resources	C409.2	BTL1
15	**What is an example for semaphore.** An example might be a buffer that has a fixed size. Every time an element is added to a buffer, the number of available positions is decreased. Every time an element is removed, the number available is increased	C409.2	BTL1
16	**What is wait and release in semaphore.** the method that acquires a semaphore might be called *wait*, *down*, or *acquire*, and the method to release a semaphore might be called *post*,*up*, *signal*, or *release*. When the semaphore no longer has resources available, the threads requesting resources will block until resources are available.	C409.2	BTL1
17	**Define *readerswriter lock*.** A *readerswriter lock* (or *multiple-reader lock*) allows many threads to read the shared data but can then lock the readers threads out to allow one thread to acquire a writer lock to modify the data.	C409.2	BTL1

18	**Show an example for Readers-Writer Lock.**
	int readData(int cell1, int cell2)
	{
	acquireReaderLock(&lock);
	int result = data[cell] + data[cell2];
	releaseReaderLock(&lock);
	return result;
	}
	void writeData(int cell1, int cell2, int value)
	{
	acquireWriterLock(&lock);
	data[cell1] += value;
	data[cell2] -= value;
	releaseWriterLock(&lock);
	}
		C409.2	BTL1

19	**What are the use of Barriers**. There are situations where a number of threads have to all complete their work before any of the threads can start on the next task. In these situations, it is useful to have a barrier where the threads will wait until all are present	C409.2	BTL1
20	**Show One common example of using a barrier.** One common example of using a barrier arises when there is a dependence between different sections of code. For example, suppose a number of threads compute the values stored in a matrix. The variable total needs to be calculated using the values stored in the matrix. A barrier can be used to ensure that all the threads complete their computation of the matrix	C409.2	BTL1

<table>
<tr><td></td><td>before the variable total is calculated .ZThe following example shows a situation using a barrier to separate the calculation of a variable from its use.

Compute_values_held_in_matrix();
Barrier();
total = Calculate_value_from_matrix();</td><td></td><td></td></tr>
<tr><td>21</td><td>What is data sharing. (Apr/May 2017)
Sharing data between multiple threads is called data sharing.</td><td>C409.2</td><td>BTL1</td></tr>
<tr><td>22</td><td>What are the difference between deadlock and livelock. (Apr/May 2017)(Nov/Dec 2018)
The deadlock occurs where two or more threads cannot make progress because the resources that they needare held by the other threads.
Example:Suppose two threads need to acquire mutex locks A and B to complete some task. If thread 1 has already acquired lock A and thread 2 has already acquired lock B, then A cannot make forward progress because it is waiting for lock B, and thread 2 cannot make progress because it is waiting for lock A. The two threads are deadlocked.
Listing 4.13 Two Threads in a Deadlock
Thread 1<pre>
void update1()
{
   acquire(A);
   acquire(B); <<< Thread 1
                   waits here
   variable1++;
   release(B);
   release(A);
}
</pre>Thread 2<pre>
void update2()
{
   acquire(B);
   acquire(A); <<< Thread 2
                   waits here
   variable1++;
   release(B);
   release(A);
}
</pre></td><td>C409.2</td><td>BTL1</td></tr>
<tr><td>23</td><td>What are conditions under which a deadlock situation may arise? (Nov/Dec 2017)
Mutual Exclusion: At least one resource is held in a non-sharable mode that is only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
Hold and Wait:There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.
No Preemption: Resouces cannot be preempted; that is, a resource can only be released voluntarily by the process holding it, after the process has completed its task.
Circular Wait: There must exist a set {p_0, p_1,.....p_n} of waiting processes such that p_0 is waiting for a resource which is held by p_1, p_1 is waiting for a resource which is held by p_2,..., p_{n-1} is waiting for a resource which is held by p_n and p_n is waiting for a resource which is held by p_0.</td><td>C409.2</td><td>BTL1</td></tr>
</table>

24	**Define thread. mention the uses of swapping. (Nov/Dec 2017)** A thread is the smallest unit of processing that can be performed in an OS. In most modern operating systems, a thread exists within a process - that is, a single process may contain multiple threads	C409.2	BTL1
25	**Define deadlock.** *Deadlock* is the situation where two or more threads cannot make progress because the resources that they need are held by the other threads. It is easiest to explain this with an example. Suppose two threads need to acquire mutex locks A and B to complete some task. If thread 1 has already acquired lock A and thread 2 has already acquired lock B, then A cannot make forward progress because it is waiting for lock B, and thread 2 cannot make progress because it is waiting for lock A. The two threads are *deadlocked*	C409.2	BTL1
26	**How to communicate multiple threads.** The easiest way for multiple threads to communicate is through memory. If two threads can access the same memory location, the cost of that access is little more than the memory latency of the system. Of course, memory accesses still need to be controlled to ensure that only one thread writes to the same memory location at a time. A multithreaded application will share memory between the threads by default, so this can be a very low-cost approach. The only things that are not shared between threads are variables on the stack of each thread (local variables) and thread-local variables.	C409.2	BTL1
27	**Illustrate an example which Use Multiple Barriers.** Compute_values_held_in_matrisx(); **Barrier();** total = Calculate_value_from_matrix(); **Barrier();** Perform_next_calculation(total);	C409.2	BTL2
28	**Define live lock.** A *livelock* traps threads in an unending loop releasing and acquiring locks. Livelocks can be caused by code to back out of deadlocks.	C409.2	BTL1
29.	**What is An *atomic operation*** An *atomic operation* is one that will either successfully complete or fail; it is not possible for the operation to either result in a “bad” value or allow other threads on the system to observe a transient value.	C409.2	BTL1
30	**Write down the performance metrics (Apr/May 2018)**	C409.2	BTL1

31	**Define Message Queue** A message queue is a structure that can be shared between multiple processes. Messages can be placed into the queue and will be removed in the same order in which they were added. Constructing a message queue looks rather like constructing a shared memory segment.	C409.2	BTL1
32	**Define Named Pipes** UNIX uses pipes to pass data from one process to another. For example, the output from the command ls, which lists all the files in a directory, could be piped into the wc command, which counts the number of lines, words, and characters in the input	C409.2	BTL1
33	**Mention the mechanism associated with Named Pipes** Setting Up and Writing into a Pipe Make Pipe(Descriptor); ID = Open Pipe(Descriptor); Write Pipe(ID, Message, sizeof(Message)); ... Close Pipe(ID); Delete Pipe(Descriptor); Opening an Existing Pipe to Receive Messages ID=Open Pipe(Descriptor); Read Pipe(ID, buffer, sizeof(buffer)); Close Pipe(ID);	C409.2	BTL1
34	**How to create and Place Message Queues** Creating and Placing Messages into a Queue ID = Open Message Queue Queue(Descriptor); Put Message in Queue(ID, Message); Close Message Queue(ID); Delete Message Queue(Description); Using the descriptor for an existing message queue enables two processes to communicate by sending and receiving messages through the queue. Opening a Queue and Receiving Messages ID=Open Message Queue ID(Descriptor); Message=Remove Message from Queue(ID); ... Close Message Queue(ID);	C409.2	BTL1
35	**What is the fundamental way to share access to resources between threads** • Deadlock • Livelock	C409.2	BTL1
36	**Give an example of critical regions** An operating system does not have an implementation of malloc() that is thread-safe, or safe for multiple threads to call at the same time. One way to fix this is to place the call to malloc() in a critical section by surrounding it with a mutex lock	C409.2	BTL1
37	**List out the issues in shared caches** • Capacity misses • Conflict misses	C409.2	BTL1
38	**Define False sharing** ***False sharing*** is the situation where multiple threads are accessing items of data held on a single cache line. Although the threads are all using separate items of data, the cache line itself is shared between them so only a single thread can write to it at any one time.	C409.2	BTL1
39	**List out the Memory Bandwidth Measured on a System with Four Virtual CPUs**	C409.2	BTL1

	Threads Time 7.437563 s Bandwidth 2.63 GB/s Threads Time 15.238317 s Bandwidth 2.57 GB/s Threads Time 24.580981 s Bandwidth 2.39 GB/s Threads Time 37.457352 s Bandwidth 2.09 GB/s		
40	**What are the measures to be taken when bandwidth size reduces** The threads are interfering on the processor. • A second interaction effect is if the threads start interfering in the caches, such as multiple threads attempting to load data to the same set of cache lines. • One other effect is the behavior of memory chips when they become saturated. The chips start experiencing queuing latencies where the response time for each request increases. Memory chips are arranged in banks. • Accessing a particular address will lead to a request to a particular bank of memory. Each bank needs a gap between returning two responses. If multiple threads happen to hit the same bank, then the response time becomes governed by the rate at which the bank can return memory	C409.2	BTL1

41	**Write a code to measure memory bandwidth using Memset**
	#include <stdio.h>
	#include <stdlib.h>
	#include <strings.h>
	#include <pthread.h>
	#include <sys/time.h>
	#define BLOCKSIZE 1024*1025
	int nthreads = 8;
	char * memory;
	double now( )
	{
	struct timeval time;
	gettimeofday(&time, 0);
	return (double)time.tv_sec + (double)time.tv_usec / 1000000.0;
	}
	void *experiment(void *id)
	{
	unsigned int seed = 0;
	int count = 20000;
	for(int i=0; i<count; i++)
	{
	memset(&memory[BLOCKSIZE * (int)id], 0, BLOCKSIZE);
	}
		C409.2	BTL1

42	**How Bandwidth share between cores** Bandwidth is another resource shared between threads. The bandwidth capacity of a system depends on the design of the processor and the memory system as well as the memory chips and their location in the system	C409.2	BTL1

43	**List out the three critical areas to address large difference scaling** • The amount of bandwidth to cache and the memory will be divided among the active threads on the system. • The design of the caches will determine how much time is lost because of capacity and conflict-induced cache misses. • The way that the processor core pipelines are shared between active software threads will determine how instruction issue rates change as the number of active threads increases.	C409.2	BTL1
44	**What is the role default malloc( )** The default malloc() provides better performance than the alternative implementation. The algorithm that provides improved scaling also adds a cost to the single-threaded situation; it can be hard to produce an algorithm that is fast for the single- threaded case and scales well with multiple threads.	C409.2	BTL1
45	**Define an idea to choose the appropriate data structures** Choosing the best structure to hold data, such as choosing an algorithm of the appropriate complexity, can have a major impact on overall performance. Some structures will be efficient when data is accessed in one pattern, while other structures will be more efficient if the access pattern is changed.	C409.2	BTL1
46	**Define Column major order** The opposite ordering is followed, so adjacent elements of the first index are adjacent in memory. This is called *column-major* order. Accessing elements by a stride is a common error in codes translated from Fortran into C. It shows how memory is addressed in C, where adjacent elements in a row are adjacent in memory.	C409.2	BTL1
47	**How to select the appropriate array access pattern** One common data access pattern is striding through elements of an array. The performance of the application would be better if the array could be arranged so that the selected elements were contiguous.	C409.2	BTL1
48	**List out the techniques to reduce the latency** • Out of order execution • Hardware prefetching • Software prefetching	C409.2	BTL1
49	**List out the non technical reasons why functionality get placed in libraries** • Libraries often represent a convenient product for an organizational unit. One group of developers might be responsible for a particular library of code, but that does not automatically imply that a single library represents the best way for that code to be delivered to the end users. • Libraries are also used to group related functionality. For example, an application might contain a library of string-handling functions. Such a library might be appropriate if it contains a large body of code. On the other hand, if it contains only a few small routines, it might be more appropriate to combine it with another library.	C409.2	BTL1

50	**Why Algorithm complexity is important** Algorithmic complexity represents the expected performance of a section of code as the number of elements being processed increases. In the limit, the code with the greatest algorithmic complexity will dominate the runtime of the application	C409.2	BTL1

.

PART B

Q. No.	Questions	CO	Bloom's Level
1.	**Explain about Synchronization and data sharing in detail.** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:121-126	C409.2	BTL5
2.	**Explain Synchronization primitives mutexes and locks.** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:126-128	C409.2	BTL5
3.	**Explain Synchronization primitives in semaphores and barriers in detail**. 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:128-129	C409.2	BTL5
4.	**Explain the concepts of deadlocks and live locks** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:132-133	C409.2	BTL5
5.	**Explain communication between threads using condition variables and signals.** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:133-139	C409.2	BTL5
6.	**Explain communication between threads using message queues and pipes.** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:138-139	C409.2	BTL5
7.	**Explain data races and scalability in parallel program. (apr/may2017)** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:121-126	C409.2	BTL5

8.	**Explain Synchronization primitives in parallel program challenges. (Apr/may2017)** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:126-130	C409.2	BTL5
9	**Explain the various approaches to parallel programming. .(Nov/Dec 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:43-47	C409.2	BTL5
10.	**What is a data race? What are the tools used for detecting data races? How to avoid races? (Nov/Dec 2017) (Apr/May 2018)** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011 Pg.No:121-125	C409.2	BTL5
11	**(i)Discuss in detail about producer consumer synchronization (ii)Write a simple semaphore to send a message (Apr/May 2018)** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011	C409.2	BTL5
12	**Write a short notes on deadlocks, livelocks and named pipes** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011	C409.2	BTL5
13	**Discuss in detail about the importance of algorithmic complexity** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011	C409.2	BTL2
14	**Explain the outline about necessity of structure reflects in performance** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011	C409.2	BTL4
15	**Write in detail and summarize about hardware constraints applicable in improving scaling** 2. Darryl Gove, "Multicore Application Programming for Windows, Linux, and Oracle Solaris", Pearson, 2011	C409.2	BTL5

UNIT III

SHARED MEMORY PROGRAMMING WITH OpenMP

OpenMP Execution Model – Memory Model – OpenMP Directives – Work-sharing Constructs – Library functions – Handling Data and Functional Parallelism – Handling Loops – Performance Considerations.

Q. No.	Questions	CO	Bloom's Level
1.	**What is OpenMP?** Like Pthreads, OpenMP is an API for shared-memory parallel programming. The "MP" in OpenMP stands for "multiprocessing," a term that is synonymous with shared-memory parallel computing. Thus, OpenMP is designed for systems in which each thread or process can potentially have access to all available memory, and, when we're programming with OpenMP, we view our system as a collection of cores or CPUs, all of which have access to main memory.	C409.3	BTL1
2.	**What is Pthread.** Pthreads is lower level and provides us with the power to program virtually any conceivable thread behavior. This power, however, comes with some associated cost—it's up to us to specify every detail of the behavior of each thread.	C409.3	BTL1
3.	**What the difference between Pthreads and OpenMP.** Pthreads requires that the programmer explicitly specify the behavior of each thread. OpenMP, on the other hand, sometimes allows the programmer to simply state that a block of code should be executed in parallel, and the precise determination of the tasks and which thread should execute them is left to the compiler and the run-time system. This suggests a further difference between OpenMP and Pthreads, that is, that Pthreads (like MPI) is a library of functions that can be linked to a C program, so any Pthreads program can be used with any C compiler, provided the system has a Pthreads library. OpenMP, on the other hand, requires compiler support for some operations, and hence it's entirely possible that you may run across a C compiler that can't compile OpenMP programs into parallel programs.	C409.3	BTL1
4.	**What is the Execution Model of OpenMp.** The OpenMP API uses the fork-join model of parallel execution. Multiple threads of execution perform tasks defined implicitly or explicitly by OpenMP directives. The OpenMP API is intended to support programs that will execute correctly both as parallel programs (multiple threads of	C409.3	BTL1

	execution and a full OpenMP support library) and as sequential programs (directives ignored and a simple OpenMP stubs library).		
5.	**What is an initial thread in OpenMP?** An OpenMP program begins as a single thread of execution, called an initial thread.	C409.3	BTL1
6.	**How to compile and running OpenMP programs** To compile this with gcc we need to include the -fopenmp option **$ gcc –g –Wall –fopenmp –o omp_hello omp_hello.c** To run the program, we specify the number of threads on the command line. For example, we might run the program with four threads and type $ ./omp_hello 4		
7	**What is termed as initial task region. .(Nov/Dec 2017)** The initial task region, that is defined by an implicit inactive `parallel` region surrounding the whole program.When any thread encounters a `parallel` construct, the thread creates a team of itself and zero or more additional threads and becomes the master of the new team. A set of implicit tasks, one per thread, is generated	C409.3	BTL1

8	**Define odd even transportation sort? . (Apr/May 2017)**
	• Parallelizable version of Bubble sort
	• Requires N passes through the array.
	• Each pass through the array analyzes either:
	– Every pair of odd indexed elements and the preceding element, or
	– Every pair of even indexed elements and the preceding element.
	• Within each pass, elements that are not in order are swapped.
	
	`Sort local keys;`
	`for (phase = 0; phase < comm_sz; phase++) {`
	`   partner = Compute_partner(phase, my_rank);`
	`   if (I'm not idle) {`
	`      Send my keys to partner;`
	`      Receive keys from partner;`
	`      if (my_rank < partner)`
	`         Keep smaller keys;`
	`      else`
	`         Keep larger keys;`
	`   }`
	`}`
		C409.3	BTL1

9.	**Develop a”hello word” program inthat uses open MP. . (Apr/May 2017)** Hello world program in C using MPI: `#include <stdio.h>`	C409.3	BTL1

	<pre>#include <mpi.h>
	
	main(int argc, char **argv)
	{
	   int ierr;
	
	   ierr = MPI_Init(&argc, &argv);
	   printf("Hello world\n");
	
	   ierr = MPI_Finalize();
	}</pre>		

10	**Define Message Queue.(Nov/Dec 2017)** Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them.	C409.3	BTL1
11.	**What is an initial task region?** An initial thread executes sequentially, as if enclosed in an implicit task region, called an initial task region, that is defined by the implicit parallel region surrounding the whole program.	C409.3	BTL1
12	**Discuss the Structure of the OpenMP Memory Model** The OpenMP API provides a relaxed-consistency, shared-memory model. All OpenMP threads have access to a place to store and to retrieve variables, called the *memory*. In addition, each thread is allowed to have its own *temporary view* of the memory.	C409.3	BTL1
13	**What is *threadprivate memory?*** Each thread also has access to another type of memory that must not be accessed by other threads, called *threadprivate memory*.	C409.3	BTL1
14	**Show the format of directive in OpenMP.** **#pragma omp** *directive-name [clause[[,] clause]...] new-line* Each directive starts with **#pragma omp**. The remainder of the directive follows the conventions of the C and C++ standards for compiler directives. In particular, white space can be used before and after the **#**, and sometimes white space must be used to separate the words in a directive.	C409.3	BTL2
15	**What is meant by Stand-alone directives?** Stand-alone directives do not have any associated executable user code. Instead, they represent executable statements that typically do not have succinct equivalent statements in the base languages. There are some restrictions on the placement of a stand-alone directive within a program. A stand-alone directive may be placed only at a point where a base language executable statement is allowed	C409.3	BTL1
16	**What is the use of parallel construct?** This fundamental construct starts parallel execution. **#pragma omp parallel** *[clause[[,]clause] ...] new-line* *structured-block* where *clause* is one of the following:	C409.3	BTL1

	#pragma omp parallel *[clause[[,]clause] ...] new-line* *structured-block* **if(***scalar-expression***)** **num_threads(***integer-expression***)** **default(shared | none)** **private(***list***)** **firstprivate(***list***)** **shared(***list***)** **copyin(***list***)** **reduction(***redution-identifier* **:***list***)** **proc_bind(master | close | spread)**		
17	**What is Worksharing Constructs?** A worksharing construct distributes the execution of the associated region among the members of the team that encounters it. Threads execute portions of the region in the context of the implicit tasks each one is executing. If the team consists of only one thread then the worksharing region is not executed in parallel.	C409.3	BTL1
18	**List some worksharing constructs.** The OpenMP API defines the following worksharing constructs. • loop construct • **sections** construct • **single** construct • **workshare** construct	C409.3	BTL4
19	**List the the Restrictions apply to worksharing constructs.** The following restrictions apply to worksharing constructs: • Each worksharing region must be encountered by all threads in a team or by none at all, unless cancellation has been requested for the innermost enclosing parallel region. • The sequence of worksharing regions and **barrier** regions encountered must be the same for every thread in a team.	C409.3	BTL4
20	**Show the syntax of the loop construct.** The syntax of the loop construct is as follows: **#pragma omp for** *[clause[[,] clause] ...] new-line* *for-loops* where *clause* is one of the following: **private(***list***)** **firstprivate(***list***)** **lastprivate(***list***)** **reduction(***reduction-identifier***:** *list***)** **schedule(***kind[, chunk_size]***)** **collapse(***n***)** **ordered** **nowait**	C409.3	BTL1
21	**What is meant by binding?** The binding thread set for a loop region is the current team. A loop region	C409.3	BTL1

	binds to the innermost enclosing **parallel** region. Only the threads of the team executing the binding **parallel** region participate in the execution of the loop iterations and the implied barrier of the loop region if the barrier is not eliminated by a **nowait** clause		
22	**What is the use of collapse clause.** The **collapse** clause may be used to specify how many loops are associated with the loop construct. The parameter of the **collapse** clause must be a constant positive integer expression. If no **collapse** clause is present, the only loop that is associated with the loop construct is the one that immediately follows the loop directive.	C409.3	BTL1
23	**List the Restrictions to the loop construct.** Restrictions to the loop construct are as follows: • All loops associated with the loop construct must be perfectly nested; that is, there must be no intervening code nor any OpenMP directive between any two loops. • The values of the loop control expressions of the loops associated with the loop construct must be the same for all the threads in the team. • Only one **schedule** clause can appear on a loop directive. • Only one **collapse** clause can appear on a loop directive. • *chunk_size* must be a loop invariant integer expression with a positive value. • The value of the *chunk_size* expression must be the same for all threads in the team. • The value of the *run-sched-var* ICV must be the same for all threads in the team. • When **schedule(runtime)** or **schedule(auto)** is specified, *chunk_size* must not be specified. • Only one **ordered** clause can appear on a loop directive. • The **ordered** clause must be present on the loop construct if any **ordered** region ever binds to a loop region arising from the loop construct. • The loop iteration variable may not appear in a **threadprivate** directive	C409.3	BTL4
24	**How to Determine the Schedule of a Worksharing Loop?** When execution encounters a loop directive, the **schedule** clause (if any) on the directive, and the *run-sched-var* and *def-sched-var* ICVs are used to determine how loop iterations are assigned to threads.. If the loop directive does not have a **schedule** clause then the current value of the *def-sched-var* ICV determines the schedule. If the loop directive has a **schedule** clause that specifies the **runtime** schedule kind then the current value of the *run-sched-var* ICV determines the schedule. Otherwise, the value of the **schedule** clause determines the schedule.	C409.3	BTL1
25	**What is The sections construct.** The **sections** construct is a non-iterative work sharing construct that	C409.3	BTL1

	contains a set of structured blocks that are to be distributed among and executed by the threads in a team. Each structured block is executed once by one of the threads in the team in the context of its implicit task.		
26	**Show the the syntax of the sections construct**. The syntax of the **sections** construct is as follows: **#pragma omp sections** *[clause[[,] clause] ...] new-line* **{***[* **#pragma omp section** *new-line]* *structured-block* *[***#pragma omp section** *new-line* *structured-block]* ... **}** where *clause* is one of the following **private(***list***)** **firstprivate(***list***)** **lastprivate(***list***)** **reduction(***reduction-identifier***:***list***)** **nowait**	C409.3	BTL1
27	**List the** Restrictions to the **sections** construct . • Orphaned **section** directives are prohibited. That is, the **section** directives must appear within the **sections** construct and must not be encountered elsewhere in the**sections** region. • The code enclosed in a **sections** construct must be a structured block. • Only a single **nowait** clause can appear on a **sections** directive. **firstprivate(***list***)** **lastprivate(***list***)** **reduction(***reduction-identifier***:***list***)** • A throw executed inside a **sections** region must cause execution to resume within the same section of the **sections** region, and the same thread that threw the exception must catch it.	C409.3	BTL4
28	**Show The syntax of the single construct .** **!$omp single** *[clause[[,] clause] ...]* *structured-block* **!$omp end single** *[end_clause[[,] end_clause] ...]* where *clause* is one of the following: **private(***list***)** **firstprivate(***list***)** and *end_clause* is one of the following:	C409.3	BTL1

	copyprivate(*list*) **nowait**		
29	**Show the syntax of the workshare construct.** **!$omp workshare** *structured-block* **!$omp end workshare *[*nowait*]*** The enclosed structured block must consist of only the following: • array assignments • scalar assignments • **FORALL** statements • **FORALL** constructs • **WHERE** statements • **WHERE** constructs • **atomic** constructs	C409.3	BTL1
30	**List the restrictions apply to the workshare construct:** • All array assignments, scalar assignments, and masked array assignments must be intrinsic assignments. • The construct must not contain any user defined function calls unless the function is **ELEMENTAL**	C409.3	BTL4
31	**What is the use of schedule clause.** If more than one loop is associated with the loop construct, then the iterations of all associated loops are collapsed into one larger iteration space that is then divided according to the **schedule** clause. The sequential execution of the iterations in all associated loops determines the order of the iterations in the collapsed iteration space	C409.3	BTL1
32	**What is the use of single construct.** The **single** construct specifies that the associated structured block is executed by only one of the threads in the team (not necessarily the master thread), in the context of its implicit task. The other threads in the team, which do not execute the block, wait at an implicit barrier at the end of the **single** construct unless a **nowait** clause is specified	C409.3	BTL1
33	**List the Restrictions to the single construct are as follows:** • The **copyprivate** clause must not be used with the **nowait** clause. • At most one **nowait** clause can appear on a **single** construct.	C409.3	BTL4
34	**What is workshare construct?** The **workshare** construct divides the execution of the enclosed structured block into separate units of work, and causes the threads of the team to share the work such that each unit is executed only once by one thread, in the context of its implicit task.	C409.3	BTL1
35	**Explain Scope of a variable (Apr/May 2018)**	C409.3	BTL1

36	**Define Race Condition (Apr/May 2018)**	C409.3	BTL1

37	**State the trapezoidal rule in OpenMP (Nov/Dec 2018)**
	for (i = 1; i <= n-1; i++)
	{
	x_i = a + i*dx;
	approx += f(x_i);
	}
	approx = dx*approx;
		C409.3	BTL1

38	**Define Coherence and consistency** • Coherence refers to the behavior of the memory system when a single memory location is accessed by multiple threads. • Consistency refers to the ordering of accesses to different memory locations, observable from various threads in the system	C409.3	BTL1
39	**Define data race** • A data race is defined to be accesses to a single variable by at least two threads, at least one of which is a write, not separated by a synchronization operation. • OpenMP does guarantee certain consistency behavior, however. That behavior is based on the OpenMP flush operation.	C409.3	BTL1
40	**Define OpenMP flush operation** The OpenMP flush operation is applied to a set of variables called the flush set. Memory operations for variables in the flush set that precede the flush in program execution order must be firmly lodged in memory and available to all threads before the flush completes, and memory operations for variables in the flush set, that follow a flush in program order cannot start until the flush completes.		
41	**Mention the actions to be taken to move the value from one thread to another thread** • The first thread writes the value to the shared variable, • The first thread flushes the variable • The second thread flushes the variable and • The second thread reads the variable	C409.3	BTL1
42	**List out the two methods available for enabling nested parallel regions** 1.The **omp_set_nested()** library routine 2. Setting of the **OMP_NESTED** environment variable to TRUE	C409.3	BTL1
43	**What is data dependences** 1. OpenMP compilers don't check for dependences among iterations in a loop that's being parallelized with a parallel **for** directive. It's up to us, the programmers, to identify these dependences. **2.** A loop in which the results of one or more iterations depend on other	C409.3	BTL1

	iterations *cannot*, in general, be correctly parallelized by OpenMP.		
44	**Define Atomic Directive** The atomic directive ensures that a specific memory location is updated atomically, rather than exposing it to the possibility of multiple, simultaneous writing threads. The atomic directive supports no OpenMP clauses. **Syntax** **#pragma omp atomic expression**	C409.3	BTL1
45	**Define Critical Directive** Specifies that code is only executed on one thread at a time. OpenMP *does* provide the option of adding a name to a critical directive: **Syntax** **# pragma omp critical(name)**	C409.3	BTL1
46	**List out the two types of locks in OpenMP to destroy lock data structures** • Simple locks • Nested Locks	C409.3	BTL1
47	**Define Barrier Directive and Master Directive** A barrier directive will cause the threads in a team to block until all the threads have reached the directive. **Syntax** **#pragma omp barrier** Specifies that only the master thread should execute a section of the program. **Syntax** **# pragma omp master**	C409.3	BTL1
48	**Which are the OpenMP clauses supported by Single directive** • copyprivate • firstprivate • nowait • private	C409.3	BTL1
49	**Define Synchronization Clauses** • Critical • Atomic • Ordered • Barrier • Nowait	C409.3	BTL1
50	**List out the three types of Scheduling?** • Static • Dynamic • Guided	C409.3	BTL1
51	**Define Data parallelism** **Data parallelism** is a form of parallelization across multiple	C409.3	BTL1

<table>
<tr><td></td><td>processors in parallel computing environments. It focuses on distributing the data across different nodes, which operate on the data in parallel. It can be applied on regular data structures like arrays and matrices by working on each element in parallel.</td><td></td><td></td></tr>
<tr><td>52</td><td>List out the four discrete steps to parallelization
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>Decomposition</td><td>The program is broken down into tasks, the smallest exploitable unit of concurrence.</td></tr>
<tr><td>Assignment</td><td>Tasks are assigned to processes.</td></tr>
<tr><td>Orchestration</td><td>Data access, communication, and synchronization of processes.</td></tr>
<tr><td>Mapping</td><td>Processes are bound to processors.</td></tr>
</table></td><td>C409.3</td><td>BTL1</td></tr>
<tr><td>53</td><td>What are Loop Carried Dependencies
Loop-Carried Dependence Verification in OpenMP. Data dependence analysis is a very difficult task, mainly due to the limitations imposed by pointer aliasing, and by the overhead of dynamic data dependence analysis.</td><td>C409.3</td><td>BTL1</td></tr>
</table>

PART B

Q. No.	Questions	CO	Bloom's Level
1.	**Explain OpenMP Execution Model in detail with example.(Nov/Dec 2017) (Apr/May 2018)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:210-215	C409.3	BTL5
2.	**Explain the Memory Model of OpenMP. (Apr/May 2018)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:213-215	C409.3	BTL5
3.	**Explain the OpenMP Directives . (Apr/may2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:224-231	C409.3	BTL5
4.	**Explain the Library functions used in OpenMP.** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:	C409.3	BTL5

5.	**Explain in detail how to Handle Loops in OpenMP (Nov/Dec 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No: 236-240	C409.3	BTL5
6.	**Explain OpenMP directives. . (Apr/may 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:224-231	C409.3	BTL5
7.	**How data and functional parallelism are handled in shared memory programming with open MP? (Apr/may2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:212-215	C409.3	BTL5
8.	**Explain in detail about the handling loops in parallel operations (Nov/Dec 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:236-240	C409.3	BTL5
9	**Write an example program for shared memory programming with Pthread (Apr/May 2018)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:236-240	C409.3	BTL5
10	**Explain in detail about the synchronization primitives in parallel program challenges (Apr/May 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:236-240	C409.3	BTL5

11	**Explain the type shared memory model** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011		BTL5
12	**Collect the all information about internal control variable** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011		BTL1
13	**Explain briefly about General Data Parallelism** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011		BTL4
14	**(i)Explain the NoWait Clause** **(ii)Explain the single pragma** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011		BTL4
15	**(i)Illustrate the runtime library definitions** **(ii)llustrate the execution environment routines** 1. Peter S. Pacheco, “An Introduction to Parallel Programming”, Morgan-Kauffman/Elsevier, 2011		BTL3

UNIT IV

DISTRIBUTED MEMORY PROGRAMMING WITH MPI

MPI program execution – MPI constructs – libraries – MPI send and receive – Point-to-point and Collective communication – MPI derived datatypes – Performance evaluation.

PART A

Q. No.	Questions	CO	Bloom's Level
1.	**What is MPI?** One process calls a ***send*** function and the other calls a ***receive*** function. The implementation of message-passing that will be using is called **MPI** (Message-**Passing Interface)**. MPI is not a new programming language. It defines a *library* of functions that can be called from C, C++, and FORTRAN Programs	C409.4	BTL1
2.	**How to execute MPI programs?** Many systems use the command called mpicc to compile and run MPI programs: **mpicc -g -Wall -o mpi hello mpi hello.c**	C409.4	BTL1
3.	**What is the use of Wrapper Script. (Apr/May 2017)** A wrapper script is a script whose main purpose is to run some program. In this case, the program is the C compiler.The wrapper simplifies the running of the compiler by telling it where to find the necessary header files and which libraries to link with the object file.	C409.4	BTL1
4.	**Show the syntax of MPI_init and MPI_finalize** The syntax of MPI_ Init ( ): **int** MPI _Init(**int*** argc p /* in/out */, **char***** argv p /* in/out */); The syntax of MPI_ Finalize( ): int MPI_ Finalize(void);	C409.4	BTL1
5.	**What is Communicator in MPI?** In MPI a **communicator** is a collection of processes that can send messages to each other. One of the purposes of MPI Init is to define a communicator that consists of all of the processes started by the user when she started the program. This communicator is called MPI_ COMM_WORLD.	C409.4	BTL1

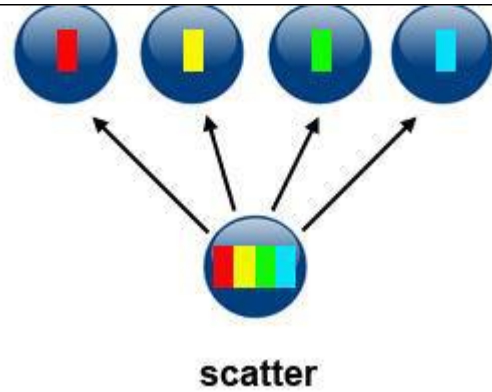
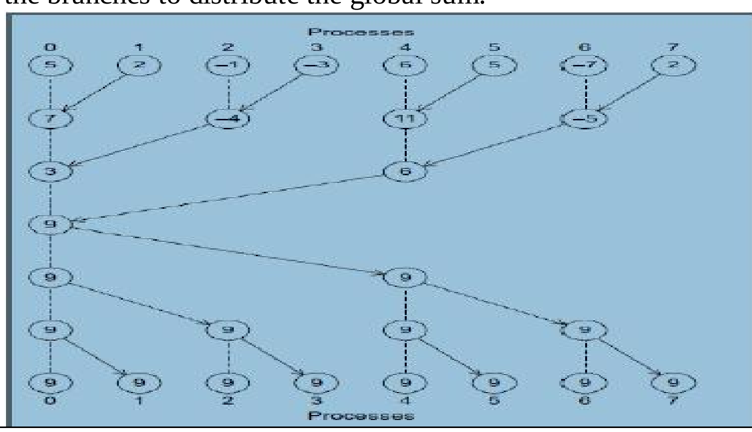
6.	**What is a SPMD program?** A single program is written so that different processes carry out different actions, and this is achieved by simply having the processes branch on the basis of their process rank. This approach to parallel programming is called **single program, multiple data, or SPMD**	C409.4	BTL1
7.	**Give the syntax of MPI_Get_count** The syntax of MPI_Get_count is int MPI_Get_count(MPI_Status* status_p /* in */, MPI_Datatype type /* in */, int* count p /*out */);	C409.4	BTL1
8.	**What is Non-overtaking?** If process q sends two messages to process r, then the first message sent by q must be available to r before the second message. There is no restriction on the arrival of messages sent from different processes. ie., if q and t both send messages to r, then even if q sends its message before t sends its message, there is no requirement that q's message become available to r before t's message. This is essentially because MPI can't impose performance on a network.	C409.4	BTL1
9	**Evaluate the performance evaluation methods in distributed memory programming. .(Nov/Dec 2017,** 1.Timing performance 2.performance Results 3.Speedup performance 4.efficiency performance 5.Scalability performance	C409.4	BTL5
10	**What is wildcard argument?** The wildcard arguments: o Only a receiver can use a wildcard argument. Senders must specify a process rank and a nonnegative tag. Thus, MPI uses a "push" communication mechanism rather than a "pull" mechanism. There is no wildcard for communicator arguments; both senders and receivers must always specify communicators	C409.4	BTL1
11	**Show the area of the trapezoid.** Area of one trapezoid = **h/2[f (x_i) + (f (x_{i+1})]**	C409.4	BTL1
12	**Define collective communications in MPI.** In MPI parlance, communication functions that involve all the processes in a communicator are called collective communications.	C409.4	BTL1
13	**What is Local Variables? Give Examples**. Local variables are variables whose contents are significant only on the process that's using them. Some examples are: **local a, local b** and	C409.4	BTL1

	local n.		
14	**What is Global Variables? Give Examples.** Variables whose contents are significant to all the processes are sometimes called global variables. Some examples are: **a, b** and **n.**	C409.4	BTL1
15	**What is point to point communications in MPI?** **MPI_Send** and **MPI_Recv** are often called point-to-point communications.	C409.4	BTL1
16	**List any two points stating that how collective communication differ from point to point communication.** ▪ All the processes in the communicator must call the same collective function. Example, a program that attempts to match a call to **MPI_Reduce** on one process with a call to **MPI_ Recv** on another process is erroneous, and, in all likelihood, the program will hang or crash.	C409.4	BTL4
17	**What is a butterfly-structured global sum?** The processes exchange partial results instead of using one-way communications. Such a communication pattern is sometimes called a butterfly-structured global sum.	C409.4	BTL1
18	**What is broadcast?** A collective communication in which data belonging to a single process is sent to all of the processes in the communicator is called a **broadcast**	C409.4	BTL1
19	**Show a broadcast function.** A broadcast function: int MPI_Bcast (void* data p /* in/out */, int count /* in */, MPI_Datatype datatype /* in */, int source proc /* in */, MPI_Comm comm /* in */);	C409.4	BTL1
20	**Define block partition of the vector.** The work consists of adding the individual components of the vectors, so we might specify that the tasks are just the additions of corresponding components. Then there is no communication between the tasks, and the problem of parallelizing vector addition boils down to aggregating the tasks and assigning them to the cores. If the number of components is n and we have comm_sz cores or processes, let's assume that n evenly divides comm_sz and define local n D n=comm sz. Then we can simply assign blocks of local n consecutive components to each process. This is often called a **block partition** of the vector.	C409.4	BTL1
21	**Show MPI Scatter function.** The communication MPI provides a function: int MPI Scatter(void* send_buf_p /* in */, int send_count /* in */, MPI_Datatype send_type /* in */,	C409.4	BTL1

	void* recv_buf_p /* out */, int recv_count /* in */, MPI_Datatype recv_type /* in */, int src_proc /* in */, MPI_Comm comm /* in */);		
22	**Give MPI Gather function.** The communication in this function can be carried out by MPI Gather, int MPI_Gather(void* send_buf_p /* in */, int send_count /* in */, MPI_Datatype send_type /* in */, void* recv_buf_p /* out */, int recv_count /* in */, MPI_Datatype recv_type /* in */, int dest_proc /* in */, MPI_Comm comm /* in */);	C409.4	BTL1
23	**What is Derived data type in MPI?** In MPI, a derived data type can be used to represent any collection of data items in memory by storing both the types of the items and their relative locations in memory.	C409.4	BTL1
24	**Show the use MPI_Type_create_struct.** We can use MPI Type create struct to build a derived datatype that consists of individual elements that have different basic types: int MPI_Type_create_struct(int count /* in */, int array_of_blocklengths[] /* in */, MPI_Aint array_of_displacements[] /* in */, MPI_Datatype array_of_types[] /* in */, MPI_Datatype* new_type_p /* out */);	C409.4	BTL1
25	**Show the ratio of the serial and parallel runtime.** The most widely used measure of the relation between the serial and the parallel run-times is the speedup. It's just the ratio of the serial run-time to the parallel run-time: $S(n, p) = \frac{T_{serial}(n)}{T_{parallel}(n, p)}$	C409.4	BTL1
26	**Show "per process" speedup?** The "Per Process speedup is	C409.4	BTL1

	$$E(n,p) = \frac{S(n,p)}{p} = \frac{T_{serial}(n)}{p * T_{parallel}(n,p)}$$		
27	**what is a wrapper script? . (apr/may2017)** A wrapper script is a script whose main purpose is to run wrapper the running of the compiler by telling it where to find the necess which libraries to link with the object file .	C409.4	BTL1
28	**How to design a parallel program?** A parallel program can be designed using four basic steps: **1.** Partition the problem solution into tasks. **2.** Identify the communication channels between the tasks. **3.** Aggregate the tasks into composite tasks. **4.** Map the composite tasks to cores.	C409.4	BTL1
29	**What is linear speedup?** The ideal value for ***S (n,p)*** is p. If ***S (n,p) = p***, then our parallel program with ***comm_ sz = p*** processes is running ***p*** times faster than the serial program. This speedup, sometimes called **linear speedup**.	C409.4	BTL1
30	**What is block-cyclic partition?** Instead of using a cyclic distribution of individual components, use a cyclic distribution of blocks of components, so a block-cyclic distribution isn't fully specified until we decide how large the blocks are	C409.4	BTL1
31	**What are the possibilities for choosing a destination when sending requests for work with MPI** MPI is designed to allow users to create programs that can run efficiently on most parallel architectures. The design process included vendors (such as IBM, Intel, TMC, Cray, Convex, etc.), parallel library authors (involved in the development of PVM, Linda, etc.), and applications specialists	C409.4	BTL1
32	**List the restrictions work sharing constructs** The sequence of *work-sharing constructs* and barrier directives encountered must be the same for every thread in a team	C409.4	BTL1
33	**Write the evaluation methods is distributed memory programming**	C409.4	BTL1
34	**Give the commands for MPI**	C409.4	BTL1
35	**Define and broadcast and butterfly MPI**	C409.4	BTL1
36	**What is MPI W_Time** MPI provides a function, MPI_Wtime, that returns the number of seconds that have elapsed since some time in the past: **Double** MPI_Wtime(**void**);	C409.4	BTL1
37	**Define MPI_Barrier** The MPI collective communication function MPI_Barrier insures that no process will return from calling it until every process in the communicator has started calling it.	C409.4	BTL1

38	**Define Speed-Up and Efficiency** The most widely used measure of the relation between the serial and the parallel run-times is the **speedup**. It's just the ratio of the serial run-time to the parallel run-time: $\mathbf{S(n,p)=T_{Serial(n)}/T_{Serial(n,p)}}$	C409.4	BTL1
39	**How the $T_{overhead}$ is represented** The parallel run-time is denoted by*T*parallel. Since it depends on both the input size, *n*, and the number of processes, comm._sz= *p*, we'll frequently denote it as *T*parallel(*n,p)*. The parallel program will divide the work of the serial program among the processes, and add in some overhead time, which we denoted *T*overhead: $\mathbf{T_{parallel}(n,p)=T_{serial}(n/p)+T_{overhead}}$	C409.4	BTL1
40	**Define Speed up** The most widely used measure of the relation between the serial and the parallel run-times is the **speedup**. It's just the ratio of the serial run-time to the parallel run-time: $\mathbf{S(n,p)=T_{serial}(n)/T_{parallel(n,p)}}$	C409.4	BTL1
41	**Define Efficiency** This speedup, sometimes called **linear speedup**. Another widely used measure of parallel performance is parallel **efficiency**. $\mathbf{E(n,p)= S(n,p)/p= T_{serial}(n)/P^* T_{parallel(n,p)}}$	C409.4	BTL1
42	**What is MPI derived data types** In MPI, a **derived datatype** can be used to represent any collection of data items in memory by storing both the types of the items and their relative locations in memory.	C409.4	BTL1
43	**Define Gather** Gathers distinct messages from each task in the group to a single destination task. This routine is the reverse operation of MPI_Scatter. The data stored in the memory referred to by send_buf_p on process 0 is stored in the first block in recv_buf p, the data stored in the memory referred to by send buf_p on process 1 is stored in the second block referred to by recv_buf_p, and so on.	C409.4	BTL1
44	**Define Scatter** Distributes distinct messages from a single source task to each task in the group.	C409.4	BTL1

			
45	**List out the types of collective operations** **Synchronization -** processes wait until all members of the group have reached the synchronization point. • **Data Movement -** broadcast, scatter/gather, all to all **Collective Computation (reductions) -** one member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on that data.	C409.4	BTL1
46	**Define MPI_Allreduce** Consider a situation in which *all* of the processes need the result of a global sum in order to complete some larger computation. For example, if we use a tree to compute a global sum, we might ―reverse‖ the branches to distribute the global sum. 	C409.4	BTL1
47	**List out the two possibilities when the message are assembled** • the sending process can **buffer** the message or • it can block	C409.4	BTL1
48	**What are the types of MPI type** The MPI type MPI_Status is a struct with at least the three members **MPI_ SOURCE, MPI_TAG**, and **MPI_ERROR.** Suppose our program contains the definition MPI_Status status; Then, after a call to MPI Recv in which &status is passed as the last argument, we can determine the sender and tag by examining the two members **status.MPI SOURCE** **status.MPI TAG**	C409.4	BTL1
49	**What is status_p argument** • The Amount of Data In The Message	C409.4	BTL1

	• The Sender of The Message, Or • The Tag of The Message.		

50	**Mention the syntax for MPI_send**
	`int MPI_Send(`
	`void* msg_buf_p /* in */,`
	`int msg_size /* in */,`
	`MPI_Datatype msg_type /* in */,`
	`int dest /* in */,`
	`int tag /* in */,`
	`MPI_Comm communicator /* in */);`
		C409.4	BTL1

51	**Write a note on Distributed memory machines (Nov/Dec 2018)** Programming on a distributed memory machine is a matter of organizing a program as a set of independent tasks that communicate with each other via messages. In addition, programmers must be aware of where data is stored, which introduces the concept of locality in parallel algorithm design. An algorithm that allows data to be partitioned into discrete units and then runs with minimal communication between units will be more efficient than an algorithm that requires random access to global structures.	C409.4	BTL1
52	**How to compile an MPI Program (Nov/Dec 2018)** Many systems use a command called mp icc for compilation mp icc is a script that's a **wrapper** for the C compiler.	C409.4	BTL1

.

PART B

Q. No.	**Questions**	**CO**	**Bloom's Level**
1.	**What is MPI? Write a program"hello,world" that makes some use of MPI. how to compile and execute MPI programs. .(Nov/Dec 2017)**. 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:86-90	C409.4	BTL1
2.	**Explain about Trapezoidal rule in MPI in detail** 1.Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:94-96	C409.4	BTL1
3.	**Give in detail about Collective Communication.** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:101-115	C409.4	BTL1

4.	**Explain the following MPI functions:** ▪ **MPI_Reduce** ▪ **MPI_Allreduce** ▪ **MPI_Scatter** ▪ **MPI_Gather** ▪ **MPI_Allgather** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:88-91 ▪	C409.4	BTL1
5.	**Compare and contrast Collective communication Vs Point to point communication.(16) (Nov/Dec 2017)**. 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:105-106	C409.4	BTL2
6.	**Disscus about MPI Derived data types with example programs. (10)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:116-118	C409.4	BTL6
7.	**i.Explain about Performance Evaluation of MPI Programs in detail.(Apr/May 2017)** **ii.What are the performance issues in multicore processor?** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:119-126	C409.4	BTL5
8.	**i.Explain tree structured communication** **ii.What are the differences between point to point and collective communication? (Apr/May 2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:102-103	C409.4	BTL5
9	**(i)Explain Loop Handling in detail** **(ii)Describe about MPI programs execution with example (Apr/May 2018)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.4	BTL5
10	**Explain the virtual memory in detail (Apr/May 2018)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.4	BTL5
11	**(i)Describe the Attribute Caching** **(ii)Discuss about the communicators** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.4	BTL2

12	**(i)Describe the Distributed array datatype constructor** **(ii)Explain the Cartesian constructor** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.4	BTL5
13	**(i) Generalize the group of process** **(ii) Explain the virtual topology** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.4	BTL6
14	**(i)Describe the MPI program execution** **(ii)Describe about MPI Init and Finalize** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.4	BTL1
15	**(i)Describe about the Datatype constructor** **(ii)Discuss about the subarray datatype constructor** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.4	BTL5

UNIT V

PARALLEL PROGRAM DEVELOPMENT

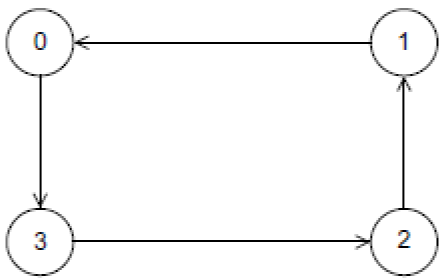
Case studies - n-Body solvers – Tree Search – OpenMP and MPI implementations and comparison.

PART A

Q. No.	Questions	CO	Bloom's Level
1.	**What is an n-body problem?** In an n-body problem, it needs to find the positions and velocities of a collection of interacting particles over a period of time. An n-body solver is a program that finds the solution to an n-body problem by simulating the behavior of the particles. The input to the problem is the mass, position, and velocity of each particle at the start of the simulation, and the output is the position and velocity of each particle at a sequence of user-specified times, or simply the position and velocity of each particle at the end of a user-specified time period.	C409.5	BTL1

2.	**Show the pseudocode of a serial n-body solver.**
	Get input data;
	for each timestep {
	    if (timestep output) Print positions and velocities of particles;
	    for each particle q
	        Compute total force on q;
	    for each particle q
	        Compute position and velocity of q;
	}
	Print positions and velocities of particles;
		C409.5	BTL1

3.	**List the two algorithms in the n-body solver.** The n-body solver with the original force calculation, the **basic** algorithm, and the solver with the number of calculations reduced the **reduced** algorithm	C409.5	BTL1
4.	**List the two algorithms in the n-body solver.** The n-body solver with the original force calculation, the **basic** algorithm, and the solver with the number of calculations reduced the **reduced** algorithm	C409.5	BTL1
5.	**Differentiate between two algorithms in n-body solver** The *n*-body solver with the original force calculation, the *basic* algorithm, and the solver with the number of calculations reduced, the *reduced* algorithm.		
6.	**Define Graph.** A **graph** is a collection of vertices and edges or line segments joining pairs of vertices.	C409.5	BTL1
7.	**List the use of Recursive depth-first search algorithm.** The algorithm makes use of several global variables: • **n:** the total number of cities in the problem • **digraph:** a data structure representing the input digraph • **hometown:** a data structure representing vertex or city 0, the salesperson's hometown • **best tour:** a data structure representing the best tour so far	C409.5	BTL1
8	**What are the disadvantages of recursive depth first stage** It also has the disadvantage that at any given instant of time only This could be a problem when we try to parallelize tree search by threads or processes		
8.	**List the similarities parallelizing the solvers using pthreads.** The more important similarities are: ➢ By default local variables in Pthreads are private, so all shared variables are global in the Pthreads version. ➢ The principal data structures in the Pthreads version are identical to those in the OpenMP version: vectors are two-dimensional arrays of doubles, and the mass, position, and velocity of a single particle are stored in a struct. The forces are stored in an array of vectors. ➢ Startup for Pthreads is basically the same as the startup for OpenMP: the main thread gets the command-line arguments, and allocates and initializes	C409.5	BTL1

	the principal data structures. ➢ The main difference between the Pthreads and the OpenMP implementations is in the details of parallelizing the inner loops. Since Pthreads has nothing analogous to a parallel for directive, we must explicitly determine which values of the loop variables correspond to each thread's calculations. To facilitate this, a function Loop schedule, which determines • the initial value of the loop variable, • the final value of the loop variable, and • the increment for the loop variable. ➢ The input to the function is • the calling thread's rank, • the number of threads, • the total number of iterations, and an argument indicating whether the partitioning should be block or cyclic		
9.	**Define communication structure** A communication structure is called a **ring pass**. In a ring pass, imagine the processes as being interconnected in a ring. Process 0 communicates directly with processes 1 and comm._ sz-1, process 1 communicates with processes 0 and 2, and so on. The communication in a ring pass takes place in phases, and during each phase each process sends data to its "lower-ranked" neighbor, and receives data from its "higher-ranked" neighbor. Thus, 0 will send to comm._sz -1 and receive from 1. 1 will send to 0 and receive from 2, and so on. In general, process q will send to process (q-1+comm._sz)%comm_sz and receive from process (q+1)%comm_sz.  	C409.5	BTL1
9	**List the properties of function First_index**. The function First index should determine a global index glb part2 with the following properties: 1. The particle glb_part2 is assigned to the process with rank owner. 2. glb_part1 < glb_part2 < glb_part1 + comm_sz.	C409.5	BTL1
10	**What is a word about I/O.** The basic I/O was designed for use by single-process, single-threaded programs, and when multiple processes or multiple threads attempt to access the I/O buffers; the system makes no attempt to schedule their access.	C409.5	BTL1
11	**What is race condition? .(Nov/Dec 2017)** A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper	C409.5	BTL1

	sequence to be done correctly.		

12	**Show the Pseudocode for the MPI implementation of the reduced n-body solver.**
	source = (my_rank + 1) % comm._sz;
	dest = (my_rank - 1 + comm._sz) % comm._sz;
	Copy loc_pos into tmp_pos;
	loc_forces = tmp_forces = 0;
	
	Compute forces due to interactions among local particles;
	for (phase = 1; phase < comm._sz; phase++) {
	Send current tmp_pos and tmp_forces to dest;
	Receive new tmp_pos and tmp_forces from source;
	/* Owner of the positions and forces we're receiving */
	owner = (my_rank + phase) % comm._sz;
	Compute forces due to interactions among my particles
	and owner's particles;
	}
	Send current tmp_pos and tmp_forces to dest;
	Receive new tmp_pos and tmp_forces from source;
		C409.5	BTL1

13	**What is NP-complete problem? .(Apr/May 2017)** In computational complexity theory, a decision problem is NP-complete when it is both in NP and NP-hard. The set of NP-complete problems is often denoted by NP-C or NPC. The abbreviation NP refers to "nondeterministic polynomial time".	C409.5	BTL1
14	**What is Depth-first search?** Depth-first search (DFS) is an algorithm .for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.	C409.5	BTL1
15	**What is Recursive depth-first search?** Depth-first search (DFS) is an algorithm that traverses a graph in search of one or more goal nodes. The defining characteristic of this search is that, whenever DFS visits a maze cell c, it recursively searches the sub-maze whose origin is c. This recursive behaviour can be simulated by an iterative algorithm using a stack. A cell can have three states: **Unvisited.** The cell has not yet been visited by DFS. • **Visit In Progress**. The cell has been discovered, but not yet finished. Ie, the recursive search which begins at this node has not yet terminated. • **Visited**. The cell has been discovered, and the submazes which start at this node have been completely visited also.	C409.5	BTL1
16	**What problem occurs when test lock condition and update lock condition combined?** The combination of **"test lock condition"** and **"update lock condition"** can	C409.5	BTL1

	cause a problem: the lock condition (e.g. the cost of the best tour) can change between the time of the first test and the time that the lock is acquired. Thus, the threads also need to check the lock condition after they acquire the lock.		

17	**Show the pseudocode for updating the best tour.**
	The pseudocode for updating the best tour should look something like this:
	if (new tour cost < best tour cost) {
	    Acquire lock protecting best tour;
	    if (new tour cost < best tour cost)
	        Update best tour;
	    Relinquish lock;
	}
		C409.5	BTL1

18	**Show the syntax for mutex checking.** Pthreads has a ***nonblocking*** version of ***pthreads_mutex_lock*** called ***pthread_mutex_trylock***. This function checks to see if the mutex is available. If it is, it acquires the mutex and returns the value 0. If the mutex isn't available, instead of waiting for it to become available, it will return a nonzero value.	C409.5	BTL1
19	**Define MPI.** The ***message passing interface (MPI)*** is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory	C409.5	BTL1

20	**Show a pseudocode for a recursive solution to TSP using depth firs search.(Apr/May2017)**
	`void Depth_first_search(tour_t tour) {`
	`   city_t city;`
	
	`   if (City_count(tour) == n) {`
	`      if (Best_tour(tour))`
	`         Update_best_tour(tour);`
	`   } else {`
	`      for each neighboring city`
	`         if (Feasible(tour, city)) {`
	`            Add_city(tour, city);`
	`            Depth_first_search(tour);`
	`            Remove_last_city(tour, city);`
	`         }`
	`   }`
	`}  /* Depth_first_search */`
		C409.5	BTL1

21	**What are the features of distributed memory?(Nov/Dec 2017)** distributed memory refers to a multiprocessor computer system in which each processor has its own private memory. Computational tasks can only operate on local data, and if remote data is required, the computational task must communicate	C409.5	BTL1

	with one or more remote processors.		
22	**Show the syntax of MPI_Pack and MPI_Unpack.** int MPI_Pack(void* data to be packed /* in */ int to be packed count / _ in _/, MPI_Datatype datatype /_ in _/, void_ contig buf /_ out _/, int contig buf size /_ in _/, int_ position p /_ in/out _/, MPI Comm comm /_ in _/); int MPI Unpack(void_ contig buf /_ in _/, int contig buf size /_ in _/, int_ position p /_ in/out _/, void_ unpacked data /_ out _/, int unpack count /_ in _/, MPI Datatype datatype /_ in _/, MPI Comm comm /_ in _/);	C409.5	BTL1
23	**List the use of MPI_IN_PLACE argument.** The use of argument ***MPI_IN_PLACE*** is that the input and output buffers are the same. This can save on memory and the implementation may be able to avoid copying from the input buffer to the output buffer	C409.5	BTL1
24	**What the use of functions MPI Scatter and MPI Gather.** The functions ***MPI_Scatter*** and ***MPI_Gather*** can be use to split an array of data among processes and collect distributed data into a single array, respectively	C409.5	BTL1
25	**What are the use of functions MPI_Scatterv and MPI_Gatherv.** When the amount of data going to or coming from each process is the same for each process. If we need to assign different amounts of data to each process, or to collect different amounts of data from each process we can use MPI_Scatterv and MPI_Gatherv, respectively.	C409.5	BTL1
26	**Show the syntax of MPI_Scatterv.** int MPI Scatterv(void* sendbuf /* in */, int* sendcounts /* in */, int* displacements /* in */, MPI_Datatype sendtype /* in */, Void* recvbuf /* out */, int recvcount /* in */, MPI_Datatype recvtype /* in */, int root /* in */, MPI_Comm comm /* in */);	C409.5	BTL1
27	**Show the syntax of MPI_ Gatherv.**	C409.5	BTL1

	int MPI_Gatherv(void* sendbuf /* in */, int sendcount /* in */, MPI_Datatype sendtype /* in */, void* recvbuf /* out */, int* recvcounts /* in */, int* displacements /* in */, MPI_Datatype recvtype /* in */, int root /* in */, MPI_Comm comm /* in */);		
28	**What are the three modes provided by MPI?** MPI provides three other modes for sending: **synchronous, standard**, and **ready**.	C409.5	BTL1
29	**List the use of functions MPI_Pack and MPI_Unpack.** MPI provides the function ***MPI_Pack*** for storing a data structure in a single, contiguous buffer before sending. Similarly, the function ***MPI_Unpack*** can be used to take data that's been received into a single contiguous buffer and unpack it into a local data structure	C409.5	BTL1
30	**List the use of ready mode.** Ready sends ***(MPI_Rsend)*** are erroneous unless the matching receive has already been started when ***MPI_Rsend*** is called. The ordinary send ***MPI_Send*** is called the standard mode send.	C409.5	BTL1
31	**List the use of Synchronous mode.** Synchronous sends won't buffer the data; a call to the synchronous send function MPI_Ssend won't return until the receiver has begun receiving the data.	C409.5	BTL1

32	**How to compute n-body forces**
	
	for each particle q
	forces[q] = 0;
	for each particle q {
	for each particle k > q
	{
	x _diff = pos[q][X] - pos[k][X];
	y_diff = pos[q][Y] - pos[k][Y];
	dist = sqrt(x diff_x diff + y diff_y diff);
	dist_cubed = dist_dist_dist;
	force qk[X] = G_masses[q]_masses[k]/dist cubed _ x_diff;
	force qk[Y] = G_masses[q]_masses[k]/dist cubed _ y_diff
	forces[q][X] += force qk[X];
	forces[q][Y] += force qk[Y];
	forces[k][X] -= force qk[X];
	forces[k][Y] -= force qk[Y];
	}
	}
		C409.5	BTL1

33	**List the data structures used for serial implementation** The data structures are the tour, the digraph, and, in the iterative implementations, the sta and the stack are essentially list structures. For tour instead of array structure with three the array storing the cities, the number of cities, and the cost of the partial tour. When the digraph are represented using List.	C409.5	BTL1
34	**Difference between Parallelizing the two *n*-body solvers using pthread and OpenMP.** The main difference between the Pthreads and the OpenMP implementations is in th parallelizing the inner loops. Since Pthreads has nothing analogous to a parallel for must explicitly determine which values of the loop variables correspond to each thre calculations. To facilitate this a function Loop schedule which contains . the initial value of the loop variable, . the final value of the loop variable, and . the increment for the loop variable.	C409.5	BTL1
35	**How the performance of the reduced solver is much superior to the performance of the basic solver?** The efficiency of the basic solver on 16 nodes is about 0.95, while the efficiency of the redu 16 nodes is only about 0.70. A point to stress here is that the reduced MPI solver makes mu efficient use of memory than the basic MPI solver the basic solver must provide storage for on each process, while the reduced solver only needs extra storage for *n*=comm_sz position *n*=comm_sz forces	C409.5	BTL1

36	**How can we use OpenMP to map tasks/particles to cores in the basic version o *n-body solver?***
	
	for each timestep
	{
	if (timestep output) Print positions and velocities of particles;
	for each particle q
	Compute total force on q;
	for each particle q
	Compute position and velocity of q;
	}
	The two inner loops are both iterating over particles. So, in principle, parallelizin the two inner **for** loops will map tasks/particles to cores, and we might try some like this:
	for each timestep
	{
	if (timestep output) Print positions and velocities of particles;
		C409.5	BTL1

	# pragma omp parallel **for** **for** each particle q Compute total force on q; # pragma omp parallel **for** **for** each particle q Compute position and velocity of q; }		

37	**Write the pseudocode for the MPI version of the basic *n*-body solver?**
	
	Get input data;
	for each timestep {
	if (timestep output)
	Print positions and velocities of particles;
	for each local particle loc q
	Compute total force on loc q;
	for each local particle loc q
	Compute position and velocity of loc q;
	Allgather local positions into global pos
	array;
	}
	Print positions and velocities of particles;
		C409.5	BTL1

38	**Write the pseudocode for the MPI implementation of the reduced *n*-body solv**
	
	source = (my_rank + 1) % comm _sz;
	dest = (my_rank - 1 + comm_sz) % comm_sz;
	Copy_loc_pos_into_tmp pos;
	loc_forces = tmp_forces = 0;
	
	Compute forces due to interactions among local particles;
	for (phase = 1; phase < comm sz; phase++) {
	Send current tmp_pos and tmp_forces to dest;
	Receive new tmp_pos and tmp_forces from source;
	/_ Owner of the positions and forces we're receiving _/
	owner = (my_rank + phase) % comm_sz;
	Compute forces due to interactions among my particles
	and owner's particles;
	}
		C409.5	BTL1

	15 Send current tmp_pos and tmp_forces to dest; 16 Receive new tmp_pos and tmp_forces from source;		
39	**What are the two phases for computation of forces?** The following choices with respect to the data structures: Each process stores the entire global array of particle masses. Each process only uses a single n-element array for the positions. Each process uses a pointer loc_pos that refers to the start of its block of pos.	C409.5	BTL1

40	**Write the pseudo code for an implementation of a depth-first solution to TSP using recursion?**
	
	for (city = n-1; city >= 1; city--)
	Push(stack, city);
	while (!Empty(stack)) {
	city = Pop(stack);
	if (city == NO CITY) // End of child list, back up
	Remove last city(curr tour);}
	else {
	Add city(curr tour, city);
	if (City count(curr tour) == n) {
	if (Best tour(curr tour)) {
	Update best tour(curr tour);
	Remove last city(curr tour);
	} else {
	Push(stack, NO CITY);
	for (nbr = n-1; nbr >= 1; nbr--)
	if (Feasible(curr tour, nbr))
	Push(stack, nbr);
	}
	}/_ if Feasible _/
	} /_ while !Empty _/
		C409.5	BTL1

41	**How the function Push_copy is used in TSP**	C409.5	BTL1

	It is necessary to push onto the stack to create a copy of the tour before actually pushing it on to the stack using the function Push _copy. The extra memory is required to allocating storage for a new tour and copying the existing tour is time-consuming. Reduce the costs by saving freed tours in our own data structure, and when a freed tour is available we can use it in the Push _copy function instead of calling malloc.		
42	**What are the algorithms for identifying which subtrees we assign to the pr threads** ➢ depth-first search ➢ breadth-first search	C409.5	BTL1
43	**Define the term POSIX or PThreads** Pthreads are libraries of type definitions, functions, and macros that can be used in C progran a standard for Unix-like operating systems—for example, Linux and Mac OS X. It specifies facilities that should be available in such systems. In particular, it specifies an application pr interface (API) for *multithreaded* programming. Pthreads is not a programming language (su Java). Rather, like MPI, Pthreads specifies a *library* that can be linked with C programs. Unl Pthreads API is only available on POSIX systems—Linux, Mac OS X, Solaris, HPUX, and s	C409.5	BTL1
44	**What are the reason for parameter threads_in_cond_wait used in Tree search?** There are also two cases to consider: o threads _in_cond_wait < thread_count, it tells us how many threads are wait o threads_in_cond_wait == thread count,all the treads are out of work, and its	C409.5	BTL1
45	**What are the global variables for Recursive depth first search?** n: the total number of cities in the problem . digraph: a data structure representing the input digraph . hometown: a data structure representing vertex or city 0, the salesperson's home tour: a data structure representing the best tour so far.	C409.5	BTL1
46	**Mention the performance of MPI solvers** The performance of the reduced solver is much superior to the performance of the basic solver, although the basic solver achieves higher efficiencies. A point to stress here is that the reduced MPI solver makes much more efficient use of memory than the basic MPI solver; the basic solver must provide storage for all *n* positions on each process, while the reduced solver only needs extra storage for *n/*commsz positions and *n/*commsz forces.	C409.5	BTL1
47	**Mention the principal data structures on pthread** The vectors are two-dimensional arrays of **double**s, and the mass, position, and	C409.5	BTL1

	velocity of a single particle are stored in a struct. The forces are stored in an array of vectors.		
48	**Name any two OpenMp environment variables (Nov/Dec 2018)** **omp_set_num_threads(*num_threads*)** **omp_get_num_threads()** **omp_get_max_threads()** **omp_get_thread_num()**	C409.5	BTL1
49	**List any two scoping clauses in OpenMP (Nov/Dec 2018)** • Shared Variables • Private Variables	C409.5	BTL1
50	**What are the reason for parameter threads_in_cond_wait used in Tree search?** there are also two cases to consider: • threads _in_cond_wait < thread_count, it tells us how many threads are waiting • threads_in_cond_wait == thread count,all the treads are out of work, and its time to quit.	C409.5	BTL1

PART-B

Q. No.	Questions	CO	Bloom's Level
1.	**1.Explain n-Body solvers in OpenMP.** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:271-297	C409.5	BTL5
2.	**Explain about Tree Search.** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:229-318	C409.5	BTL5
3.	**Explain OpenMP implementations in detail.** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:15-20	C409.5	BTL5

4.	**Explain MPI implementations in detail.** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No: 86-88	C409.5	BTL4
5.	**Compare OpenMP and MPI implementations.** **Refer Notes**	C409.5	BTL4
6.	**Explin how to Parallelizing the tree search in detail.** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No: 306-308	C409.5	BTL5
7.	**i.How to parallelize the basic solver using MPI. .(Apr/May2017)** **ii.Explain Non recursive depth first search. .(Apr/May2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No: 316-317 & 303-304	C409.5	BTL5
8.	**Explain the implementation of tree search using MPI and dynamic partitioning. .(Apr/May2017)** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No:229-318	C409.5	BTL5
9	**What does the n-body problem do/give the pseudocode for serial n-body solver and for computing n-body forces. .(Nov/Dec 2017)**. 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No: 271-297	C409.5	BTL1
10	**How will you parallelize the reduced solver using Open Mp? How will you parallelize the reduced solver using Open MP? .(Nov/Dec 2017)**. 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011 Page No: 271-297	C409.5	BTL1
11	**Generalize about the two Serial program** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.5	BTL6
12	**Describe about the Parallelizing the reduced solver using OpenMP** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.5	BTL2

13	**Describe about the Recursive and non recursive DFS** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.5	BTL2
14	**Examine the Data structures for the serial implementation** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.5	BTL3
15	**Express detail about the static parallelizing of tree search using PThreads** 1. Peter S. Pacheco, "An Introduction to Parallel Programming", Morgan-Kauffman/Elsevier, 2011	C409.5	BTL1