Roll No.												

PRESIDENCY UNIVERSITY
BENGALURU

SCHOOL OF ENGINEERING

TEST 1

Sem & AY: Odd Sem. 2019-20
Course Code: CSE 305
Course Name: PARALLEL COMPUTING
Program & Sem: B.Tech (CSE) & V DE

Date: 27.09.2019
Time: 11:00AM to 12PM
Max Marks: 40
Weightage: 20%

Instructions: i) Non- Programmable calculators are allowed
ii) This question paper contains three parts. Read the question paper carefully and answer the following questions

Part A

Answer all the Questions. Each question carries five marks **(4Qx5M=20M)**

1) Identify the need for ever-increasing Processor Performance. [5M]
(C.O.NO. 1) [Knowledge]

2) Describe the bottle-necks in the Von Newmann Model with the help of an Analogy.
[5M]
(C.O.NO.1) [Knowledge]

3) Define with suitable examples i) A Process ii) Threads. [5M]
(C.O.NO.1) [Knowledge]

4) Describe Data Parallelism and Functional Parallelism using examples. [5M]
(C.O.NO.1) [Knowledge]

Part B

Answer the Question. The Question carries eight marks. **(1Qx8M=8M)**

5) Explain how very large programs that access large data sets which do not fit into main memory are executed. [8M]

(C.O.NO.1) [Comprehension]

Part C

Answer all the Question. The Question carries twelve marks. **(1Qx12M=12M)**

6) Consider a processor operating at 1 GHz connected to a DRAM with a latency of 100ns.(no caches). Assume that the processor has two multiply-add units and is capable of executing 4 instructions in each cycle of 1ns. Block size=1 word and each element of the vector/matrix is stored as a word.

Calculate the following. [12M]

(C.O.NO.1) [Application]

i) Peak Processor Performance [1M]

ii) Speed of computation for obtaining the dot-product of two vectors [5M]

iii) Speed of computation for the Multiplication of two Matrices when a Cache memory (32KB) having a latency of 1ns is added to the Processor. Assume that the cache is capable of storing Matrix A of size 32 x 32, B of size 32 x 32 and the resultant Matrix. Also assume ideal cache placement strategy in which no data item is overwritten by others. [6M]

****

SCHOOL OF ENGINEERING

Year -:2019 - 20
Course Code: CSE 305
Course Name: Parallel Computing
Program & Sem: B.Tech, 5

Date 27/9/19
Time: 11 am -12 pm
Max Marks 40
Weightage: 20%

Extract of question distribution [outcome wise & level wise]

Q.NO	C.O.NO	Unit/Module Number/Unit /Module Title	Memory recall type [20M] Bloom's Levels			Thought provoking type [8M] Bloom's Levels			Problem Solving type [12M]			Total Marks
			K			C			A			
1	1	Module-1	5									5
2	1	Module-2	5									5
3	1	Module-2	5									5
4	1	Module-1	5									5
5	1	Module-2	3			5						8
6	1	Module-2	1			3			6	2		12
	Total Marks		24			8			6	2		40

K =Knowledge Level C = Comprehension Level, A = Application Level

Note: While setting all types of questions the general guideline is that about 60%

Of the questions must be such that even a below average students must be able to attempt, About 20% of the questions must be such that only above average students must be able to attempt and finally 20% of the questions must be such that only the bright students must be able to attempt.

[I hereby certify that All the questions are set as per the above guide lines. Ms. Sudhakamaraju]

Reviewers' Comments

Annexure- II: Format of Answer Scheme

SCHOOL OF ENGINEERING

SOLUTION

Year -:2019 - 20
Course Code: CSE 305
Course Name: Parallel Computing
Program & Sem: B.Tech, 5

Date: 27/9/19
Time: 11 am -12 pm
Max Marks: 40
Weightage: 20%

Part A (4Qx5M=20)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
1	i) Many problems like Climate Modelling, Protein folding, Drug Discovery, energy Research, data Analysis etc. was not possible because of the less computational power available then. ii) Super computers are very expensive (but with the drop in the cost of microprocessors, parallel systems became popular and supplied the high performance requirements) i) Processor - Memory Interconnect ii) Location of the Instructions and Data with respect to the processor.	Expanding each point (1M for each)	8
2	Process – An instance of a program in execution	Expand each point and explain analogy (2) +(3)	8

	Threads are independent parts of a program that can be executed simultaneously. + examples			

3	Data Parallelism: Independent tasks applying the same operation to different elements of a data set.
	
	For i = 0 to 99 do
	  a[i]= b[i]+c[i]
	end for
		1+1+2+1		5

4	Functional Parallelism: Independent tasks applying different operations to different elements of a data set.
	a=2
	b=3
	m=(a + b)/2
	s=(a^2 * $b^{2)}$/2
	v=s – m^2
		2.5 + 2.5		8

Part B (1Qx8M=8)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
5	Virtual Memory- main mem acts as a cache for the secondary memory. Exploiting spatial and temporal locality.- holds only active parts of many running programs-swap space-pages- page hit/miss-virtual addresses assigned during compilation-address translation required-page table-mapping between the physical address of a page and the location in main memory. Virtual address= virtual page number+ byte offset. – TLB-TLB hit/miss	Virtual Memory (2M) Swap -space -Page - Table-Virtual add - translation(4) TLB -(2M)	15

Part C (Q x M = Marks)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
6	i) Peak Processor Performance=4/1nsec=40GFLOPS ii) dot-product calculation	i) 1M ii) 3M iii) 8 M (2+2+2+2)	15

	Performance=2/200nsec=1/100ns =10MFLOPS iii) Matrix Mul- Performance=no: of operations/time taken No: of operations =$2n^3$ (n=32) =64K operations Time taken = Time to access the elements of A and B in to cache + computation time . Total access time =2K x 100nsec=200×10^{-6} sec. Total computation time = 1×10^{-9} x 64K/4=16×10^{-6} sec Total execution time =200+16 micro sec Performance = $2 \times 32^{3} / 216 \times 10^{-6}$ = 303 MFLOPS		

Roll No.													

PRESIDENCY UNIVERSITY
BENGALURU

SCHOOL OF ENGINEERING

TEST – 2

Sem & AY: Odd Sem. 2019-20	**Date**: 16.11.2019
Course Code: CSE 305	**Time**: 11:00 AM to 12:00 PM
Course Name: PARALLEL COMPUTING	**Max Marks**: 40
Program & Sem: B.Tech (CSE) & V	**Weightage**: 20%

Instructions:

I. Non- Programmable calculators are allowed

II. This question paper contains three parts. Read the question paper carefully and answer the following questions

Part A [Memory Recall Questions]

Answer all the Questions. Each Question carries five marks. (4Qx5M=20M)

1. Write an efficient C program that will create 4 threads and
   a. Print the message – "Hello World" from each thread along with their respective thread ids [3M]
   b. Print the number of processors available in the system for execution. [1M]
   c. Print the total number of threads. [1M]
   (C.O.NO.3) [Knowledge]
2. Describe the characteristics of a Vector processor. (C.O.NO.1) [Knowledge]
3. Define with suitable examples i) Private Clause ii) Critical Pragma (C.O.NO.3) [Knowledge]
4. Sketch a complete Omega Network that connects 8 processors and 8 Memory Banks and show the route taken from processor 5 to memory bank 1. (C.O.NO.1) [Knowledge]

Part B [Thought Provoking Questions]

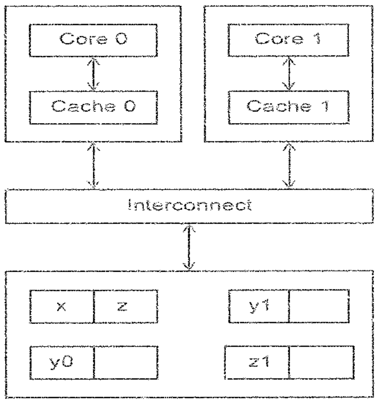
Answer the Question. Each Question carries eight marks. (1Qx8M=8M)

5. Explain the effect of strided memory access on Processor Performance using a code fragment as an example. Also, Illustrate how to overcome the problem using suitable diagrams (C.O.NO.1) [Comprehension]

Part C [Problem Solving Questions]

Answer the Question. The Question carries twelve marks. **(1Qx12M=12M)**

6. Consider a Shared Memory System with 2 cores and 2 caches as shown below.



Time	Core 0	Core 1
0	`y0 = x;`	`y1 = 3*x;`
1	`x = 7;`	Statement(s) not involving x
2	Statement(s) not involving x	`z1 = 4*x;`

Core 0 and Core1 execute the above statements at the given time.

(C.O.NO.1) [Application]

Variable x is shared by the cores and variable y_0 is private to core0 and variables y_1 and z_1 are private to core1.

a) Compute the value of x at time 3 [1M]

b) Identify and explain any possible unpredictable behavior in the above scenario. [3M]

c) Describe two Solutions to overcome this behavior. [8M]

SCHOOL OF ENGINEERING

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

Year -:2019 - 20
Course Code: CSE 305
Course Name: Parallel Computing
Program & Sem: B.Tech, 5

Date: 16/11/19
Time: 11 am -12 pm
Max Marks: 40
Weightage: 20%

Extract of question distribution [outcome wise & level wise]

Q.NO.	C.O.NO	Unit/Module Number/Unit /Module Title	Memory recall type [20M] Bloom's Levels			Thought provoking type [8M] Bloom's Levels			Problem Solving type [12M]			Total Marks
			K			C			A			
1	3	Module-4	5									
2	1	Module-2	5									5
3	3	Module-4	5									5
4	1	Module-1	5									5
5	1	Module-2				8						5
6	1	Module-2							12			8
	Total Marks		20			8			12			12 40

K =Knowledge Level C = Comprehension Level, A = Application Level

Note: While setting all types of questions the general guideline is that about 60%

Of the questions must be such that even a below average students must be able to attempt, About 20% of the questions must be such that only above average students must be able to attempt and finally 20% of the questions must be such that only the bright students must be able to attempt.

Annexure- II: Format of Answer Scheme

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

SCHOOL OF ENGINEERING

SOLUTION

Year -:2019 - 20

Course Code: CSE 305

Course Name: Parallel Computing

Program & Sem: B.Tech, 5

Date: 27/9/19

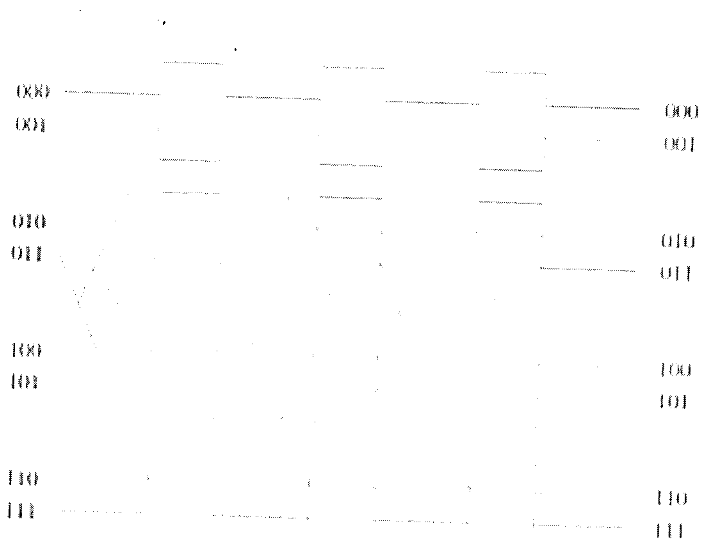
Time: 11 am -12 pm

Max Marks: 40

Weightage: 20%

Part A **(4Qx5M=20)**

Q No	Solution	Scheme of Marking	Max. Time required for each Question
1	#include<omp.h> …… #pragma omp parallel {printf("Hello World from thread %d ",omp_get_thread_num());} printf(" the number of processors are %d",omp_get_num_proc(); printf(" the total number of threads are %d",omp_get_num_threads(); ……..	3+1+1	5
2	1)vector registers 2) Vectorized pipelined functional units		

	3) Vector Instructions 4) Interleaved Memory 5) Strided Memory access	Expand each point – 1 mark for each feature	8
3	Private clause is used to declare private variables – expand eg # pragma omp parallel (default shared) (private i,j) Critical pragma – defines the critical region that makes the operations on shared variables serial.	2.5+2.5	8
4	 000 001 010 011 100 101 110 111 000 001 010 011 100 101 110 111 processor 5- stage1 (switch 2- cross) – stage2 (switch 3- cross)- 1 (memory bank)	4	8

Part B (1Qx8M=8)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
5	Array – stored in row major order but data is accessed in column major order –in strided access.	4+4	8

	`for (i = 0; i < 1000; i++)`
	`        column_sum[i] = 0.0;`
	`        for (j = 0; j < 1000; j++)`
	`                column_sum[i] += b[j][i];`
	
	Consider the following restructuring of the column-sum fragment:
	
	`for (i = 0; i < 1000; i++)`
	`        column_sum[i] = 0.0;`
	`for (j = 0; j < 1000; j++)`
	`        for (i = 0; i < 1000; i++)`
	`                column_sum[i] += b[j][i];`
	In Large arrays we overcome using Tiling.		

Part C (Q x M = Marks)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
6	a) undefined b) Cache Coherence Note that this unpredictable behavior will occur regardless of whether the system is using a write-through or a write-back policy. If it's using a write-through policy, the main memory will be updated by the assignment x = 7. However, this will have no effect on the value in the cache of core 1. If the system is using a write-back policy, the new value of x in the cache of core 0 probably won't even be available to core 1 when it updates z1. Clearly, this is a problem. The programmer doesn't have direct control over when the caches are updated, so her program cannot execute these apparently innocuous statements and know what will be stored in z1. There are several problems here, but the one we want to look at right now is that the caches we described for single processor systems provide no mechanism for insuring that when the caches of multiple processors store the same variable, an update by one processor to the cached variable is "seen" by the other processors. That is, that the cached value stored by the other processors is also updated. This is called the **cache coherence** problem. c) Snoopy cache and Directory based cache There are two main approaches to insuring cache coherence: **snooping cache coherence** and directory-based cache coherence. The idea behind snooping comes from bus-based systems: When the cores share a bus, any signal transmitted on the bus can be "seen" by all the cores connected to the bus. Thus, when core 0 updates the copy of x stored in its cache, if it also broadcasts this information across the bus, and if core 1 is "snooping" the bus, it will see that x has been updated and it can mark its copy of x as invalid. This is more or less how snooping cache coherence works. The principal difference between our description and the actual snooping protocol is that the broadcast only informs the other cores that the *cache line* containing x has been updated, not that x has been updated.	a) 1M b) 3M c) 8 M	12

	Directory based cache – the system maintains a directory of information about each cache line – using presence bits that is used to identify the processor that updates a shared variable and references to the shared variable will be served by this processor.		

Roll No												

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY
BENGALURU

SCHOOL OF ENGINEERING

END TERM FINAL EXAMINATION

Semester: Odd Semester: 2019 - 20	**Date**: 20 December 2019
Course Code: CSE 305	**Time**: 9:30 AM to 12:30 PM
Course Name: PARALLEL COMPUTING	**Max Marks**: 80
Program & Sem: B.Tech (CSE) & V (DE-I)	**Weightage**: 40%

Instructions:
(i) Read the all questions carefully and answer accordingly.

Part A [Memory Recall Questions]

Answer all the Questions. Each Question carries 5 marks. (6Qx5M=30M)

1. Explain the two different types of parallelism with examples (C.O.No.1) [Knowledge]
2. Describe how the addition of a cache memory will help in improving Processor Performance with an example (C.O.No.1) [Knowledge]
3. Sketch a complete Omega Network that connects 8 processors and 8 Memory Banks and show the route taken from processor 3(011) to memory bank 7 (111)

   (C.O.No.1) [Knowledge]
4. Explain Ian Foster's Design methodology for parallel algorithm design

   (C.O.No.2) [Knowledge]
5. State Amdahl's law List the formulae for the calculation of Speedup and Efficiency for a parallel System. (C.O.No.1) [Knowledge]
6. Write the Syntax for the MPI functions MPI_Init and MPI_Finalize .Describe the functions and their arguments. (C.O.No.3) [Knowledge]

Part B [Thought Provoking Questions]

Answer all the Questions. Each Question carries 14 marks. (3Qx14M=42M)

7. a) i) Use OpenMP pragmas to make the following loop parallel and explain how it is executed in parallel.

```
for( i=0; i < n; i++)
   a[ i ] = foo( i ) ;
```

[4M] (C.O.No.3) [Comprehension]

ii)

[4M] (C.O.No.3) [Comprehension]

```
for( i=0; i < n; i++) {
      a[i] = foo(i) ;
      if( a[ i ] < b [ i ])
          break ;
}
```

b. Describe the circumstances under which the “firstprivate” and “lastprivate “ clauses are used in parallel programming using OpenMP- explain how they can be used with suitable examples

[6M] (C.O.No.3.) [Comprehension]

8. Mr. Smith wants to add the data stored in an array **a of size N** (where N > 10,000) on a system that has 4 processors and default number of threads is 4. All the elements of the array are initialized to 1. He has written the code segment given below and obtained the output 130796.

[14M] (C.O.No.3.) [Comprehension]

```
#include<stdio.h>
#include<omp.h>
#define SIZE 40000

void main(){

int  a[SIZE], i , sum=0;

for(i=0 ; i<SIZE ; ++)
    a[i]=1;

#pragma omp parallel
{
 for(i=0; i<SIZE; i++)
   sum = sum + a[i];
}
print("sum is %d \n",sum);
}
```

i) What is the expected output?
ii) Explain what modification should be done to get the expected output?

9. a) Write a complete C program using MPI [4M] (C.O.No.3) [Application]

i) that will print the message – “Parallel Computing” from each process along with their respective process ids

ii) that will print the total number of processes

iii) Give the command line for the compilation and execution of your program by creating 4 processes

b) Apply the steps of Foster's Design algorithm to the boundary value problem of calculating the temperature at any point of the rod at any point in time given that the rod is made up of uniform material, surrounded by a blanket of insulation and the ends of the rod placed in an ice bath having temperature 0^{o} C.

[10M] (C.O.No.2) [Application]

Part C [Problem Solving Questions]

Answer the Question. The Question carries 8 marks. **(1Qx8M=8M)**

10. Suppose you are given a Parallel System with 3 cores and a serial program with $T_{serial}=20$ seconds. Further suppose that 90% of this serial program is parallelized.

  i) Compute the overall parallel-runtime ($T_{parallel}$) and Speedup under ideal conditions.

  ii) Explain how Speedup is affected when the number of cores are very large.

(C.O.No.3.) [Application]

Set-2

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

SCHOOL OF ENGINEERING

END TERM FINAL EXAMINATION

Extract of question distribution [outcome wise & level wise]

Q.NO.	C.O.NO (% age of CO)	Unit/Module Number/Unit /Module Title	Memory recall type [Marks allotted] Bloom's Levels	Thought provoking type [Marks allotted] Bloom's Levels	Problem Solving type [Marks allotted]	Total Marks
			K	C	A	
1	1	1	5			5
2	1	2	5			5
3	1	2	5			5
4	2	3	5			5
5	1	3	5			5
6	3	4	5			5
7	3	4		14		
8	3	4		14		
9	2	3			14	
10	1	3			8	
	Total Marks		30	28	22	80

K =Knowledge Level C = Comprehension Level, A = Application Level

Note: While setting all types of questions the general guideline is that about 60%

Of the questions must be such that even a below average students must be able to attempt, About 20% of the questions must be such that only above average students must be able to attempt and finally 20% of the questions must be such that only the bright students must be able to attempt.

I hereby certify that all the questions are set as per the above guidelines.

Faculty Signature: [illegible]

Reviewer Commend: Part-B. 7th Q: ~~[illegible]~~ the students to identify the problem. inputs given to the faculty member

Format of Answer Scheme

SCHOOL OF ENGINEERING

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

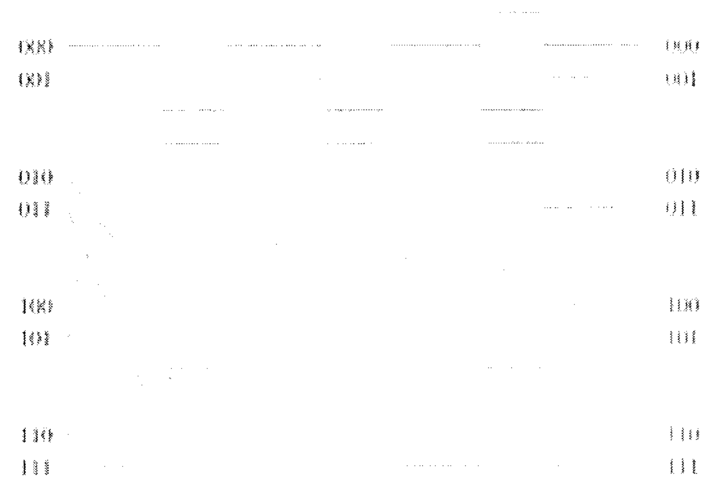
SOLUTION

Semester: Odd Sem. 2019-20
Course Code: CSE 305
Course Name: Parallel Computing
Program & Sem: B.Tech (CSE) & V

Date: 20 Dec 2019
Time: 9:30 AM to 12:30 PM
Max Marks: 80
Weightage: 40%

Part A (6Q x 5M = 30Marks)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
1	Data and Functional parallelism (dividing data) dividing the task) examples	**2.5 +2.5**	**7**
2	Using cache, time taken for the execution of a program is reduced. Explanation can be given using cache mappings- with hits and misses	**2.5 +2.5**	**7**

3	 000 001 010 011 100 101 110 111 000 001 010 011 100 101 110 111 3 to 7 (011 xor 111) =100 Route from 3 to 6, 6 to 7 , 7 to 7.	**4 +1**	**7**
4	Foster's design methodology – Partitioning, Communication, Agglomeration and Mapping	**5**	**7**
5	**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 **speedup** of a parallel program t $$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}},$$ $$E = \frac{S}{p} = \frac{\left(\frac{T_{\text{serial}}}{T_{\text{parallel}}}\right)}{p} = \frac{T_{\text{serial}}}{p \cdot T_{\text{parallel}}}.$$	**3+1+1**	**5**

6	The first MPI function call made by every MPI process is the call to `MPI_Init`, which allows the system to do any setup needed to handle further calls to the MPI library. The call to `MPI_Init` does not have to be the first executable statement of the program. In fact, it does not even have to be located in function `main`. The only requirement is that `MPI_Init` be called before any other MPI function.[1]
	Note that all MPI identifiers, including function identifiers, begin with the prefix `MPI_`, followed by a capital letter and a series of lowercase letters and underscores. All MPI constants are strings of capital letters and underscores beginning with `MPI_`.
	
	`MPI_Init (&argc, &argv);`
	
	After a process has completed all of its MPI library calls, it calls function `MPI_Finalize`, allowing the system to free up resources (such as memory) that have been allocated to MPI.
	
	`MPI_Finalize();`
	`return 0;`
	`}`
		Each 1	**8**

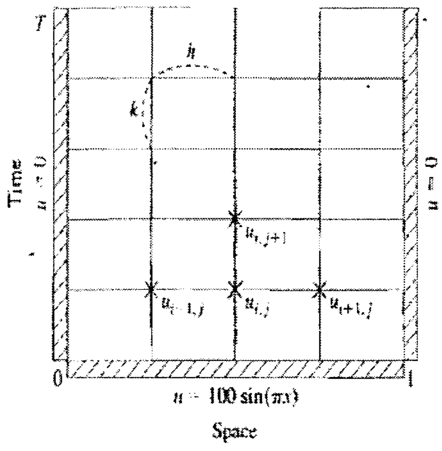
	Function `check_circuit`, passed the ID number of a process and an integer `z`, first extracts the values of the 16 inputs using the macro `EXTRACT_BITS`. Element `v[0]` corresponds to input `a`, element `v[1]` corresponds to input `b`, and so on. Calling function `check_circuit` with values of `z` ranging from 0 through 65,535 generates all 2^{16} combinations of values. After function `check_circuit` has determined the values of the 16 inputs, it checks to see if they result in the circuit having the output 1. If so, the process prints the values of `a` through `p`.		

Part B (3Q x 14M = 42 Marks)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
7	a) The loop can be executed parallel by using #pragma omp parallel for – describe the parallel region b) it cannot be executed in parallel as it has a break statement and the no: of iterations cannot be known in advance required by the run time env. c) Critical –avoid racing Single – when statements are to be executed by only one process- explain ,example	4+4+6	12

8	#pragma omp for
	for(i=0;i<SIZE;i++)
	psum=psum+a[i];
	
	//synchronize to sum up the partial sum
	#pragma omp critical
	sum=sum+psum;
		Syntax – 4
		
		Code + explanation- 10
			15

9	`#include <stdio.h>`
	`#include <mpi.h>`
	
	`int main(void) {`
	`int my_rank, comm_sz;`
	
	`MPI_Init(NULL, NULL);`
	`MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);`
	`MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);`
	
	printf(“ parallel computing from process %d of %d”,my_rank,comm_sz);
	
	`MPI_Finalize();`
	`return 0;`
	`} /* main */`
	
	b)
		4
		
		10
			10
			
			15

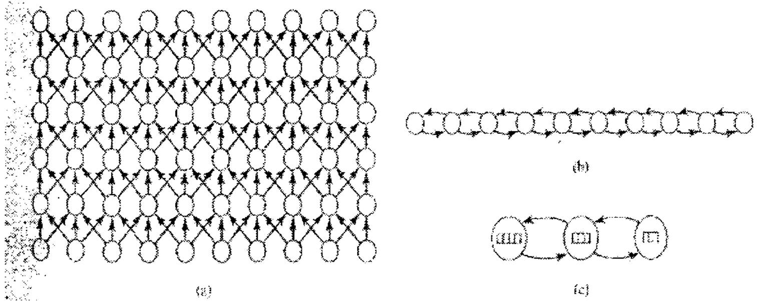


Our first step is partitioning. In this case the data being manipulated are easy to identify: there is one data item per grid point. To start, let's associate one primitive task with each grid point. This yields a two-dimensional domain decomposition.

3.4.3 Communication

Now that we have identified our tasks, we need to determine the communication pattern between the tasks. If task A needs a value from task B to perform its computation, we need to draw a channel from task B to task A. Since the task computing $u_{i,j+1}$ requires the values of $u_{i-1,j}$, $u_{i,j}$, and $u_{i+1,j}$, in general each task will have three incoming channels and three outgoing channels (see Figure 3.11a). The tasks on the edges have fewer channels.

b)



	Even if enough processors were available, it would be impossible to compute every task shown in Figure 3.11a concurrently, because the tasks computing rod temperatures later in time depend upon the results produced by tasks computing rod temperatures earlier in time. This is made plain by the vertical paths of channels stretching from the bottom tasks to the top tasks. There is no point in maintaining the illusion of multiple tasks when they must be performed sequentially. Let's agglomerate all the tasks associated with each point in the rod, that is, tasks in the same column in Figure 3.11a. The resulting task/channel graph, shown in Figure 3.11b, is much less complicated. Now we have a linear array of tasks, each communicating solely with its neighbor(s). Each is responsible for computing the temperature at a particular grid point for all time steps. However, even this graph is likely to have far more tasks than we need to keep all of our processors fully occupied, since in a real problem the number of rod segments would be large. We can use the decision tree of Figure 3.7 to come up with a mapping strategy. The number of tasks is static (left branch), the communication pattern among them is regular (left branch), and each task performs the same computations (left branch). Hence a good strategy is to create one task per processor, agglomerating primitive tasks so that computational workloads are balanced and communication is minimized. Associating a contiguous piece of the rod with each task (Figure 3.11c) preserves the simple nearest-neighbor communication between tasks and eliminates unnecessary communications for those data points within a single task.		

Part C (1Q x 8M = 8Marks)

Q No	Solution	Scheme of Marking	Max. Time required for each Question
10	cores available. Suppose, for example, that we're able to parallelize 90% of a serial program. Further suppose that the parallelization is "perfect," that is, regardless of the number of cores p we use, the speedup of this part of the program will be p. If the serial run-time is T_{serial}= 20 seconds, then the run-time of the parallelized part will be $0.9 \times T_{serial}/p = 18/p$ and the run-time of the "unparallelized" part will be $0.1 \times T_{serial} = 2$. The overall parallel run-time will be $$T_{parallel} = 0.9 \times T_{serial}/p + 0.1 \times T_{serial} = 18/p + 2,$$ and the speedup will be $$S = \frac{T_{serial}}{0.9 \times T_{serial}/p + 0.1 \times T_{serial}} = \frac{20}{18/p + 2}.$$ Now as p gets larger and larger, $0.9 \times T_{serial}/p = 18/p$ gets closer and closer to 0, so the total parallel run-time can't be smaller than $0.1 \times T_{serial} = 2$. That is, the denominator in S can't be smaller than $0.1 \times T_{serial} = 2$. The fraction S must therefore be smaller than $T_{serial}/2$	2+2+4	8