USN										

Internal Assessment Test 2 – Nov. 2025

Subject/Code : Parallel computing/ BCS702						
Date: 28/11/25	Duration: 90 mins	Max. Marks:50	Semester: 07	Branch: CSE		
.	**Answer any FIVE FULL Questions**			**Marks**	**CO**	**RBT**
1	Explain four different techniques used to ensure an operation is atomic			[10]	1	L2
2	Describe the different classes of parallel computers based on Flynn's taxonomy			[10]	1	L2
3a)	A sequential program takes 40 seconds to execute, while its parallel version takes 10 seconds using 4 processors. Determine the efficiency of the parallel program and the parallel overhead.			[6]	2	L3
3b)	Explain when a parallel program is strongly scalable or weakly scalable.			[4]	2	L2
4a)	Explain speedup, efficiency and the effect of the number of processes on speedup and efficiency of a parallel program.			[6]	2	L2
4b)	Explain scalability using the formula of efficiency.			[4]	5	L2
5a)	Explain difference between '__host__' and '___device__'.			[2]	5	L2
5b)	Explain the different built-in struct variables used to get information on the grid started by the kernel			[8]	5	L2
6a)	Differentiate cudaMallocManaged and cudaMalloc in terms of syntax and purpose			[4]	5	L2

6b)	Write the kernel function definition for parallelizing trapezoidal rule I, given the following serial version for reference. Show which part(s) can be executed on the host.
	float Serial_trap(const float a, const float b, const float n){
	float x, h = (b-a)/n;
	float trap = 0.5*(f(a)+f(b));
	for (int i = 1; I <= n-1; i++){
	x = a + i* h;
	trap += f(x);
	}
	
	return trap;
	}
				[6]	5	L3

1. Explain four different techniques used to ensure an operation is atomic. [10 marks]

Listing each technique- 0.25 mark each – total 1 mark
Explanation of each technique – 2 marks each

- Mutual exclusion lock or mutex,or lock.
- Busy-waiting

- Semaphores
- Monitor

2. Describe the different classes of parallel computers based on Flynn's taxonomy. [10 marks]

Listing each class- 0.25 mark each – total 1 mark
Explanation of each class – 2 marks each

- SISD
- SIMD
- MISD
- MIMD

3a) A sequential program takes 40 seconds to execute, while its parallel version takes 10 seconds using 4 processors. Determine the efficiency of the parallel program and the parallel overhead. [6 marks]
Efficiency – 3 marks
parallel overhead – 3 marks
Given

Sequential execution time: $T_s = 40$ seconds $T_s = 40$ seconds

Parallel execution time: $T_p = 10$ seconds $T_p = 10$ seconds

Number of processors: $p = 4$ $p = 4$

Speedup

$$S = \frac{T_s}{T_p} = \frac{40}{10} = 4$$

Efficiency measures how well the processors are utilized:

$$E = \frac{S}{p} = \frac{4}{4} = 1.0$$

Parallel overhead is the extra time spent due to communication, synchronization, or other parallelization costs. It's defined as:

$$T_o = p \cdot T_p - T_s$$

$$T_o = 4 \cdot 10 - 40 = 40 - 40 = 0$$

3b) Explain when a parallel program is strongly scalable or weakly scalable. [4 marks]
strongly scalable – 2 marks
weakly scalable – 2 marks

If, when we increase the number of processes/threads, we can keep the efficiency fixed without increasing the problem size, the program is said to be strongly scalable.

If we can keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, then the program is said to be weakly scalable.

4a) Explain speedup, efficiency and the effect of the number of processes on speedup and efficiency of a parallel program.
speedup– 2 mark
efficiency – 2 mark
effect of the number of processes on speedup – 1 mark
effect of the number of processes on efficiency – 1 mark

Speedup of a parallel program

$S = T_{serial} / T_{parallel}$

As p increases, speedup increases.

S/p, is called the efficiency of the parallel program.
If we substitute the formula for S, we see that the efficiency is

$$E = \frac{S}{p} = \frac{\left(\frac{T_{\text{serial}}}{T_{\text{parallel}}}\right)}{p} = \frac{T_{\text{serial}}}{p \cdot T_{\text{parallel}}}.$$

As p increases, efficiency decreases.

5a) Explain difference between ‘ __host__ ’ and ‘ __device__’. [2 marks]

Qualifier	Runs On	Called From
__device__	GPU	GPU
__host__	CPU	CPU

5b) Explain the different built-in struct variables used to get information on the grid started by the kernel [8 marks]
variable and its description- 2 marks each
The following four variables are structs that are initialized in each thread’s memory when a kernel begins execution:

- threadIdx: the rank or index of the thread in its thread block
- blockDim: the dimensions, shape, or size of the thread blocks
- blockIdx: the rank or index of the block within the grid
- gridDim: the dimensions,shape,or size of the grid

6a) Differentiate cudaMallocManaged and cudaMalloc in terms of syntax and purpose. [4 marks]
syntax – 1 mark each
purpose– 1 mark each
cudaMallocManaged is a CUDA API function that allocates Unified Memory, a single memory space accessible by both the CPU (host) and GPU (device).

```
__host__ cudaError_t cudaMallocManaged (
             void**    devPtr   /* out */,
             size_t    size     /* in  */,
             unsigned  flags    /* in  */);
```

- The __host__ qualifier is a CUDA addition to C, and it indicates that the function should be called and run on the host. This is the default for functions in CUDA
- The return value, which has type cudaError_t, allows the function to return an error.
- The first argument is a pointer to a pointer: it refers to the pointer that’s being allocated.
- The second argument specifies the number of bytes that should be allocated.
- The flags argument controls which kernels can access the allocated memory. It defaults to cudaMemAttachGlobal and can be omitted.

When unified memory isn't available, use the CUDA function cudaMalloc for the memory allocation on the device.

```
__host__ __device__ cudaError_t cudaMalloc (
            void**  dev_p   /* out */,
            size_t  size    /* in  */);
```

- The first argument is a reference to a pointer that will be used on the device.
- The second argument specifies the number of bytes to allocate on the device.

6b) Write the kernel function definition for parallelizing trapezoidal rule I, given the following serial version for reference. Show which part(s) can be executed on the host.

```
float Serial_trap(const float a, const float b, const float n){
  float x, h = (b-a)/n;
  float trap = 0.5*(f(a)+f(b));
  for (int i = 1; I <= n-1; i++){
     x = a + i* h;
     trap += f(x);
  }

return trap;
}
```

[6 marks]

kernel function definition for parallelizing trapezoidal rule I- 4 marks
part(s) can be executed on the host – 2 marks

kernel function definition for parallelizing trapezoidal rule I

```
1  __global__ void Dev_trap(
2        const float  a       /* in      */,
3        const float  b       /* in      */,
4        const float  h       /* in      */,
5        const int    n       /* in      */,
6        float*       trap_p  /* in/out  */) {
7     int my_i = blockDim.x * blockIdx.x + threadIdx.x;
8
9     /* f(x_0) and f(x_n) were computed on the host.  So */
10    /* compute f(x_1), f(x_2), ..., f(x_(n-1))         */
11    if (0 < my_i && my_i < n) {
12       float my_x = a + my_i*h;
13       float my_trap = f(my_x);
14       atomicAdd(trap_p, my_trap);
15    }
16 }  /* Dev_trap */
17
```

part(s) can be executed on the host

```
/* Host code */
float* trap_p;
cudaMallocManaged(&trap_p, sizeof(float));
...
```

```
*trap_p = 0.5*(f(a) + f(b));

/* Call kernel */
...

/* After return from kernel */
*trap_p = h*(*trap_p);
```