

Register number _____

SRM Institute of Science and Technology
College of Engineering and Technology
School of Computing

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamilnadu

Academic Year: 2023-24 (EVEN)

B.Tech-Computer Science & Engineering

SET – A-Answer Key

Test: CLA-T3
Date: 03.05.2024
Course Code & Title: 18CSE419T & GPU Programming
Duration: 2 periods
Year & Sem: III Year /VI Sem
Max. Marks: 50
Course articulation matrix:

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO-1	3														3
CO-2		3	2												3
CO-3		3	3												3
CO-4		3	3												3
CO-5			3	1									2		3

Part – A(1*10=10 Marks)

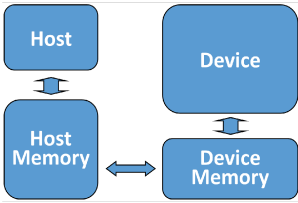
Answer All the Questions

Q. No	Questions	Marks	BL	CO	PO	PI Code
1	The lowest level of parallelism is a) Gang b) Worker c) Vector d) warp	1	1	CO 4	2 & 3	2.2.1
2	Data must be visible on the ____ when we run our ____ code a) device, parallel b) host, parallel c) device, sequential d) host, sequential	1	1	CO 4	2 & 3	2.2.1
3	Which of the following is an unstructured data directive? a) Data directive b) Enter data directive c) Kernel directive d) Exit parallel directive	1	1	CO 4	2 & 3	2.2.1
4	Which one is not a limitation of managed memory in OpenACC? a) Can transfer data asynchronously b) Memory allocation takes longer with managed memory c) Able to get better performance by manually handling data transfers d) Only available from PGI on NVIDIA GPUs	1	1	CO 4	2 & 3	2.2.1
5	The collapse(n) clause a) Turn the next n threads into one b) Break the next n loops into dependent c) Turn the next n loops into one d) Break the next n loops into independent	1	1	CO 4	2 & 3	2.2.1
6	Programming in which two or more unrelated operations can occur independently or even at the same time without immediate synchronization is a) Synchronous programming b) Asynchronous programming c) GPU programming d) High-level programming	1	1	CO 5	3	2.2.1

Register number _____

7	Auto clause in OpenACC a) Tells whether the loop is parallelizable b) Automatically executes loop code c) Sets the variables to its default value d) Tells whether the loop is safe	1	1	CO 5	3	2.2.1
8	When using the parallel directive, _____ clause is implied a) Seq clause b) Dependent clause c) Independent clause d) Auto clause	1	1	CO 5	3	2.2.1
9	The worker clause in OpenACC is a) To merge multiple smaller vector into one large vector b) To split-up one large vector into multiple smaller vectors c) To. Merge multiple smaller gang into one large gang d) To split-up one large gang into multiple smaller gangs	1	1	CO 5	3	2.2.1
10	Which data clause of OpenACC allocates memory on device but does not copy? a) Copy(list) b) Copyin(list) c) Copyout(list) d) Create(list)	1	1	CO 5	3	2.2.1
Part – B (4*4=16 marks) Answer any four Questions						
Q. N o	Question	Mark s	B L	CO	P O	PI Cod e
11	Write an OpenACC program for matrix multiplication.	4	2	CO 4	2 & 3	2.2.1
12	Write short notes on unstructured data directives of OpenACC. Enter Data Directive <ul style="list-style-type: none"> Data lifetimes aren't always neatly structured. The enter data directive handles device memory allocation You may use either the create or the copyin clause for memory allocation The enter data directive is not the start of a data region, because you may have multiple enter data directives Exit Data Directive <ul style="list-style-type: none"> The exit data directive handles device memory deallocation You may use either the delete or the copyout clause for memory deallocation You should have as many exit data for a given array as enter data These can exist in different functions copyin (list) Allocates memory on device and copies data from host to device on enter data. 	4	3	CO 4	2 & 3	2.2.1

	<p>create (list) Allocates memory on device without data transfer on enter data.</p> <p>copyout (list) Allocates memory on device and copies data back to the host on exit data.</p> <p>delete (list) Deallocates memory on device without data transfer on exit data</p>					
13	<p>Describe OpenACC memory model.</p> <ul style="list-style-type: none"> In an OpenACC memory model, the host memory and the device memory are treated as separated. It is assumed that the host is not able to access device memory directly and the device is not able to access host memory directly. This is to ensure that the OpenACC programming model can support a wide range of accelerator devices, including most of the current GPUs that do not have the capability of unified memory access between GPUs and CPUs. The unified virtual addressing and the GPUDirect introduced by NVIDIA in CUDA 4.0 allow a single virtual address space for both host memory and device memory and allow direct cross-device memory access between different GPUs. However, cross-host and device memory access is still not possible. Just like in CUDA C/C++, in OpenACC input data needs to be transferred from the host to the device before kernel launches and result data needs to be transferred back from the device to the host. However, unlike in CUDA C/ C++ where programmers need to explicitly code data movement through API calls, in OpenACC they can just annotate which memory objects need to be transferred, as shown by line 4 in Figure 15.1. The OpenACC compiler will automatically generate code for memory allocation, copying, and de-allocation. OpenACC adopts a fairly weak consistency model for memory on the accelerator device. Although data on the accelerator can be shared by all execution units, OpenACC does not provide a reliable way to allow one execution unit to consume the data produced by another execution unit. There are two reasons for this. First, recall OpenACC does not provide any mechanism for synchronization between execution units. Second, memories between different execution units are not coherent. Although some hardware provides instructions to explicitly invalidate and update cache, they are not exposed at the OpenACC level. Therefore, in OpenACC, different execution units are expected to work on disjoint memory sets. Threads within an execution unit can also share memory and threads have coherent memory. However, OpenACC currently only mandates a memory fence at the thread fork and join, which are also the only synchronizations OpenACC provides for threads. While the device memory model may appear very limiting, it is not so in practice. For data-race free OpenACC data-parallel applications, the weak memory model works quite well. 	4	3	CO 4	2 & 3	2.2.1

	 <p>Figure 1.1: OpenACC Abstract Architecture Model</p>					
14	<p>Brief on the data management process involved in OpenACC Moving data between the Host and Device using copy</p> <ul style="list-style-type: none"> Data clauses allow the programmer to tell the compiler which data to move and when Data clauses may be added to kernels or parallel regions, but also data, enter data, and exit data, which will be discussed shortly <p>This is a quick step-by-step of the copy clause. It will be presented graphically in the next slide.</p> <p>When managing data between the CPU and GPU, the CPU and GPU copy will be different. Though, you will refer to them with the same variable name. In this case, if you attempt to access array 'a' outside of the kernels region, then it will use the CPU copy. If you attempt to access array 'a' inside of the kernels region, it will use the GPU copy. This process is automatic. This is different than how CUDA functions (no need to separate device pointers)</p> <p>The "Execute Kernels" step basically means to run our loop on the GPU.</p> <p>ARRAY SHAPING</p>	4	3	CO 5	3	2.2.1
15	<p>Illustrate the worker and gang loop.</p>	4	3	CO 5	3	2.2.1
<p align="center">Part – C (2*12=24 marks) Answer any Two Questions</p>						
16	<p>(i). Examine how the OpenACC reduction clause is used to solve the problem of parallel loop. (6) (ii). Demonstrate how update directive can be used for data synchronization between host and device. (6) update: Explicitly transfers data between the host and the device</p> <p>Useful when you want to synchronize data in the middle of a data region</p> <p>Clauses:</p>	12	3	CO 4 & CO 5	2 & 3	2.2.1

	<p>self: makes host data agree with device data device: makes device data agree with host data</p> <pre> int* allocate_array(int N){ int* A=(int*) malloc(N*sizeof(int)); #pragma acc enter data create(A[0:N]) return A; } void deallocate_array(int* A){ #pragma acc exit data delete(A) free(A); } void initialize_array(int* A, int N){ for(int i = 0; i < N; i++){ A[i] = i; } #pragma acc update device(A[0:N]) } </pre> <ul style="list-style-type: none"> • Inside the initialize function we alter the host copy of 'A' • This means that after calling initialize the host and device copy of 'A' are out-of-sync • We use the update directive with the device clause to update the device copy of 'A' • Without the update directive later compute regions will use incorrect data. 					
17	<p>State and code the following OpenACC constructs: (i). Data Regions (ii). Data clauses Enter data region Exit data region</p> <p>Copyin Copyout Create delete</p>	12	3	CO 4	2 & 3	2.2.1
18	<p>Sketch and describe the OpenACC execution model.(6)</p> <ul style="list-style-type: none"> • The OpenACC target machine has a host and an attached accelerator device, such as a GPU. • Most accelerator devices can support multiple levels of parallelism. • Figure 15.2 illustrates a typical accelerator that supports three levels of parallelism. • At the outermost coarse-grain level, there are multiple execution units. Within each execution unit, there are multiple threads. • At the innermost level, each thread is capable of executing vector operations. • Currently, OpenACC does not assume any synchronization capability on the accelerator, except for thread forking and joining. • Once work is distributed among the execution units, they will execute in parallel from start to finish. • Similarly, once work is distributed among the threads within 	12	3	CO 5	3	2.2.1

- an execution unit, the threads execute in parallel.
- An OpenACC program starts its execution on the host single-threaded (Figure 15.3).
 - When the host thread encounters a parallel or a kernels construct, a parallel region or a kernels region that comprises all the code enclosed in the construct is created and launched on the accelerator device.
 - The parallel region or kernels region can optionally execute asynchronously with the host thread and join with the host thread at a future synchronization point.
 - The parallel region is executed entirely on the accelerator device.
 - The kernels region may contain a sequence of kernels, each of which is executed on the accelerator device.
 - The kernel execution follows a fork-join model. A group of gangs are used to execute each kernel.
 - A group of workers can be forked to execute a parallel work-sharing loop that belongs to a gang.
 - The workers are disbanded when the loop is done.
 - Typically a gang executes on one execution unit, and a worker runs on one thread within an execution unit.
 - The programmer can instruct how the work within a parallel region or a kernels region is to be distributed among the different levels of parallel-ism on the accelerator.

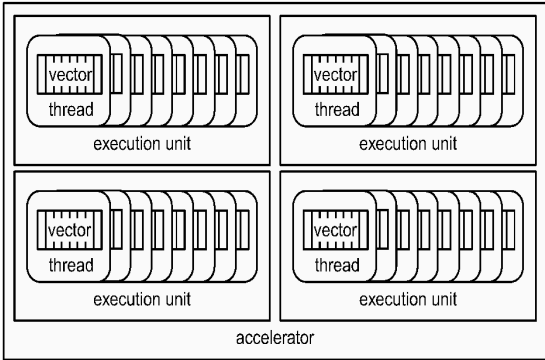


FIGURE 15.2
A typical accelerator device

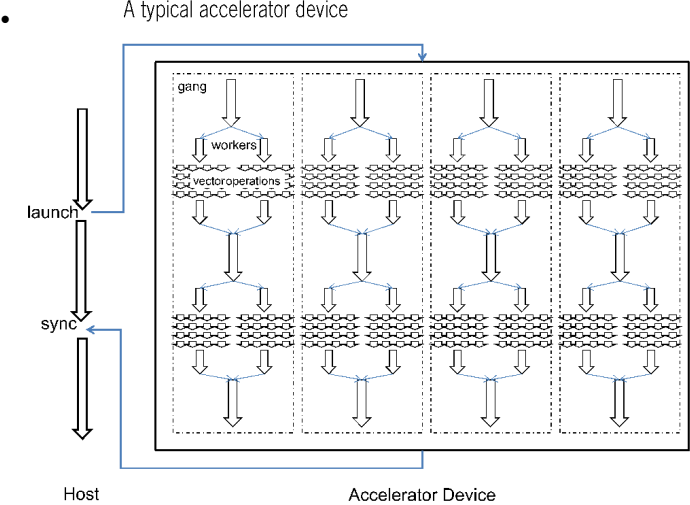
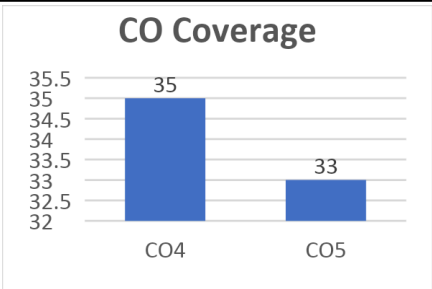
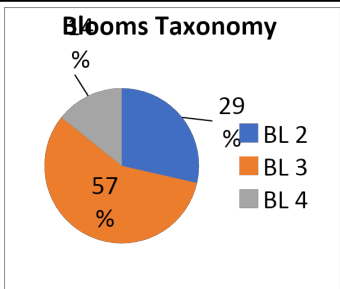


FIGURE 15.3
OpenACC execution model.

List out the clauses used for loop optimization in OpenACC and describe its functions.(6)

Register number					
	tile clause gang, worker,vector clause refer ppt for code and notes				



Approved by Audit Professor/ Course Coordinator

Register number _____