Register Number									

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110 (An Autonomous Institution, Affiliated to Anna University, Chennai)						
Computer Science and Engineering **Continuous Assessment Test -1** **Question Paper**						
Degree & Branch	B.E				**Semester**	VII
Subject Code & Name	UCS1727– GPU Computing				**Regulation:**	**2018**
Academic Year	2023-2024 ODD	**Batch**	2020-2024	**Date**	**.09.2023**	**FN**
Time: 08:15 - 09:45 AM (90 Minutes)	**Answer All Questions**				**Maximum: 50 Marks**	

(K1: Remembering, K2: Understanding, K3: Applying, K4: Analyzing, K5: Evaluating)

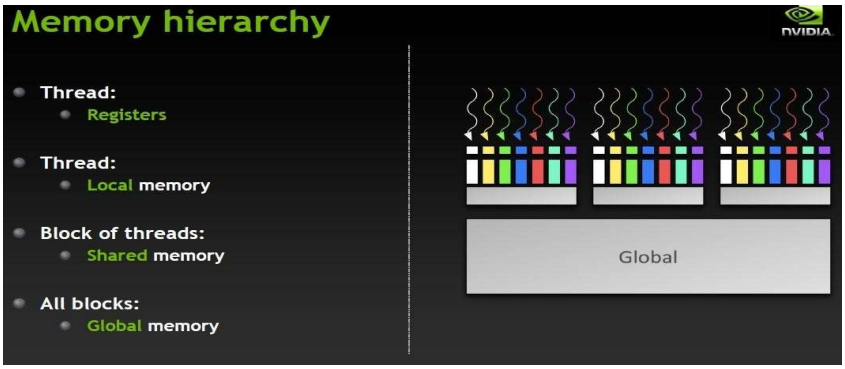
CO1:	Understand GPU architecture (K2)
CO2:	Write programs using CUDA, identify issues and debug them (K3)
CO3:	Implement efficient algorithms in GPUs for common application kernels such as matrix multiplication (K3)
CO4:	Write simple programs using OpenCL (K3)
CO5:	Write an efficient parallel program for a given problem(K3).

Part – A (6×2 = 12 Marks)

1	What are Warps? What is the size of a Warp defined by nVidia? Ans: Group of 32 threads is called a Warp. A warp of threads can be issued for SM for execution.	K1	CO1	1.3.1 2.1.3
2	What is the bandwidth of PCI-E lanes in Core-2 series GPU architecture? Ans: 5GB/s	K1	CO1	1.3.1 2.1.3
3	What are PTX instructions? Explain the fields of the PTX instruction format. Ans: opcode.type d, a, b, c; – where d is the destination operand; a, b, and c are source operands Source operands are 32-bit or 64-bit registers or a constant value. Destinations are registers,except for store instructions	K2	CO1	1.4.1
4	List the differences between CPU and GPU	K2	CO1	1.3.1 2.2.2
5	How does thread divergence impact the performance of CUDA applications,? Ans: If some threads take the if branch and other threads take the else branch, they cannot operate in lockstep Some threads must wait for the others to execute	K2	CO1	

	Renders code at that point to be serial rather than parallel			
6	Identify the CUDAAPIcall that make sure that all previous kernel executions and memory copies have been completed. Ans: cudaDeviceSynchronize()	K3	CO2	

Part – B (3×6 = 18 Marks)

7	1) We want to use each thread to calculate two(adjacent) elements of a vector addition. Assume that variable i should be the index for the first element to be processed by a thread. Identify the correct expression for mapping the thread/block indices to data index? Justify your answer. Ans: To map thread/block indices to data indices for calculating two adjacent elements of a vector addition, you can use the following expression: **int i = blockIdx.x * blockDim.x + threadIdx.x * 2;** Here's an explanation of each part of the expression: blockIdx.x represents the index of the block in the x dimension. blockDim.x represents the number of threads per block in the x dimension. threadIdx.x represents the index of the thread in the x dimension within the block. The expression threadIdx.x * 2 is used to ensure that each thread processes two adjacent elements of the vector since it multiplies the thread index by 2. This is appropriate if each thread is responsible for computing two adjacent elements in the vector addition.	K3	CO1	1.3.1 2.1.3
8	Explain the Memory Structure of GPU processor. Ans: 	K2	CO1	1.3.1 2.1.3

9	We are processing vector a that has 64 million data. We can launch one thread for each element of a vector. If we are grouping 1024 elements in to a block and assign one block to each Streaming Multiprocessor. a)Estimate the number of threads, warps, blocks need to process the data. b)Estimate the number of Streaming Multiprocessors needed in the system. c) Estimate the storage space requirements for storing: i)single precision floating point data. ii) Double precision floating point data Ans: ANS: 64 MILLION THREADS 64 ILLION /32 = 2 MILLION WARPS Single-precision floating-point number, requiring 4 bytes of data, need around 256 million bytes, or 256 MB, of data storage space for single precision and 512 MB for Double precision data	K3	CO1	1.3.1 1.4.1 **2.2.2** **3.2.2**

Part – C (2×10 = 20 Marks)

10	Explain the with a neat block diagram the architecture of Core 2 series GPU processor.	K2	CO1	1.3.1 1.4.1 2.1.3
	(OR)			
11	Explain in detail various CUDA compute levels.	K2	CO1	1.3.1 1.4.1 2.1.3

12	Explain the process of performing vector addition using CUDA programming. Provide a step-by-step explanation of how to parallelize the addition of two large vectors on a GPU, Discuss the advantages of using CUDA for vector addition compared to a sequential CPU approach.
	
	Develop a CUDA program to perform addition of two vectors.
	
	Ans:
	
	_global void VecAdd(float* A, float* B, float* C, int N) {
	
	int i = blockDim.x * blockIdx.x + threadI dx.x;
	
	if (i < N)
	
	C[i] = A[i] + B[i];
		K3	CO2	1.3.1
				2.2.2
				2.2.3
				3.3.1

```
}

int main() {

float* h_A = (float*)malloc(size);

 float* h_B = (float*)malloc(size);

float*   h_C  =  (float*)malloc(size);

float* d_A;  cudaMalloc(&d_A, size);  float* d_B;  cudaMalloc(&d_B, size);
float* d_C;  cudaMalloc(&d_C, size);

// Copy vectors from host memory to

// device memory  cudaMemcpy(d_A, h_A, size,

cudaMemcpyHostToDevice);

cudaMemcpy(d_B, h_B, size,

cudaMemcpyHostToDevice);

// Invoke kernel

int threadsPerBlock = 256;

int blocksPerGrid = N/threadsPerBlock;

VecAdd<<<blocksPerGrid, threadsPerBloc  k>>>(d_A, d_B, d_C, N);

// Copy result from device memory to

// host memory  cudaMemcpy(h_C, d_C, size,

cudaMemcpyDeviceToHost);

…

cudaFree(d_A);

  cudaFree(d_B);

cudaFree(d_C);
```

	(OR)			
13	Describe the key steps involved in implementing Radix Sort in GPU systems, including how you would handle parallelization and memory management.Develop a CUDA program to sort list of elements in an array using Radix sort. Ans: • It has a fixed number of iterations and a consistent execution flow.	K3	CO2	1.3.1 2.2.2 2.2.3 3.3.1

- It works by sorting based on the least significant bit and then working up to the most significant bit.
- With a 32-bit integer, using a single radix bit, you will have 32 iterations of the sort, no matter how large the dataset.
- example : { 122, 10, 2, 1, 2, 22, 12, 9 }
- The binary representation of each of these would be

  122 = 01111010
  10 = 00001010
  2 = 00000010
  22 = 00010010
  12 = 00001100
  9 = 00001001

- In the first pass, all elements with a 0 in the LSB would form the first list.
- Those with a 1 as the LSB would form the second list. Thus, the two lists are

  0 = { 122, 10, 2, 22, 12 } & 1 = { 9 }
- The two lists are appended in this order, becoming

  { 122, 10, 2, 22, 12, 9 }
- The process is then repeated for bit one, generating the next two lists based on the ordering of the previous cycle:

  0 = { 12, 9 } & 1 = { 122, 10, 2, 22 }
- The combined list is then

  { 12, 9, 122, 10, 2, 22 }
- Scanning the list by bit two, we generate

  0 = { 9, 122, 10, 2, 22 } & 1 = { 12 }
  = { 9, 122, 10, 2, 22, 12 }
- The program continues until it has processed all 32 bits of the list in 32 passes.
- To build the list you need **N + 2N** memory cells.
- one for the source data, one of the **0 list**, and one of the **1 list**.

```
__device__ void radix_sort (u32 * const sort_tmp, const u32 num_lists,
const u32 num_elements, const u32 tid, u32 * const sort_tmp_0, u32 * const
sort_tmp_1)
{
// Sort into num_list, lists
// Apply radix sort on 32 bits of data
        for (u32 bit=0;bit<32;bit++)
        {
        u32 base_cnt_0 = 0;
        u32 base_cnt_1 = 0;
        for (u32 i=0; i<num_elements; i+=num_lists)
        {
        const u32 elem = sort_tmp[i+tid];
        const u32 bit_mask = (1 << bit);
if ( (elem & bit_mask) > 0 )
        {
        sort_tmp_1[base_cnt_1+tid] = elem;
        base_cnt_1+=num_lists;
        }
        else
        {
        sort_tmp_0[base_cnt_0+tid] = elem;
        base_cnt_0+=num_lists;
```

```
        }
        }
// Copy data back to source - first the zero list
for (u32 i=0; i<base_cnt_0; i+=num_lists)
        {
        sort_tmp[i+tid] = sort_tmp_0[i+tid];
        }
// Copy data back to source - then the one list
        for (u32 i=0; i<base_cnt_1; i+=num_lists)
        {
        sort_tmp[base_cnt_0+i+tid] = sort_tmp_1[i+tid];
        }
        }
        __syncthreads();
}
```

Prepared By **PAC Members** **Approved By (HOD/CSE)**