Reg. No.										

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

V SEMESTER B.TECH. (COMPUTER AND COMMUNICATION ENGINEERING)

END SEMESTER EXAMINATIONS, NOV 2019

SUBJECT: PARALLEL PROGRAMMING [ICT 3153]

REVISED CREDIT SYSTEM
(20/11/2019)

Time: 3 Hours MAX. MARKS: 50

Instructions to Candidates:

- Answer **ALL** the questions.
- Missing data, if any, may be suitably assumed.

1A. Discuss the new features of Kepler micro-architecture. Also, explain the components of a Kepler SMX. **5**

1B. Suppose a multi-core system has three cores, each core has its write-back L1 cache. These caches do not have any cache-coherent support. It means that these L1 caches behave as if each of these caches was just alone in a uniprocessor. So, they behave correctly for a uniprocessor cache, but they do not try to maintain coherence. Assume that the following valid code snippet is executed by each of the three cores core 0, core 1, and core 2 one after the other.

```
Load R0, A
R0=R0+1
Store A, R0
```

Where R0 is the register and A is the address of the memory location in the memory shared by three cores. Assume that the location A contains 0 before the core 0 starts its execution. The first operand is the destination and the second operand is the source in Load and Store instructions. Write and explain the possible different values the core 2 writes to address A. **3**

1C. Explain the possible CUDA qualifier keywords used in the function declarations. **2**

2A. Explain the working of a simple work-inefficient parallel inclusive scan kernel that uses shared memory. What is the modification needed to convert this kernel into an exclusive scan kernel? Analyse its work efficiency. **5**

2B. Describe the OpenCL data-parallel execution model. **3**

2C. Which of the statement(s) in the following kernel snippet has/have coalesced access pattern? Explain.

```
__global__ void foo(float *g)
{
float a = 3.14;
int i = threadIdx.x;
        ⋮
a = g[BLOCK_WIDTH/2 + i] ;
```

```
        ⋮
g[BLOCK_WIDTH-1-i] = a;
        ⋮
}
```

2

3A. Explain the following components of Nehalem micro-architecture with their block diagrams.

i) Execution engine

ii) Memory hierarchy

5

3B. Write a complete Thrust program that sorts data on a GPU. 3

3C. Let a program have a portion of its code parallelized to run four times faster, to yield a system speedup of 3.3 times faster. What is the fraction parallelized according to Amdahl's law? 2

4A. Given an integer array A, of size N=8, contains{ 0 1 2 3 4 5 6 7}, an another array M contains {1 2 3 2 1}, and Mask_Width=5. Then the kernel is launched with 1D grid size=2 and 1D block size = 4. Analyze the kernel below and write the content of the array P after the execution of this kernel.

```
__global__ void convolution_1D_basic_kernel(int *A, int *M, int *P) {
 int i = blockIdx.x*blockDim.x + threadIdx.x;
 float Pvalue = 0;
 int N_start_point = i - (Mask_Width/2);
 for (int j = 0; j < 1; j++) {
  if (N_start_point + j >= 0 && N_start_point + j < N) {
   Pvalue += A[N_start_point + j]*M[j];
  }
 }
 P[i] = Pvalue;
}
```

5

4B. Correct the following syntactically valid kernel snippet, if there is any barrier synchronization problem.

```
__global__ void fun (.....){
__shared__ float s[1024];
int i  = threadIdx.x;
        ⋮
s[i] = (s[i-1]+s[i]+s[i+1])/3.0;
printf(" s[%d] =%f \n", i, s[i]);
__syncthreads( );
        ⋮
}
```

3

4C. What is transparent scalability? Explain the feature that enables transparent scalability for CUDA programs. 2

5A. Write a CUDA C program to square each element of an integer array A and store the result in B. Launch the kernel with a single 1D block of threads. The program should work for the number of elements of A that may be either less than, equal to, or more than the number of threads. Use error handling and boundary checking in the code. 5

5B. What is the drawback of branch divergence in a CUDA kernel? Analyse the following kernel. Also, identify the branch divergence, if any, in the following kernel.

```
__global__ void gpuCube(float *d_in, float *d_out)
{
int tid = threadIdx.x;
if(tid%2 == 0)
{
    float temp = d_in[tid];
    d_out[tid] = temp*temp*temp;
}
else
{
    float temp = d_in[tid];
    d_out[tid] = temp*temp*temp;
}
}
```

3

5C. With suitable CUDA constructs, illustrate how to use constant memory in a program. **2**