CSET 305 High Performance Computing

Time: 1 hour   Name Shaurya Enrolment No 1062   Max. Marks: 20

Instructions:

1. **Q1 to Q10 carry 0.5 mark each and Q11 to Q25 carry 1 mark each.**
2. **Submit duly signed OMR sheet at the end of exam, ensure your name and enrolment number.**
3. **Ink correction fluid is strictly prohibited. Extra OMR sheet will not be provided.**
4. **Use black/blue ball point pen to fill circles of OMR sheet. The question(s) having more than one filled circle will not be evaluated. Also, answers with correction fluid will not be evaluated.**

Q1. In Heterogeneous application serial parts run on ________ and parallel parts run on________.

[✓ A.] host, device
[✓ B.] device, host
C. device, device
D. host, host

Q2. When all threads in a grid run the same kernel, the code called:

[✓ A.] Single Program Multiple Data
B. Single Data Multiple Program
C. Single Data Multiple Data
D. Single Program Single Data

Q3. What are the various ways in CUDA to accelerate applications: 1. Libraries, 2. Compiler Directive, 3. Programming Languages?

A. 1,2
B. 2,3
[✓ C.] 1,2,3
[✓ D.] None of these

Q4. Which of the following libraries are used for parallel algorithms in CUDA?

A. CuDNN
B. TensorRT
C. Thrust
[✓ D.] cuFFT

Q5. CPU is a:

A. Latency oriented
B. Throughput oriented
C. Both latency and throughput oriented
[✓ D.] None of these

Q6. The same application runs efficiently on different type of cores refers to ______________ in GPU computing:

[✓ A.] Scalability
B. Portability
C. Concurrency
[✓ D.] Parallelism

Q7. Which of the following is not a CUDA device memory management API function?

A. cudaMalloc()
B. cudaFree()
C. cudaHostcpy()
[✓ D.] cudaMemcpy()

Q8. What is the main goal of tiling (also known as blocking) in HPC?

A. To increase the number of instructions executed per cycle
[✓ B.] To reduce memory access time by improving data locality
C. To decrease the number of arithmetic operations
D. To increase the number of threads in a program

Q9. The main advantage of Unified Memory in heterogeneous systems is:

[✓ A.] It eliminates the need for explicit data transfers between CPU and GPU
B. It increases GPU clock speed
[✓ C.] It reduces the number of CPU cores required
D. It allows simultaneous kernel execution

Q10. What is the main purpose of a barrier synchronization in parallel programming?

A. To divide threads into separate warps
[✓ B.] To make all threads in a block wait until every thread reaches a certain point
C. To terminate threads that are idle
D. To prevent memory allocation errors

Q11. Which statement about shared memory in CUDA is TRUE?

[✓ A.] Shared memory is accessible across the grid.
B. Shared memory is shared among all SMs.
C. Shared memory is accessible only by threads within the same block.
D. Shared memory is global across all kernels.

Q12. On a GPU, each SM can accommodate up to 64 warps, but due to register constraints, a kernel uses only 48 warps per SM. If each block has 12 warps, what is the maximum number of active blocks per SM?

A. 2
B. 3
[✓ C.] 4
[✓ D.] 5

Q13. A block is launched with 116 threads. Each warp can have only 32 threads. How many threads are idle in the last warp?

A. 7
[✓ B.] 12
C. 20
D. 32

Q14. If a kernel is launched with << P , Q >>, how many threads are created in total?

[✓ A.] P+Q
[✓ B.] P*Q

C. P
D. Q

Q15. A kernel function defined as __global__ myKernelFunction() is executed on _______and callable from _________.

A. Host, Host
[✓ B.] Host, Device
C. Device, Device
D. Device, Host

Q16. In GPU programming (CUDA), threads in different blocks __________.

A. Always interact
B. Never interact
C. Do not interact directly
[✓ D.] Share a common local memory

Q17. The smallest schedulable unit of threads in CUDA is a ________.

A. Block
[✓ B.] Warp
C. Thread
D. Grid

Q18. Transparent scalability in CUDA refers to the fact that each block can execute in _________ with respect to others.

[✓ A.] A fixed sequence
[✓ B.] Any order
C. Parallel only
D. Strict synchronization

Q19. In CUDA architecture, threads are assigned to Streaming Multiprocessors (SMs) in ________.

A. Thread granularity
B. Warp granularity
[✓ C.] Block granularity
D. Grid granularity

Q20. In Von Neumann architecture, the main difference between SISD and SIMD systems is in the number of __________.

A. Memory units
[✓ B.] Processing units
C. Instruction decoders
D. Cache levels

Q21. A kernel is launched in a 3D Grid with dimensions (16 x 4 x 32) blocks. Each block has (64 x 32 x 16) threads. How many total threads are launched.

A. 7286123
B. 32816
[✓ C.] 67108864
D. None of these

32^3

1024
x 32
2048
+30720
32768
~~x 204~~

2048
2656
32768
x 2048
262144
1310720
65536000
67108864

65536
2) 131072
-12
11
-10
1

Q22. Each thread calculates two (adjacent) output elements of a vector addition. Assume that variable i should be the index for the first element to be processed by a thread. What would be the expression for mapping the thread/block indices to data index of the first element?

A. i=blockIdx.x*blockDim.x + threadIdx.x +2;
[✓ B.] i=blockIdx.x*threadIdx.x*2
C. i=(blockIdx.x*blockDim.x + threadIdx.x)*2
D. i=blockIdx.x*blockDim.x*2 + threadIdx.x

Q23. In CUDA programming, what does the following expression compute inside a kernel?

int Row = blockIdx.y * blockDim.y + threadIdx.y;
int Col = blockIdx.x * blockDim.x + threadIdx.x;

[✓ A.] The total number of threads in the grid
B. The global row and column indices of a thread in a 2D grid
C. The number of blocks in each dimension
D. The block and thread IDs only within the block

Q24. A CUDA kernel is launched with the following configuration for processing an image of size 62 × 76 pixels. How many total threads are launched, and how many of them will perform valid pixel computations?

dim3 DimGrid((76-1)/16 + 1, (62-1)/16 + 1, 1);
dim3 DimBlock(16, 16, 1);

A. 4864 threads launched; all perform valid computations
B. 6144 threads launched; 4712 perform valid computations
C. 3214 threads launched; 4712 perform valid computations
[✓ D.] 4864 threads launched; 4096 perform valid computations

Q25. If we want to copy 3000 bytes of data from host array h_A (h_A is a pointer to element 0 of the source array) to device array d_A (d_A is a pointer to element 0 of the destination array), what would be an appropriate API call for this in CUDA?

A. cudaMemcpy(3000, h_A, d_A, cudaMemcpyHostToDevice);
B. cudaMemcpy(h_A, d_A, 3000, cudaMemcpyDeviceTHost);
C. cudaMemcpy(d_A, h_A, 3000, cudaMemcpyHostToDevice);
[✓ D.] cudaMemcpy(3000, d_A, h_A, cudaMemcpyHostToDevice);