MANIPAL ACADEMY of HIGHER EDUCATION (Deemed to be University under Section 3 of the UGC Act, 1956)

Question Paper - Report

08-Apr-2024 00:25:07
SRUJANA AKELLA . .
Logout

Question Paper

Back

MANIPAL ACADEMY OF HIGHER EDUCATION

B.Tech VIth Semester Mid semester Examination March 2024

PARALLEL PROGRAMMING [DSE 3254]

Marks: 30 **Duration: 120 mins.**

MCQ

Answer all the questions. Section Duration: 20 mins

Missing valies may be suitably assumed

1) Given that the maximum number of threads in a thread block is 1600, which among the following distributions of thread is not allowed: (0.5)

(4,4,256) (4, 64, 4) (8, 8, 16) (32,32,1)

2) A CUDA device has 40 SMs and each SM can accommodate 8 thread blocks simultaneously. Each thread block has 4 warps of size 24. The device can have up to ________________ threads simultaneously residing on the device: (0.5)

30,720 3,840 7,680 9,60.

3) In the following function call, a message is sent to which process? (0.5)

MPI_Send(message, 2, MPI_CHAR, 4, 0, MPI_COMM_WORLD)

Process 4 Process 2 Process 1 Process 0

4) In MPI, what is the difference between blocking and non-blocking communication (0.5)

Blocking communication waits for the message to be received/sent before proceeding, while non-blocking communication	Blocking communication involves copying data, while non-blocking uses references.	Blocking communication is for inter-process communication, and non-blocking is for intra-process communication.	There is no significant difference in practice.

continues execution.

5)	What is a common approach to avoid deadlocks in MPI programs?					
	Use non-blocking communication whenever possible.	Employ synchronization with barriers and locks.	Increase the buffer size for message passing.	Reduce the number of processes involved in communication.	(0.5)
6)	What is a potential challenge in OpenMP programming with shared memory?					
	Limited communication overhead between threads.	Race conditions when accessing shared data	Difficulty in managing threads.	Lack of portability across different compilers.	(0.5)
7)	What is the advantage of using atomic operations in OpenMP?					
	They enable efficient communication between threads.	They guarantee thread-safe updates to shared variables.	They define a barrier for thread synchronization.	They improve performance for frequent read/write operations.	(0.5)
8)	What is the difference between ***#pragma omp for*** and ***#pragma omp parallel for*** directives?					
	#pragma omp for implies a single thread iterating over the loop, while #pragma omp parallel for creates multiple threads iterating over the loop.	#pragma omp for is for data-parallel loops, while #pragma omp parallel for is for task-parallel loops.	#pragma omp for requires explicit synchronization, while #pragma omp parallel for handles it internally.	There is no significant difference; they achieve the same functionality.	(0.5)
9)	The rank of an MPI process in the MPI_COMM_WORLD communicator is a					
	Floating point number starting from 1.0	Floating point number starting from 0.0	Integer number starting from 0	Integer number starting from 1	(0.5)
10)	In the vector addition kernel, the global threadID can be obtained by using the following formula				(0.5)

threadIdx.x + blockIdx.x*blockDim.x	threadIdx.y + blockIdx.y* blockDim.y	blockDim.x + blockIdx.x* threadIdx.x	blockIdx.x + blockIdx.x*blockDim.x

DESCRIPTIVE

Answer all the questions.

11) You're working on a real-time gray-scale image processing application. The processing involves a pre-processing step where you are supposed to determine the Cube of every pixel and then divide it by the total number of pixels in the given input image. Design a CUDA program to do the above task parallelly. Assume that the image dimension is 780x650. Also, consider that the streaming multi-processor can take a maximum of up to 2048 threads and it allows up to 1024 threads in each block. Assume that the Image is already read into variable **I** and the size of the image is already estimated and stored in variable **SIZE_I** . (4)

12) Design an MPI program to read an integer value in the root process. Root process sends this value to process 1, process 1 sends this value to process 2, and so on. The last process sends this value back to the root process. When sending the value each process will first decrement the received value by one. Design the program using point-to-point communication routines. (4)

13) Parallelize the following code using OpenMP pragmas. Be sure to explicitly specify the iterations to be divided into chunks of size **C_SIZE** which is based on **N**. For each rewrite the code. If necessary you can assume that the variable **P** represents the number of processors to be used. Assume that **N** is large (in the tens of thousands or more). You must explicitly list all variables within the range of a parallel pragma that are private using the private() directive. (3)

```
i) for (i=0;i< N.i++){
for (j=0;j< N;j++){

A[i,j] = max(A[i,j],B[i,j]);

}

}

ii) C[0] = 1;

for (i=1;i< N;i++){

C[i] = C[i-1];

for (j=0;j< N;j++){

C[i] *= A[i,j] + B[i,j];

}

}.
```

14) In the following code, one process sets array **A** and then uses it to update **B**; the other process sets array **B** and then uses it to update **A**. Argue that this code can lead to deadlock. How could you fix this? (3)

```
#pragma omp parallel shared(a, b, nthreads, locka, lockb)

#pragma omp sections nowait

{

#pragma omp section

{

omp_set_lock(&locka);

for (i=0; i< N; i++)

a[i] = .... /* Some operation*/

omp_set_lock(&lockb);

for (i=0; i< N; i++)

b[i] = .... /* Some operation*/ a[i] .... /* Some operation*/

omp_unset_lock(&lockb);

omp_unset_lock(&locka);

}

#pragma omp section

{

omp_set_lock(&lockb);

for (i=0; i< N; i++)

b[i] = ... /* Some operation*/

omp_set_lock(&locka);

for (i=0; i< N; i++)

a[i] = .... /* Some operation*/ b[i] ..... /* Some operation*/

omp_unset_lock(&locka);

omp_unset_lock(&lockb);

}

}
```

}.

15) You're developing a parallel image processing application that filters large images. The application follows these steps:

a. Image Loading: Each thread loads a specific tile (sub-section) of the image into its local memory.

b. Filtering: Each thread applies a filter (e.g., blur, sharpen) to its assigned tile independently.

c. Result Accumulation: The filtered tiles need to be combined to form the final filtered image. However, this requires ensuring all filtering is complete before any thread starts modifying the final image. (3)

At what point in the code would you strategically place the OpenMP Barrier directive to achieve proper synchronization and avoid race conditions? Explain your reasoning. Briefly discuss the potential performance implications of using the Barrier directive in this scenario. Are there any alternative synchronization mechanisms you might consider (if applicable)?.

16) You're working on a physics simulation that involves calculating the force exerted by a large number of particles (N). Each particle has acceleration and mass. Assume that the acceleration and mass of all the particles are stored in two vectors ACC and MASS. The force exerted by each particle is calculated using the following formula: F=MA where F is force, M is mass and A is acceleration. Design a CUDA program that efficiently calculates the forces exerted on all individual particles in parallel and print the same from host code. Assume that, N=770 and a block can have 256 threads. (3)

17) Compare and contrast MPI_BCAST and MPI_SCATTER operations with suitable examples. (3)

18) Discuss the cache coherence problem and its significance in parallel programming. (2)

^ Top