Name : Sumanth Tanga

Student Identity Number : 2016

Programme : B.Tech CICT

Supervisor's Signature : .....

Start here

1. (i) If we want to allocate an array of **n integer** elements in CUDA device global memory, what would be an appropriate expression for the **second argument** of the cudaMalloc() call?

   Answer: sizeof(int) * n

   (ii) If we want to allocate an array of **n floating-point elements** and have a floating-point pointer variable d_A to point to the allocated memory, what would be an appropriate expression for **the first argument** of the cudaMalloc() call? (1)

   Answer: (void**) &d_A

2. (i) 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?

   Answer: cudaMemcpy(d_A, h_A, 3000 * sizeof(h_A[0]), cudaMemcpyHostToDevice)

   (ii) How would one declare a variable **err** that can appropriately receive returned value of a CUDA API call? (1)

   Answer: cudaError_t err;

3. (i) For a vector addition, assume that the vector length is 8000, each thread calculates one output element, and the thread block size is 1024 threads. The programmer configures the kernel launch to have a minimal number of thread blocks to cover all output elements. How many threads will be in the grid?

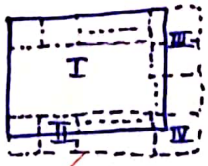
   Answer: 1024 x 8 = 8192 threads

   (ii) For a tiled matrix-matrix multiplication kernel, if we use a 32 X 32 tile, what is the reduction of memory bandwidth usage for input matrices A and B? (1)

   Answer: $\frac{1}{32}$ times

4. Assume a tiled matrix multiplication that handles boundary conditions. Assume that we use 32 X 32 tiles to process square matrices of 1,000 X 1,000. Within EACH thread block, what is the maximal number of warps that will have control divergence due to handling boundary conditions for loading A tiles throughout the kernel execution? Block Organisation = (ceil(1000/32), ceil(1000/32)) = (32, 32)

   Answer:



(1)

In blocks in section I, no warp will have control divergence

In section II, No warp will have control divergence

In section III, All 32 warps of the block will have control divergence

In section IV, 32 - 24 = 8 warps of the block will have control divergence.

Name : Sumanth Tangi
Student Identity Number : 2016.01
Programme : B.Tech (ICT)
Supervisor's Signature :
Start here

5. We are to process a 600 X 800 (800 pixels in the x or horizontal direction, 600 pixels in the y or vertical direction) picture with the PictureKernel(). That is m's value is 600 and n's value is 800.
Code is given below

```
__global__ void PictureKernel(float* d_Pin, float* d_Pout, int n, int m) {
  // Calculate the row # of the d_Pin and d_Pout element to process
  int Row = blockIdx.y*blockDim.y + threadIdx.y;
  // Calculate the column # of the d_Pin and d_Pout element to process
  int Col = blockIdx.x*blockDim.x + threadIdx.x;
  // each thread computes one element of d_Pout if in range
  if ((Row < m) && (Col < n)) {
    d_Pout[Row*n+Col] = 2*d_Pin[Row*n+Col];
  }
}
```

Assume that we decided to use a grid of 16X16 blocks. That is, each block is organized as a 2D 16X16 array of threads.
How many warps will be generated during the execution of the kernel **and** How many warps will have control divergence?

(1)

Answer: Total no. of warps = 8 warps/block × ceil(600/16) × ceil(800/16)
= 8 × 38 × 50 = 15200 warps

No warp will have control divergence

6. Assume the following simple matrix multiplication kernel

```
__global__ void MatrixMulKernel(float* M, float* N, float* P, int Width)
{
int Row = blockIdx.y*blockDim.y+threadIdx.y;
int Col = blockIdx.x*blockDim.x+threadIdx.x;
if ((Row < Width) && (Col < Width)) {
    float Pvalue = 0;
    for (int k = 0; k < Width; ++k) {Pvalue += M[Row*Width+k] * N[k*Width+Col];}
    P[Row*Width+Col] = Pvalue;
}
}
```

Which of the following is/are **true?**

(A) M[Row*Width+k] and N[k*Width+Col] are coalesced but P[Row*Width+Col] is not
(B) M[Row*Width+k], N[k*Width+Col] and P[Row*Width+Col] are all coalesced
(C) M[Row*Width+k] is not coalesced but N[k*Width+Col] and P[Row*Width+Col] both are
(D) M[Row*Width+k] is coalesced but N[k*Width+Col] andt P[Row*Width+Col] are not

(1)

Answer: M[Row×Width+k] is not coalesced but N[k*Width+Col] and P[Row×Width+Col] are coalesced.

Since consecutive threads, have threadIdx.x consecutively => Col is consecutive => N & P are coalesced during access

Ans: While fetching the input tile, atleast 1 warp will have code divergence, since 112 elements must be fetched outside the 12×12 tile ⇒ 112/32 → 4 warps are required with 6 threads in the last warp being inactive.

7. In a tiled 2D convolution with 12x12 output tiles and 5x5 mask, how many warps in each thread block will have control divergence?

Answer: ~~All 5 warps of each thread block will have control divergence~~ Assuming that boundary conditions are handled, only the last warp will have control divergence

8. For a tiled 2D convolution, if each output tile is a square with 12 elements on each side and the mask is a square with 5 elements on each side, how many elements are in each input tile?

Answer: n = 2 where 2n+1 is filter's side
⇒ No of elements in I/P tile = (12 + 2×2)·(12+2×2) = 256 elements

9. For the following basic reduction kernel code fragment, if the block size is 1024 and warp size is 32, how many warps in a block will have divergence during the iteration where stride is equal to 16?

```
unsigned int t = threadIdx.x;
Unsigned unsigned int start = 2*blockIdx.x*blockDim.x;
partialSum[t] = input[start + t];
partialSum[blockDim.x+t] = input[start+ blockDim.x+t];
for (unsigned int stride = 1; stride <= blockDim.x; stride *= 2)
{
__syncthreads();
if (t % stride == 0) {partialSum[2*t]+= partialSum[2*t+stride];}
}
```

Answer: In each warp, there will be 2 active & 30 inactive threads
So all 32 warps will have ~~code~~ divergence

10. For the following improved reduction kernel, if the block size is 1024 and warp size is 32, how many warps will have divergence during the iteration where stride is equal to 16?

```
unsigned int t = threadIdx.x;
Unsigned unsigned int start = 2*blockIdx.x*blockDim.x;
partialSum[t] = input[start + t];
partialSum[blockDim.x+t] = input[start+ blockDim.x+t];
for (unsigned int stride = blockDim.x; stride > 0; stride /= 2)
{
__syncthreads();
if (t < stride) {partialSum[t] += partialSum[t+stride];}
}
```

Answer: In the 1st warp, first 16 threads are active and rest are i
In all other 31 warps, all threads are inactive

So only 1 warp has ~~code~~ divergence

11. For the work efficient exclusive scan kernel based on reduction trees and inverse reduction trees, assume that we have **1024** elements, give the **closest approximation** on the total number of add operations performed in both the reduction tree phase and the reverse reduction tree phase?

Answer: 3064 additions. X

12. For the work inefficient scan kernel based on reduction trees, assume that we have 1024 elements in each section and warp size is 32, how many warps in each block will have control divergence during the iteration where stride is 16? For your convenience, the relevant code fragment from the kernel is given below:

```
for (unsigned int stride = 1; stride <= threadIdx.x; stride *= 2) {
__syncthreads();
float in1 = XY[threadIdx.x-stride];
__syncthreads();
XY[threadIdx.x] += in1;
}
```

1

Answer: Only the 1st warp will have code divergence

13. Which of the following is a correct CUDA API call that allocates 1024 bytes of pinned memory for h_A?

(A) cudaHostAlloc((void **) h_A, 1024, cudaHostAllocDefault);
(B) cudaPinnedAlloc((void **) h_A, 1024, cudaPinnedAllocDefault);
(C) cudaHostAlloc((void **) &h_A, 1024, cudaHostAllocDefault);
(D) cudaPinnedAlloc((void **) &h_A, 1024, cudaPinnedAllocDefault);

1 ✓

Answer: cudaHostAlloc((void **) &h_A, 1024, cudaHostAllocDefault);

14. Which CUDA API call can be used to perform an asynchronous data transfer?

1 ✓

Answer: cudaMemcpyAsync (dev_A, host_A, size, cudaMemcpyHostToDevice, can be any flag

15. What is the CUDA API call that makes sure that all previous kernel executions and memory copies in a device have been completed?

(A) __syncthreads()
(B) cudaDeviceSynchronize()
(C) cudaStreamSynchronize()
(D) __barrier()

Answer: cudaStreamSynchronize() X

Student Identity Number : 2016.01.105

Programme : B. Tech (ICT)

nature :

16. Given the fact: The memory is distributed among banks in such a way that each 32-bit word in a sequence is sequentially assigned to one of 32 banks. Suppose each thread loads 2 elements into shared memory using the following code:

```
int tid = threadIdx.x;
shared[2*tid] = global[2*tid];
shared[2*tid + 1 ] = global[2*tid + 1];
```

Does it have bank conflict?

Answer: Yes. Example: tid 0 & 16 access Bank 0.

17. (a) What is the **CGMA** ratio of the following operation in a kernel (k, row, col are integers and Pvalue, Md, Nd are floating point variables):

```
Pvalue += Md[row][k] * Nd[k][col];
```

Answer: CGMA = $\frac{2}{2} = 1$

18. Assume that each atomic operation in a DRAM system has a total latency of 100ns. Assume that a kernel performs 5 floating-point operations per atomic operation. What is the maximal floating-point throughput of the kernel execution as limited by the throughput of the atomic operations?

Answer: Throughput = $\frac{5}{100 \times 10^{-9}}$ Flops/sec = 0.05 GFlops/sec.

19. **In the previous Question,** assume that we privatize the global memory variable into shared memory variables in the kernel and the shared memory access latency is 1ns. All original global memory atomic operations are converted into shared memory atomic operation. For simplicity, assume that the additional global memory atomic operations for accumulating privatized variable into the global variable adds 10% to the total execution time. Assume that a kernel performs 5 floating-point operations per atomic operation. What is the maximal floating-point throughput of the kernel execution as limited by the throughput of the atomic operations?

Answer: Assuming that privatized variable is accumulated after every shared memory access atomic operation.

$\Rightarrow 1 + \frac{10}{100} \times 1 = 1.1$ ns for each of atomic operation.

Throughput = $\frac{5}{1.1\text{ns}} = \frac{5}{1.1}$ GFlops/sec.

$= 4.55$ GFlops/sec.

20. What is the theoretical performance of the machine (or computer) you use in lab-207, make sensible assumptions to answer the question. (**answer with proper units**). Show your work in the space provided below. Write the answer separately for CPU and GPU.

Answer: Assuming that the CPU is a Quad core machine with bandwidth of 3 GHz and that each core can compute 4 Flops/cycle.

CPU Performance = $4 \text{ Flops/cycle} \times 3 \times 10^9 \times \text{cycles/sec} \times 4 \text{ cores}$

$= 48$ GFlops/sec.

The GPU ~~in the cluster has a~~ is assumed to have a bandwidth of 1 GHz and 600 cores

GPU Performance = $4 \text{ Flop/cycle} \times 1 \times 10^9 \text{ cycles/sec} \times 600 \text{ cores}$

$= 2400$ GFlops/sec.

21. Consider the following code, which we want to parallelize.

```
do i = 1, n
a [2*i] = b[i] + c[i]
d[i] = a [2*i+1]
enddo
```

Is parallelization possible, explain briefly your "Yes" or "No". (No marks for writing only Yes/No.)

Answer: Yes, But only after Loop distribution.

1.5 In the current loop we have an anti-dependence => d[i] uses a value in a[i+1] before a[i+1] stores another value in the next iteration.

It can be distributed into:

```
Loop-1 [ do i = 1, n
           d[i] = a[2*i+1]
         end do

Loop-2 [ do i = 1, n
           a[2*i] = b[i]+c[i]
         end do.
```

Here Loop-1 & Loop-2 can be separately parallelised, however we must have a synchronisation between Loop-1 & Loop-2.

22. In a problem of given size, 6% of the operations of a parallel program are inside a I/O functions, that are executed on a single processor, What is the minimum no. of processors required so that the parallel program gives a speedup of 10?

Answer: Assuming that the rest of the 94% of the operations can be parallelised,

From ~~Alan~~ Amhdal's Law, theoritical max. speedup = $\frac{1}{\frac{P}{N} + S}$

P -> Fraction of parallelisable code

S -> Fraction of serial code

N -> No. of processors.

1.5 Since I/O operations are serial,

For a speedup of 10, $10 = \frac{1}{\frac{0.94}{N} + 0.06}$ $\Rightarrow 9.4 + 0.6N = N$

$\Rightarrow 0.4N = 9.4 \Rightarrow N = \frac{94}{4} = 23.5 \approx 24$

So we need a 24 processors for a theoritical maximum speedup of 10

most number of threads in each SM?

a. 64 threads per block
b. 128 threads per block
c. 512 threads per block
d. 1,024 threads per block

Answer: 512 threads per block & 3 blocks per SM

⇒ 512 × 3 = 1536 threads per SM.

1.5

24. Consider a general case where a pipeline of depth "m" is executing "N" independent subsequent operations.

Suppose you want to get 0.5 instruction per cycle , how large should be "N" in terms of "m".

Answer: Total No. of cycles = $\underbrace{(m-1)}_{\text{Windup time}} + \underbrace{N}_{\text{o/p phase}}$.

No. of instructions per cycle = $\frac{N}{m-1+N} = \frac{1}{2}$

$\Rightarrow 2N = N + m - 1 \Rightarrow N = m - 1$

So for 0.5 instructions per cycle, $N \approx m$

1.5

25. Can we parallelize the following loop? Support your "Yes" or "No" and show your work. (no marks for writing only "yes" or "no", or for partially correct answer)

```
a[0] = 0;
for( i = 1; i < n; i++)
a[i] = a[i-1] + i;
```

Answer: No. We have a loop carried dependency with a dependence distance of 1.

We cannot parallelise it since computation of a[i] needs to wait for computation of a[i-1] which needs to wait for computation of a[i-2] and so on.

26. Consider the following sparse matrix; represent it in CSR (Compressed Sparse Row) storage format.

3	0	1	0
0	0	0	0
0	2	4	1
1	0	0	1

Answer:

Data = { 3, 1, 2, 4, 1, 1, 1 } ✓

Column = { (0,2), (1,2,3), (0,3) } ✓

Row = { 0, 2, 2, 5, 7 } ✓

1.5

27. Complete the following table from row (b) to (e). (*Some of the cells in the table for example row (a) are already filled for representation*).

	Variable Decleration	Memory	Scope
(a)	int var;	**register**	**thread**
(b)	__shared__ int shared_val;	Shared	Block. ✓
(c)	int array_var [100];	Local	Thread ✓
(d)	__device__ int global_var;	Global	Kernel ✗ ✓
(e)	__constant__ int constant_var;	Constant	kernel. ✗ ✓

1
