King Saud University

Course Code:	CSC 453
Course Title:	Parallel Computing
Semester:	Spring 2022
Exercises Cover Sheet:	**Midterm 1 Exam**

Student Name:	
Student ID:	
Student Section No.	10 - 11

16/25

Question 1

1. Give the definition of the following terms:

   - Parallel Computing:

     Form of compution which calculations are carred out simutivesley.

     1

   - Parallel Programming:

     decomposing Programming Problems into tasks and deploy them on Processors and run them simuteously.

     1

2. Use the following table to make a comparison between parallel and Distributed computing:

Criteria	Distributed Computing	Parallel Computing
Objectives	it aims to ~~make~~ increase Servieces Availeble and relaiable.	it aims to increase Performance
Work load	it has heavey weighted load.	It has ~~Low~~ over head.
Interaction	the interaction between Processors is not frequent. · ~~Closest griead~~	it has Frequent interection with Processors. · ~~Fine grid~~.

Question 2

1. Use an example to explain the differences between the SIMD and MISD computers. ①

SIMD: Processors has same instruction stream, but have thir own Data.

A + B
C + D

2

MISD:
Processor have same date stream, but have thir own instruction stream.

Instructions

A+B
A * B

2. Explain the main differences between the **Blocking non-buffered** and the **Non-Blocking non-buffered** send/receive operations of the message passing paradigm.

1- Blocking non-buffer: Sender send operation send and it will be blocked until recever send ~~match operation Signal~~.

2- Non-Blocking non-Buffered: Sender send ~~to~~ send operation and it will remain continous proccessing until recever send interpution signal.

1

3. Describe the ***Task Farming*** and the ***Divide-and-Conquer*** programming models and explain the main differences between them.

1- Divide-and-Conquer: decompose problem into tasks ~~and~~ have the same nature, solve them recersivly

1

2-Task Framing: it's Master-Slave model, decompose problem into tasks ~~that~~ do not have the same nature

Question 3

1. Explain why the Global memory is not cached in CUDA, while the Constant memory is cached.

Because Global memory is Read/write while constact memory is Read only.

2

2. Explain why the access to the Shared memory of CUDA is faster than the access to the Local memory

Shared memory is on-chip ~~[illegible]~~ while Local memory is on DRAM

2

Question 4

Let's consider 2 arrays of integers A and B of size *N.* Let's consider that we would like to write a C program that runs in parallel and that computes the sum of 2 arrays as following:

C[i] = A[i] + B[i]

Let's consider the following kernel:

```
__global__ void add(int *a, int *b, int *c, int N) {
        int cell_id = ......................;
        if (cell_id < N)
                c[cell_id] = a[cell_id] + b[cell_id];
}
```

1. We would like to run this kernel on grid composed of **1 block** where threads are organized as a 2-D matrix. Every thread evaluates a single cell as shown in the following figure:

Block (0, 0)

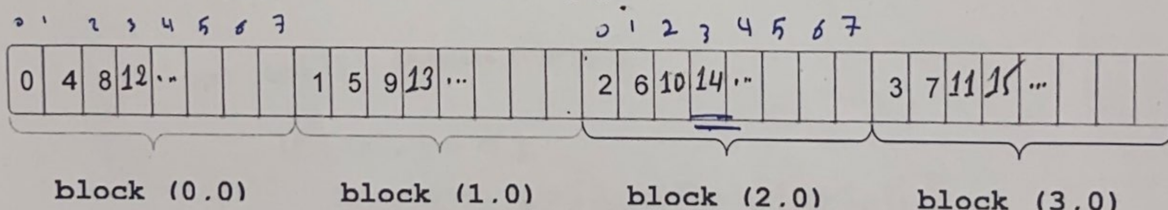
Cell 0	Cell 3	Cell 6	Cell 9	Cell 12
Cell 1	Cell 4	Cell 7	Cell 10	Cell 13
Cell 2	Cell 5	Cell 8	Cell 11	Cell 14

- Give the formula that allows every thread to compute the cell_id of the cell he is going to process.

threadIdx.y + (BlockDim.y * threadIdx.x)   2

M

2. We would like to run this kernel on grid composed of **N blocks** of **M threads** each. Every thread evaluates a single cell as shown in the following figure:



- Give the formula that allows every thread to compute the cell_id of the cell he is going to process.

Blockidx.x + (threadidx.x * N).

N = 4
M = 8

gridDim.x

Ex: 2 + (3 * 4) = 14

1.5

Question 5

Let's consider 2 arrays of integers A and B of size *N*. Let's consider that we would like to write a kernel in C that computes the sum of 2 arrays:

C[i] = A[i] + B[i]

We would like to run this kernel on a grid composed of **1 block** where every thread evaluates ***W*** cells as shown in the following figure (where *W* = 4 as a sample):

Block (0, 0)

Cells 0-3	Cells 4-7	Cells 8-11	Cells 12-15	Cells 16-19
Cells 20-23	Cells 24-27	Cells 28-31	Cells 32-35	Cells 36-39
Cells 40-43	Cells 43-47	Cells 48-51	Cells 52-55	Cells 56-60

- Write the kernel

```
__global__ void add(int *a, int *b, int *c, int size, int N, int W) {

int cell_id = ~~threadidx.x * W~~;

    if (cell_id < N)
        c[cell_id] = a[cell_id] + b[cell_id];

}
```

0

King Saud University

Course Code:	CSC 453
Course Title:	Parallel Computing
Semester:	Spring 2022
Exercises Cover Sheet:	**Midterm 2 Exam**

Student Na	
Student ID:	
Student Section No.	10-11

13.75 / 30

Question 1

Let's consider 2 arrays of integers A and B of size *N*. Let's consider that we would like to write a C program that runs in parallel and that computes the product of 2 arrays as following:

C[i] = A[i] * B[i]

Let's consider the following kernel:

```
__global__ void product(int *a, int *b, int *c, int N) {
        int cell_id = ......................;
        if (cell_id < N)
                c[cell_id] = a[cell_id] * b[cell_id];
}
```

1. We would like to run this kernel on grid configured as M * N matrix of thread blocks. Every thread handles only one cell. Give the statement that calculates the ***cell_id*** for each thread as shown in the following figure:

gridDim.x=2
—.. .y=2

grid(0,0)

Block (0, 0)

Cell 0	Cell 1	Cell 2	Cell 3	Cell 4
Cell 5	Cell 6	Cell 7	Cell 8	Cell 9
Cell 10	Cell 11	Cell 12	Cell 13	Cell 14

Block (1, 0)

Cell 15	Cell 16	Cell 17	Cell 18	Cell 19
Cell 20	Cell 21	Cell 22	Cell 23	Cell 24
Cell 25	Cell 26	Cell 27	Cell 28	Cell 29

Block (0, 1)

Cell 30	Cell 31	Cell 32	Cell 33	Cell 34
Cell 35	Cell 36	Cell 37	Cell 38	Cell 39
Cell 40	Cell 41	Cell 42	Cell 43	Cell 44

Block (1, 1)

Cell 45	Cell 46	Cell 47	Cell 48	Cell 49
Cell 50	Cell 51	Cell 52	Cell 53	Cell 54
Cell 55	Cell 56	Cell 57	Cell 58	Cell 59

- Give the formula that allows every thread to compute the cell_id of the cell he is going to process.

threadIdx.x + (blockDim.x * threadIdx.y)
+ (blockDim.x * blockDim.y * blockIdx.x)
+ (gridDim.x * blockDim.x * blockDim.y * blockIdx.y)

2

2. Let's consider that there are many devices available. We would like to run the kernel described above on the device having the maximum number of multi-processors.

    - Give the fragment of code that allows to select such device and run the kernel on that device.

↓

```
int fun{

int count;

cudagetProperties prop;
cudagetDeviceCount(&count); —2

for(int i=0, i<count, i++){
~~cudagetDeviceCount(& count);~~

~~cuda get max multiprocessor(prop, i)~~
}
```

Question 2

size = 8

Let's consider that we want to sort the following array:

5	3	18	12	6	10	14	4

− + − +

1. Let's consider that we want to apply the odd-even parallel sort algorithm to sort the given array in an *ascending* order.

odd-even

   a. How many iterations are required to sort an array of size N.

~~$2^3 = 8$, so we need 3~~ 0

   b. How many steps are required in every iteration.

~~Iteration 1 → 1 step~~
~~Iteration 2 → 2 steps~~
~~Iteration 3 → 3 steps~~ 0

   c. Describe the role of every Step.

~~Step 1: Sort the first ~~two~~ threads... odd values~~
~~Step 2: sort them as if succeed even conditions~~ 0
~~Step 3: Sort them ~~after~~ after the merge.~~

   d. For every Step S_i of the iteration No 1:

      i. Show all changes made on the array during the step S_i.

      ii. Give the formula that allows to identify threads involved in the step S_i.

~~2~~ 3	~~5~~ 5	~~12~~ 18	~~18~~ 12	~~6~~	~~10~~	[illegible] 4	14

0

~~ThreadIdx % n / L shift where n = step 2 / 2stage, shift = $\frac{n}{2}$~~

~~for 0 to n~~
if(thread %2 != 0) odd
swap(i, i+1)

for 0 to n
if(tread %2 == 0) even
swap(i, i+1)

1

4

2. Let's consider that we want to apply the bitonic merge-sort algorithm to sort the given array in a ***descending*** order.

e. Show all changes made on the array during ***stage 1 of step 1*** of the algorithm.

5	3	12	18	10	6	4	14

1.5

f. Which threads will be involved in ***stage 1 of step 1*** in case the algorithm is performed in parallel. Don't forget to specify, for every thread, the index of the cells it will process as well as the operation it will apply (+BM or –BM)

~~T0~~ $T_0[0,1]$, $T_2[2,3]$, $T_4[4,5]$, $T_6[6,7]$ 2
(~~+BM~~) (~~+BM~~) (~~±BM~~) (+BM)

g. Show all changes made on the array during ***stage 1 of step 2*** of the algorithm.

~~18~~	~~12~~	5	3	~~6~~	~~4~~	~~14~~	~~10~~

0.5

h. Which threads will be involved in this ***stage 1 of step 2*** in case the algorithm is performed in parallel. Don't forget to specify, for every thread, the index of the cells it will process as well as the operation it will apply (+BM or –BM).

$T_0[0,2]$, $T_1[1,3]$, $T_4[4,6]$, $T_5[5,7]$
2 (±BM) (±BM) (+BM) (+BM)

Question 3

Let's consider the following parallel CUDA code:

```
__global__ void Kernel_A(int *data) {
    data[threadIdx.x] = threadIdx.x;
    __syncthreads();
    if (threadIdx.x == 0) {
        Kernel_C<<< 1, 256 >>>(data);
        Kernel_D<<< 1, 256 >>>(data);
        cudaDeviceSynchronize();
    }
    __syncthreads();
}
__global__ void Kernel_C(int *data) {
    data[threadIdx.x] = threadIdx.x;
    __syncthreads();
    if (threadIdx.x == 0) {
        Kernel_E<<< 1, 256 >>>(data);
        Kernel_F<<< 1, 256 >>>(data);
        cudaDeviceSynchronize();
    }
    __syncthreads();
}
void host_launch(int *data) {
        kernel_A<<< 1, 256 >>>(data);
        kernel_B<<< 1, 256 >>>(data);
        cudaDeviceSynchronize();
}
```

1. Give and explain the order of execution of the given parallel nested kernels.

A , C , E , F , D , B   1,25

2. Explain the role of the syncthreads() statements.

It force threads to wait at certen Point waiting for synchrnizetion   1

3. Explain the role of the cudaDeviceSynchronize() statements.

It force the caller (CPU or device) to wait for threeds to finish.   0.5