King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 1 – Fall 2021

Question 1

1. Give the definition of Parallel computing and Parallel Programming

Parallel computing: is a form of computation in which many calculations are carried out simultaneously.
Parallel Programming: Decomposing a programming problem into tasks and Deploy the tasks on multiple processors and run them simultaneously

2. Enumerate and give a brief description of the main opportunities of parallelism.

3. Enumerate and give a brief description of the main aspects of parallel computing.

4. What are the main differences between Distributed and Parallel Computing.

Q2:
1- Instruction Level Parallelism:
Hidden Parallelism in computer programs

2- Single computer level:
Multi core computers: Chip multi processors
Dual core, Quad core, GP GPU
Multi processor computers: Symmetric multi processors
Supercomputers

3- Multiple computers level:
Clusters(Known Configs at compile time), Servers, Grid computing(Daynamic can change at run time)

Q3
1- Parallel Computers Architecture

2- Algorithms and applications
Reasoning about performance
Designing parallel algorithms.

3- Parallel Programming
Paradigms
Programming Models
Programming languages
Frameworks
Dedicated environments

Q4
1- in distributed computing the procssors are geograhpic distant but in parallel computing they are in the same machine
2- in distributed computing the aim is to make services avilable and realiable but in parallel computing the aim is to increase the preformance
3- in parallel computing interaction is Fine grained with low overhead but in distributed computing interaction is Coarse grained and heavier weight

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 2 – Spring 2021

Question

1. Give the flynn's classification of computers.
   SISD , SIMD , MISD , MIMD
2. Use an example to explain the differences between the ***SIMD*** and ***MIMD*** computers.
3. Explain the main differences between the **Blocking non-buffered** and the **Non-Blocking non-buffered** send/receive operations of the message passing paradigm.
4. Let's consider that a root process has N child processes. Let's consider that the root process has an array called ***Data*** of size N. Explain the following operations using the array Data.
   a. The root executes the operation ***broadcast*** of the message passing paradigm.
   b. The root executes the operation ***scatter*** of the message passing paradigm.
   c. The root executes the operation ***gather*** of the message passing paradigm.
5. Describe the ***Task Farming*** and the ***Divide-and-Conquer*** programming models and explain the main differences between them.

Q2:
SIMD: Every proccsor has same instruction stream but each proccsor has it's own data streamMIMD:
Every proccsor has it's own instruction stream and data stream

Q3:
Blocking non-buffered: The sender when he send a send operation he will be blocked until the reciver match a recive operation
Non-Blockiing non-bufferd: The sender sends a send operation and then he will continue proccsing until the reciver send an interuption signal

Q4:
A- The Same Data will be copied to each proccses
B- The Data will be split between the procces
C- The Data will be gathered and joined in the root

Q5:
In divide and conquer the sub-tasks have the same nature and will be recursivly decomposed
In master-slave the sub-tasks may have diffrent nature

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 3 – Fall 2021

Question

1. What GPGPU stands for and what does it mean.

General Purpose Graphical Procces Unit, It means that the gpu does the procceses that are usally done by cpu

2. Why CUDA is said Heterogeneous computing.

Becasue there are some portaion of the code that is done in serial by the cpu and other portaion of the code is done in parallel by gpu

3. Give the definition of the following terms:
   a. Device: GPU and it's memory
   b. Kernel. the portaion of the code that will be preformed on the device
   c. Grid of thread blocks. it's a computing model of cuda it consist of orginazing threads in diffrent blocks where each of them is composed of set of threads
   d. Warp.
      groups of 32 thread that always excute same instruction

4. Explain the parallel programming model of CUDA.

Grids composed of set of blocks evry block composed of set threads, threads are grouped of warps that excute the same instruction
warps are time sliced

5. Enumerate and explain the different types of memory adopted by CUDA.

1- register (on chip, fast, per thread, store data up to 32bit)

2- local memory (slow, DRAM ,not cached, per thread)

3- shared memory (on chip , fast , per block, not cached)

4- global memory (DRAM , not cached , per grid)

5- constant (DRAM , read-only , cached , per grid)

6- texture memory (DRAM , cached , read-only)

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 4 – Spring 2021

Question 1

1. Let's consider 2 integer Arrays A and B of dimension N. Let's consider that we would like to write a C program that runs in parallel and that computes the sum of the 2 arrays:

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

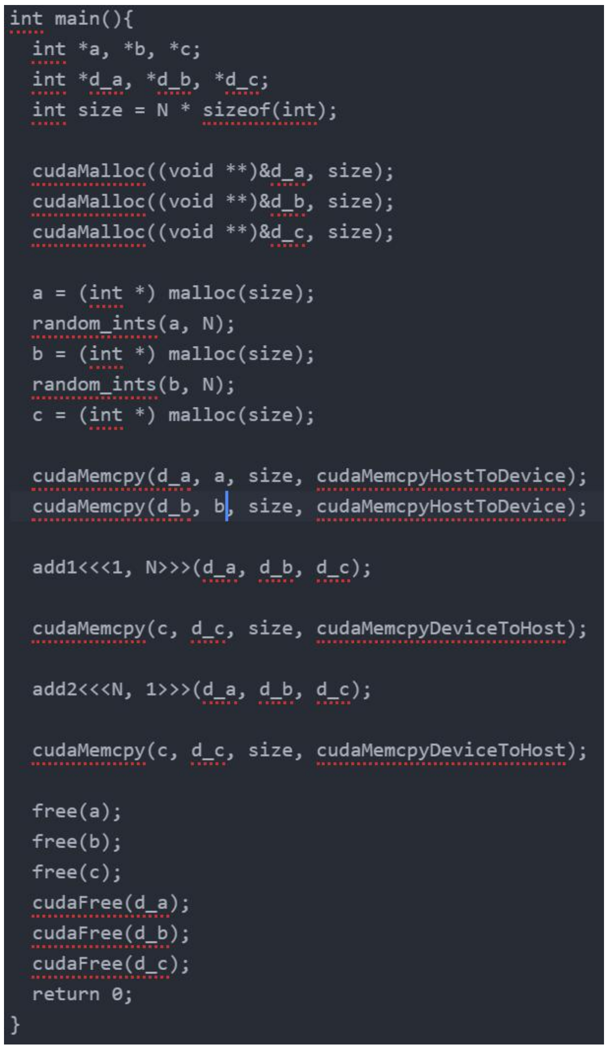
   a. Write the kernel (called ***kernel_1***) that will run on 1 Block of N threads.

```
__global___void add( int *a, int *b, int
*c){ c[threadIdx.x ] = a[ threadIdx.x ] +
b[ threadIdx.x];
}
```

   b. Write another kernel (called ***kernel_2***) that will run on N blocks with 1 thread each.

```
__global___void add( int *a, int *b, int
*c){ c[blockIdx.x ] = a[blockIdx.x ] +
b[blockIdx.x]
}
```

   c. Write the main program that will call both kernels.



```
int main(){
  int *a, *b, *c;
  int *d_a, *d_b, *d_c;
  int size = N * sizeof(int);

  cudaMalloc((void **)&d_a, size);
  cudaMalloc((void **)&d_b, size);
  cudaMalloc((void **)&d_c, size);

  a = (int *) malloc(size);
  random_ints(a, N);
  b = (int *) malloc(size);
  random_ints(b, N);
  c = (int *) malloc(size);

  cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
  cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

  add1<<<1, N>>>(d_a, d_b, d_c);

  cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

  add2<<<N, 1>>>(d_a, d_b, d_c);

  cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

  free(a);
  free(b);
  free(c);
  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);
  return 0;
}
```

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 4 – Spring 2021

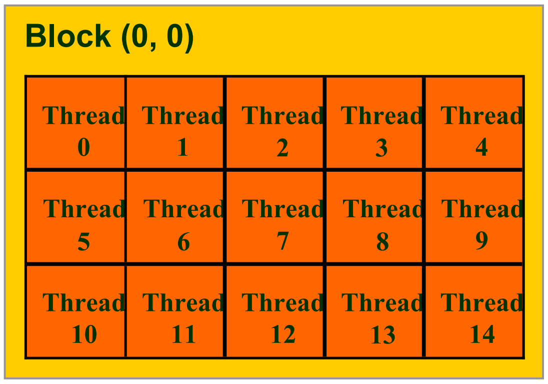
Question 2

Let's consider 2 integer Arrays A and B of dimension N. Let's consider that we would like to write a C program that runs in parallel and that computes the sum of the 2 arrays:

C[index] = A[index] + B[index];

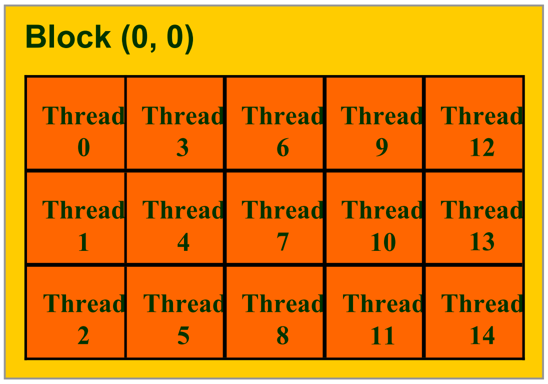
For every configuration of the grid of thread blocks described below, give the statement that computes the index for each each thread:

1. The grid is composed of 1 block and threads should have ids as in the following figure:



```
int index = threadidx.x + ( blockdim.x * threadidx.y );
```

2. The grid is composed of 1 block and threads should have ids as in the following figure:

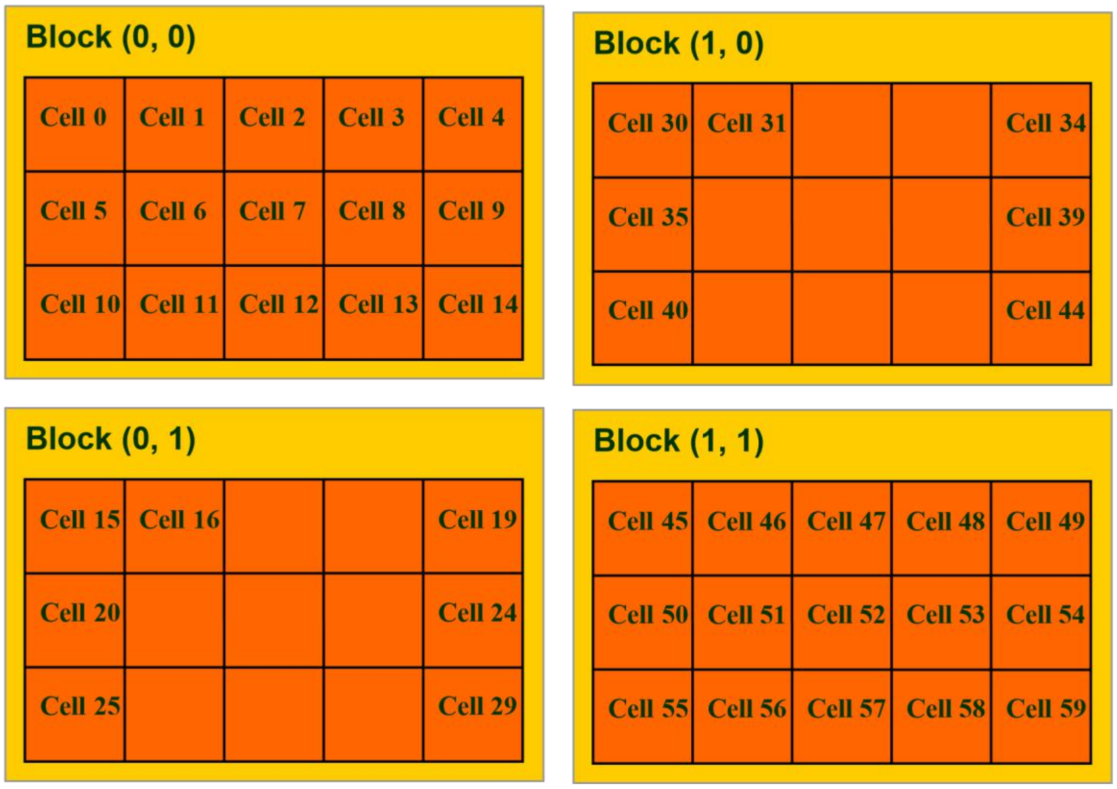


```
int index = threadidx.y + (blockdim.y * threadidx.x);
```

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 5 – Fall 2021

Question 1

We would like to run a 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:



cell_id = (threadIdx.x+threadIdx.y*blockdim.x)
+(blockIdx.y+blockIdx.x*gridDim.y)*(blockDim.x*blockDim.y)

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 6 – Fall 2021

Question 1

Write a C program which do the following:

a. Displays the number of devices available on your computer.

```
int count;

cudaGetDeviceCount(&count);

printf("Number of devices: %d\n", count);
```

b. The number of multiprocessors on every device.

```
int count;
cudaDeviceProp prop;

cudaGetDeviceCount(&count);

int i;
for(i = 0; i < count; i++){
        cudaGetDeviceProperties(&prop, i)
        printf("Count of multiprocessors: %d\n", prop.multiProcessorCount);
}
```

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 7– Fall 2021

General rules:

Number of steps $\log_2$(size) or 2^x = size, where x is number of steps
Sequence length: 2^{step}
n = $2^{step} / 2^{(stage - 1)}$
Shift = n/2
Active threads: thread id % n < shift, if true then thread is active
Step is ascending or descending: thread id / sequence length, if even then ascending, if odd then descending

Question 1

[1] Give the number of steps that are required to sort the elements of the array (size 32).
[2] Give the size of bitonic sequences in every step.
[3] Give the bitonic sequences (just give an interval [b, e] where b is the index of the first cell of the sequence and the e is the index of the last cell of the sequence) that are processed in every step.
[4] Specify, in every step, for every bitonic sequence whether the sequence is sorted in an ascending (+BM) or a descending (-BM) way.

1- Number of steps = $\log_2(32) = 5$ or $2^5 = 32$
2- Step 1 ---> 2, step 2 ---> 4, step 3 ---> 8, step 4 ---> 16, step 5 ---> 32
3-
Step 1 [0,1],[2,3],…,[30,31]
Step 2 [0,3],[4,7],[8,11],[12,15],[16,19],[20,23],[24,27],[28,31]
Step 3 [0,7],[8,15],[16,23],[24,31]
Step 4 [0,15],[16,31]
Step 5 [0,31]
4-
Step 1: +BM[0,1], -BM[2,3],…,-BM[30,31]
Step 2+BM[0,3],-BM[4,7],+BM[8,11],-BM[12,15],+BM[16,19],-BM[20,23],+BM[24,27],-BM[28,31]
Step 3: +BM[0,7],-BM[8,15],+BM[16,23],-BM[24,31]
Step 4:+BM [0,15],-BM[16,31]
Step 5: +BM[0,31]

Question 2

Let's consider an array of integers of size 8. We would like to sort the elements of the array in an ascending way using the Bitonic sort algorithm. We focus on step No 1. Give the following information related to step 1.

[1] Give the IDs of threads involved instep 1.
[2] Give for every thread the bitonic sequence (just give an interval [b, e] where b is the index of the first cell of the sequence and the e is the index of the last cell of the sequence) that the thread is processing.
[3] Specify for every thread whether the corresponding sequence is sorted in an ascending (+BM) or a descending (-BM) way.

1- Threads involved in step 1: T0, T2, T4, T6
2- T0 : [0,1], T2: [2,3], T4: [4,5], T6: [6,7]
3- T0 :+BM [0,1], T2: -BM[2,3], T4: +BM[4,5], T6: -BM[6,7]

Question 3

Let's consider an array of integers of size 8. We would like to sort the elements of the array in an ascending way using the Bitonic sort algorithm. We focus on step No 2. Give the following information related to every stage **i** of step 2:

[1] Give the IDs of threads involved in stage i.
[2] Infer the condition that is satisfyied by the involved thread in stage i .
[3] Give for every thread, involved in stage i , the bitonic sequence (just give an interval [b, e] where b is the index of the first cell of the sequence and the e is the index of the last cell of the sequence) that the thread is processing.
[4] Specify for every thread whether the corresponding sequence is sorted in an ascending (+BM) or a descending (-BM) way.

Stage 1:
1- involved threads: T0, T1, T4, T5
2- Thread id % 4 < 2, n= $2^{step} / 2^{(stage - 1)}$, shift = n/2, Thread id % n < shift
3- T0: [0,2], T1: [1,3], T4: [4,6], T5: [5,7]
4- T0: +BM[0,2], T1: +BM[1,3], T4: -BM[4,6], T5: -BM[5,7]

Stage 2:
1- involved threads: T0, T2, T4, T6
2- Thread id % 2 < 1, n= $2^{step} / 2^{(stage - 1)}$, shift = n/2, Thread id % n < shift
3- T0: [0,1], T2: [2,3], T4: [4,5], T6: [6,7]
4- T0: +BM[0,1], T2: +BM[2,3], T4: -BM[4,5], T6: -BM[6,7]

Let's consider an array of integers of size 8. We would like to sort the elements of the array in an Descending way using the Bitonic sort algorithm. We focus on step No 2.

1.Give the IDs of threads involved in stage i.

Stage 1: T0, T1, T4, T5.
Stage 2: T0, T2, T4, T6.

2.Give for every thread, involved in stage i , the Bitonic sequence (just give an interval [b, e] where b is the index of the first cell of the sequenc and the e is the index of the last cell of the sequence) that the thread is processing.

Stage 1: T0[0, 2], T1[1, 3], T4[4, 6], T5[5, 7].
Stage 2: T0[0, 1], T2[2, 3], T4[4, 5], T6[6, 7].

3.Specify for every thread for each stage whether the corresponding sequence is sorted in an ascending (+BM) or a descending (-BM) way.

Stage 1: T0--> -BM[0, 2], T1--> -BM[1, 3], T4--> +BM[4, 6], T5--> +BM[5, 7].
Stage 2: T0--> -BM[0, 1], T2--> -BM[2, 3], T4--> +BM[4, 5], T6--> +BM[6, 7].

4. Infer the condition that is satisfied by the ascending or descending threads for step 2.

```
(index / 2^step) --ODD--> +BM
                 --EVEN--> -BM
```

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No 8 – Fall 2021

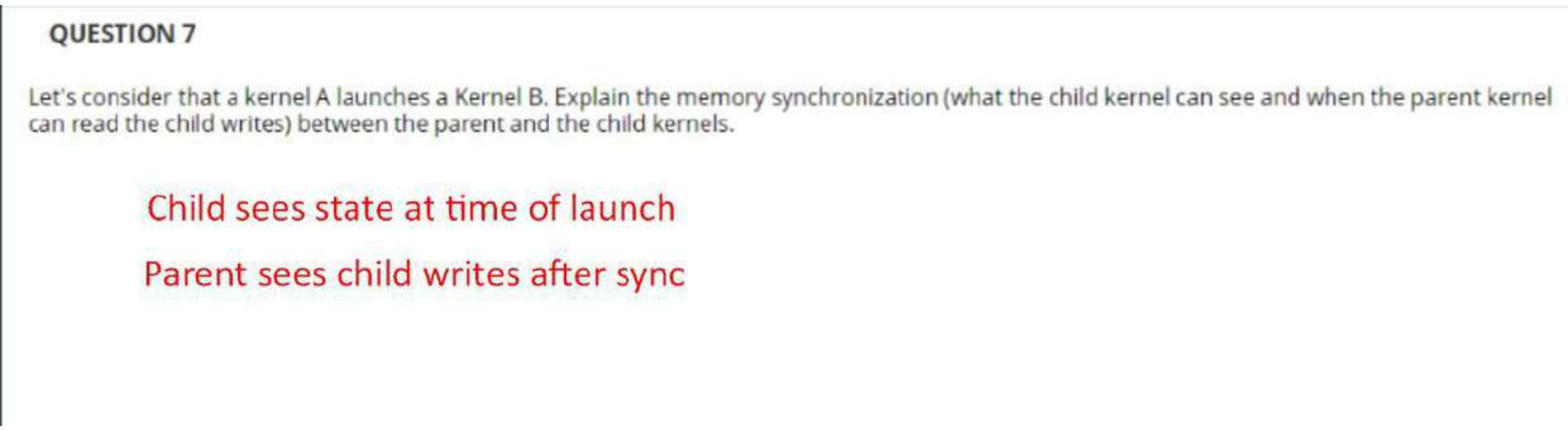
Question

Let’s consider the following parallel 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);
        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, D, B
2. Explain the role of the `__syncthreads()` statements.
   _syncthreads() forces all threads in a block to wait for synchronization at a certain point, this is done to avoid consistency problems
3. Explain the role of the `cudaDeviceSynchronize()` statements.
   It will force caller (CPU or device) process to wail until all threads of all blocks finish

   In case of device “all threads of all blocks to finish” refers to child processes



QUESTION 7

Let's consider that a kernel A launches a Kernel B. Explain the memory synchronization (what the child kernel can see and when the parent kernel can read the child writes) between the parent and the child kernels.

Child sees state at time of launch

Parent sees child writes after sync