King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC453 – Parallel Processing – Tutorial No xx – Quarter 3 - 2023

Question 1

Let's consider the following parallel Java code that calculates, in parallel, the number of occurrences of the number 3 in an array.

```
public class Count3sParallel1 implements Runnable {
	int array[];
	int count, nbThread;
	Thread t;
	LinkedList<Integer> threadIds = new LinkedList<Integer>();

	public void count3s() {
		count =0;
		for (int i=0; i < nbThread; i++) {
			t = new Thread(this);
			threadIds.add(new Integer(i));
			t.start();
		}
	}
	public void run() {
		int depth = (array.length / nbThread);
		int start = threadIds.poll().intValue() * depth;
		int end = start + depth;

		for (int i = start; i < end; i++ ) {
			if (array[i] == 3)
				count ++;
		}
	}
}
```

1. Write using CUDA the kernel that corresponds to the parallel code of the program given above.

2. What is the main problem of this parallel code.

   Race Condition, threads are competing to increment the value of count, there is a risk they overwrite their increment operation

3. How this problem could be fixed.

   a. Describe the techniques that can be used in such case and how.

Mutex, Locks, Semaphore, they guarantee exclusive access to the critical section, threads will not override their increment operation, only one thread can update the variable at a time, so no risk of threads overriding their increment operations. How? we have to acquire an exclusive access to our variable for example atmoicAdd(count, 1)

   b. How the performance of the solution could be enhanced.

We have to reduce the idle time, How? each thread will compute his calculation using localCount and then acquire the exclusive access once per thread instead of 3's time. at the end.

```
_global__ void runKernel(int *array, int *count, int N){
int depth = N / (blockDIm.x * gridDim.x);
int index = blockDim.x * blockIdx.x + threadIdx.x;
int start = index * depth;
int end = start + depth;
int localCount = 0;
for (int i = start; i<end; i++)
   if ( array[i] == 3)
      localCount++;
atomicAdd(count, localCount);}
```

4. Let's assume that we would like to parallelize a serial solution that takes 20 milliseconds to run on a single machine. Let's consider that 40% of this solution is not parallelizable. Let's consider that we will run the parallel solution on 6 processors.
    a. Calculate the execution time of the parallel solution according to Amdahl's law.
    b. Give the speedup of the parallel solution compared to the serial solution.
5. Enumerate and describe the main factors that may cause performance loss of a parallel program.

1)

```
__global__ void runKernel(int *array, int *count, int N){

int depth = N / (blockDIm.x * gridDim.x);
int index = blockDim.x * blockIdx.x + threadIdx.x;
int start = index * depth;
int end = start + depth;

for (int i = start; i<end; i++)
    if ( array[i] == 3)
        *count+=1;
}
```

4) T1 = 20ms, S = 0.4, P = 6

a) Tp = T1(S + (S-1)/P)
= 20(0.4 + 0.6/6) = 10ms

b) SpeedUp = T1/Tp = 20ms/10ms = 2

5)
1- Overhead: any additional cost in parallel solution but not in the serial solution
2- Idle time: threads spend time without work, doing nothing due to lack of work
ex(waiting for external event such as arrival of data)
3- Non-parallelizable computation
4- Contention of resources: Competition of resources, ex race condition. lock contention can reduce performance

extra Q) Sources of Overhead?
1- Communication
2- Synchronization
3- Computation (Parallel computations always perform extra computations that are not needed in the serial solution)
4- Memory