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

Question 1

1. Let's define the **sum of N integers** as follows:

$$\sum_{i=0}^{2^n} x_i = \sum_{i=0}^{2^{n-1}-1} x_i + \sum_{i=2^{n-1}}^{2^n-1} x_i$$

Let's consider that we would like to calculate the **sum of N integers** in a parallel way using the divide and conquer programming model.

Let's consider the following kernel:

__global__ void **sum_Kernel(**int *** data,** int *** res,** int **startingIndex,** int **nbElements);**

- This kernel will calculate: $res = \sum_{i=0}^{nbElements-1} data[i + startingIndex]$

The kernel launched by the main program using the following call:

sum_Kernel<<<1,2>>>(data, res, 0, N);

This will launch a grid composed of 1 block of 2 threads. Every thread will calculate the sum of ***N/2*** elements as follows:

- Thread T_0 will calculate:

$$\sum_{i=0}^{\frac{nbElements}{2}-1} data[i + startingIndex]$$

- Thread T_1 will calculate:

$$\sum_{i=\frac{nbElements}{2}}^{nbElements-1} data[i + startingIndex]$$

Every thread $\mathbf{T_i}$ will calculate 2 values $\mathbf{A_i}$ and $\mathbf{B_i}$ which will be used to calculate the address of elements it will process. Values of $\mathbf{A_i}$ and $\mathbf{B_i}$ for every thread are as follows:

	Thread T_0	**Thread T_1**
A_i	0	1
B_i	1	2

As such, a thread $\mathbf{T_i}$ will calculate

$$\sum_{i=A_i\times\frac{nbElements}{2}}^{B_i\times\frac{nbElements}{2}-1} data[i + startingIndex]$$

a. Give the code that calculates $\mathbf{A_i}$ and $\mathbf{B_i}$ for every thread $\mathbf{T_i}$.

```
int A = threadIdx.x;
int B = A + 1;
```

b. Describe the base case and give the corresponding implementation knowing that it will run on a GPU device.

```
__device__ void baseCase(int *data, int *result, int start){
int sum = data[start] + data[start+1];
atmoicAdd(result, sum);
}
```

c. Give an implementation of the kernel.

PS: `int atomicAdd(int* address, int val)`

This `atomicAdd` function can be called within a kernel. It allows to multiple threads to add concurrently the value ***val*** to the same memory ***address*** without loss of operation.

```
__global__ void kernelSum(int *data, int *result, int startingIndex, int nbElements){

int A = threadIdx.x;
int B = A + 1;
int w = nbElements/2;
int start = A * w + startingIndex;

int localResult  = 0;

if( w == 2)
      baseCase(data, &localResult, start);
else{
      kernelSum<<<1,2>>>(data, &localResult, start, w);
      cudaDeviceSynchronize();
      }

atomicAdd(result, localResult);

}
```

```
__device__ void baseCase(int *data, int *result, int start){

*result = data[start] + data[start + 1];
}
```

Tracing Example