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

Question

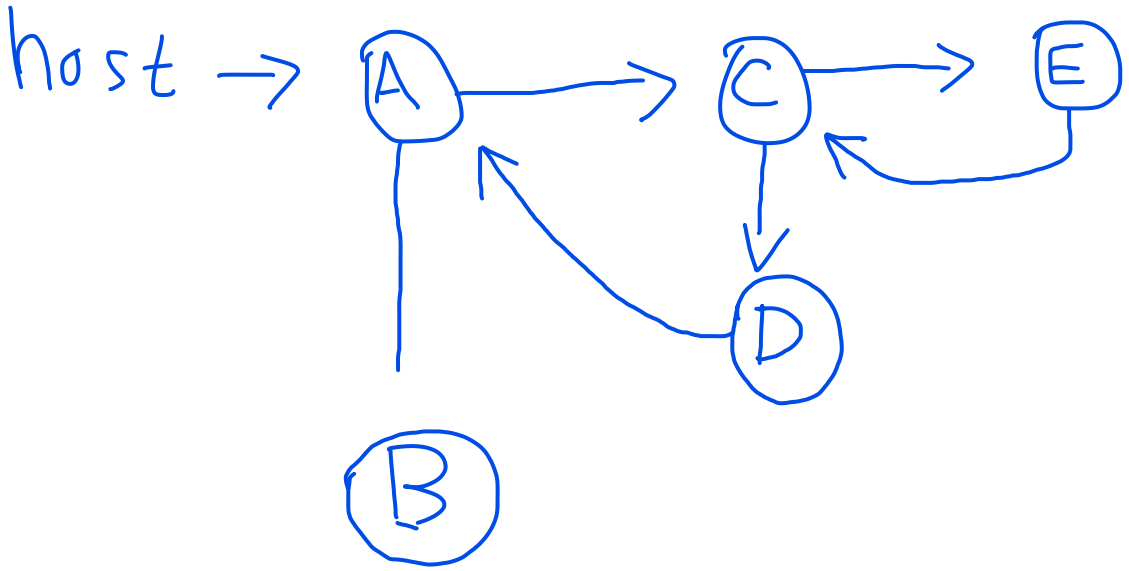
Let's consider the following parallel code:

```
__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.
2. Explain the role of the `__syncthreads()` statements.
3. Explain the role of the **`cudaDeviceSynchronize()`** statements.

__syncthreads() Allows threads of the same block to wait for each other

cudaDeviceSynchronize() the parent thread waits for its child threads to finish,
In HOST the parent waits for its last called kernel



A, C, E, D, B

What is Dynamic Parallesim?

The ability to lunch new kernel from GPU, Dynamically based on run time, Stimulatingly from multiple threads at once, Independently each thread can lunch a different grid.

Serial Code:

```
for i = 0 to N
    for j = 0 to x[i]
        do_something(i, j)
```

```
void main(){
kernel<<< 1, N >>>(x);
}
```

```
void main()
{
callKernel<<< N, max(x) >>>(x…);
}
```

```
__global__ void kernel( … ){
int i = threadIdx.x;
Childkernel<<< 1, x[i]>>>(i)
}
```

```
__global__ void callKernel(int x[], …){
int i = blockIdx.x;
int j = threadIdx.x;
if ( j < x[i])
do_something (i, j)
}
```

```
__global__ void ChildKernel(int i) {
int j = threadIdx.x;
do_something(i, j);
}
```