Homework 6 – not due (for practice only)

Questions 1: Find the bisection bandwidth (the bisection width * link bandwidth), the total network bandwidth (the sum of the bandwidths of all the links), the diameter and the node degree for each of the following network topologies assuming that the bandwidth of each link in a network is B. You can find the definitions of the above measures in the book (Section 6.8) and on the slides.

- A ring network with N nodes (see the figure on page 536)
- A 2-D torus with N nodes (called 2-D grid in Figure 6.14).
- A binary tree network with N nodes (not a fat tree)
- A binary hypercube network with N nodes (N is a power of 2).

Question 2. Given the cuda code for parallel reduction of N integers (copied from class slides)

```c
__global__ void plus_reduce(int *input, int N, int *total)
{
    int tid = threadIdx.x;
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    __shared__ int x[blocksize];
    x[tid] = input[i]; // assuming that N is a multiple of the block size
    __syncthreads();

    for(int s=blockDim.x/2; s>0; s=s/2)
    {
        if(tid < s) x[tid] += x[tid + s];
        __syncthreads();
    }
    if( tid == 0 ) atomicAdd(total, x[tid]);
}
```

Rewrite the kernel without using shared memory.

Question 3. Assuming that you wrote a cuda kernel that declares a shared memory array consisting of 4K bytes and that the compiler determined that each thread in that kernel needs 16 integer registers. Assume also that your GPU has 2 SMs, each with a register file of 2048 integer registers and a shared memory of 16K bytes. If your application will execute kernel <<<nbblocks, blksize>>> , answer the following questions:

a. What is the maximum number of threads that can execute on the GPU

b. What is the maximum number of thread blocks that can execute on the GPU?

c. To execute the maximum number of threads what is the value of nbblocks and blksize that you would use when launching the kernel
**Question 4.** True/False questions:

1. All the threads of a thread block execute in lock-step
2. _syncthreads() is a barrier for all the threads in a thread block
3. Variable declared as _global_ in a CUDA kernel are allocated in the shared memory
4. Shared memory in CUDA is shared by all the threads in a kernel
5. Global memory in CUDA is shared by all the threads in a block
6. cudaMemcpy() can be called from a Kernel to copy data between host and global memory
7. cudaMemcpy() is used to copy data between host and global memory

**Question 5.** Show the content of array A after the execution of the following program:

```c
__global__ F(int *A)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    A[idx] = idx;
    A[blockIdx.x] = blockIdx.x;
    if (idx < blockDim * gridDim / 2) _syncthreads();
}

void main()
{
    Allocate a 16 element int array A in the GPU global memory and initialize its elements to 0;
    F<<<2, 4>>>(A);
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Question 6.** What is wrong with the following cuda code?

```c
__global__ F(int *A)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    A[idx] = idx;
    if (idx < blockDim * gridDim / 2) __syncthreads();
    A[blockIdx.x] = blockIdx.x;
}
```