In the following kernel call in Cuda Kernel<<<< dim3 (8,4,2),

dim3(16,16) >>> (...)

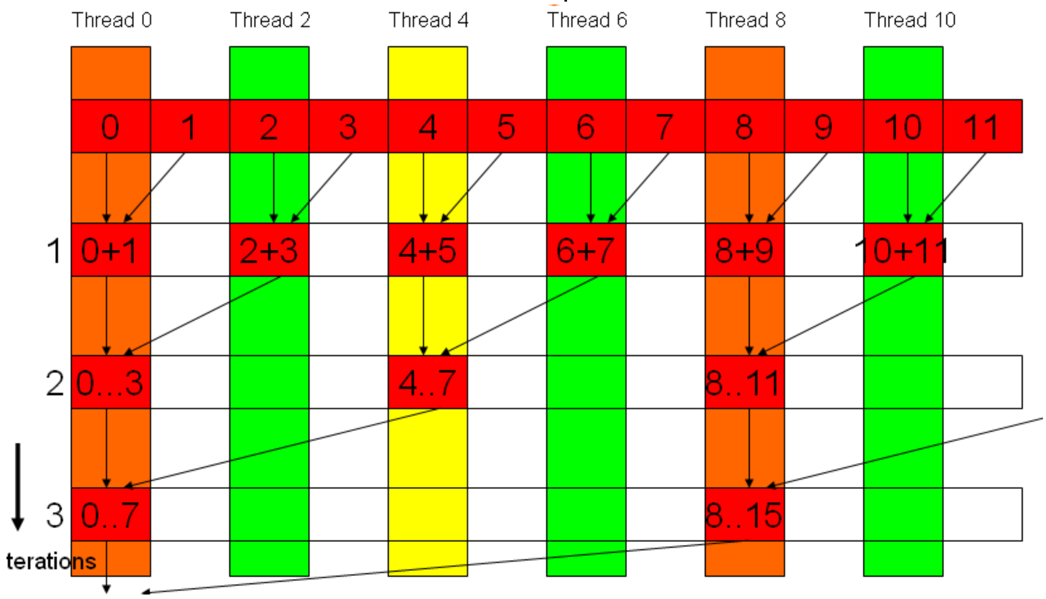
- How many blocks are being launched?
- How many threads per block?
- How many total threads in the Stream Multiprocessor(SM)?

SOL

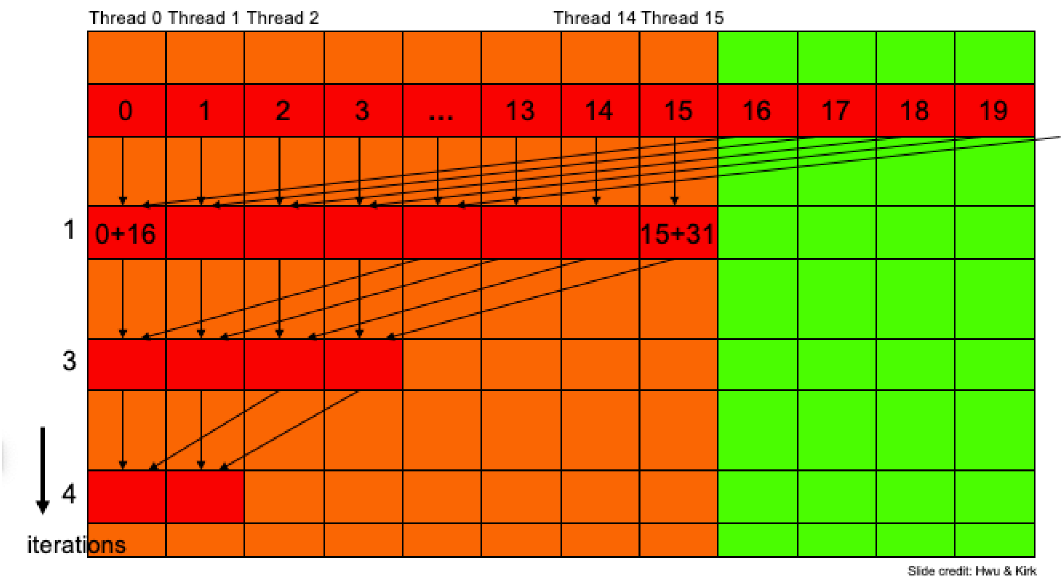
- How many blocks are being launched? 64
- How many threads per block? 256
- How many total threads in the Stream Multiprocessor(SM)? 16364

Exercise 2. For two reduction kernels that are implemented for GPUs as shown in the figures and assuming that they operate with 256 elements in global memory using <u>a block</u> of 256 threads, answer the following questions with justification:

Naïve reduction implementation



Optimized reduction implementation



a. What efficiency and speedup is expected to be achieved for each of the GPU kernels compared to running the serial sum of n=256 elements on a CPU with a two-operand sum operation?

T series (n-1)xTs

T parallel = log (n) x Ts
S= n/log(n)
E=1/log(n)

b. For the kernel called "naïve reduction", how many steps (reduction iterations) are needed?
   8 steps - 128, 64, 32, 16, 16, 8, 4, 2, 1

c. Explain what thread divergence is and indicate for the "naïve reduction" implementation how many steps have divergence indicating the reason.

   8 steps - All, because not all threads in the warps are active

d. For the implementation called "optimized reduction" indicate which steps have thread divergence and which do not?

   5 steps - When the number of active threads is less than warp size

e. Would the kernel named Optimized Implementation benefit from the use of shared memory? Detail how this would be realized and indicate why or why not.

```
 global void reduce0(float *d_in,float *d_out){
   shared float sdata[THREAD_PER_BLOCK];


  //each thread loads one element from global memory to shared mem
  unsigned int i=blockIdx.x*blockDim.x+threadIdx.x;
  unsigned int tid=threadIdx.x;
  sdata[tid]=d_in[i];
   syncthreads();


  // do reduction in shared mem
  for(unsigned int s=1; s<blockDim.x; s*=2){
     if(tid%(2*s) == 0){
         sdata[tid]+=sdata[tid+s];
     }
       syncthreads();
  }


  // write result for this block to global mem
  if(tid==0)d_out[blockIdx.x]=sdata[tid];
}
```

For the following two-vector addition kernel and the corresponding code used for its execution, answer the following questions with justification:

```
1  __global__ void vecAddKernel (float* A, float* B, float* C, int n)
2  {
3    int i = threadIdx.x + blockDim.x * blockIdx.x * 2;
4
5    if (i < n) { C_d[i] = A_d[i] + B_d[i]; }
6    i += blockDim.x;
7    if (i < n) { C_d[i] = A_d[i] + B_d[i]; }
8  }
9
10 int vectAdd (float* A, float* B, float* C, int n)
11 {
12   // Parameter "n" is the length of arrays A, B, and C.
13   int size = n * sizeof (float);
14   cudaMalloc ((void **)&A_d, size);
15   cudaMalloc ((void **)&B_d, size);
16   cudaMalloc ((void **)&C_d, size);
17   cudaMemcpy (A_d, A, size, cudaMemcpyHostToDevice);
18   cudaMemcpy (B_d, B, size, cudaMemcpyHostToDevice);
19
20   vecAddKernel<<<ceil (n / 2048.0), 1024>>> (A_d, B_d, C_d, n);
21   cudaMemcpy (C, C_d, size, cudaMemcpyDeviceToHost);
22 }
```

a. If the size n of vectors A, B and C is 50,000 elements, each. How many thread blocks are generated?

ceil(50000/2048) = 25

b. If the size n of vectors A, B, and C is 50,000 elements, each. How many warps are in each thread block?

1024 threads per block / 32 threads per warp = 32

c. If the size n of vectors A, B and C is 50,000 elements each, how many threads in total are generated in the grid launched in line 20?

25 blocks x 1024 threads = 25600

d. If the size n of vectors A, B and C is 50,000 elements each. Indicate on which elements the first and last thread of the first, second and last block acts.

Block	Thread	First thread	Last thread
0	0	0	1024
0	1023	1023	2047
1	0	2048	3072
1	1023	3071	4095
	0	47104	48128
	1023	48127	49151
	0	49152	50176
	1023	50175	51199

e. If the size n of vectors A, B and C is 50,000 elements, each. Is there divergence of threads in the kernel execution? Explain when it occurs and when it does not occur, identifying the number of blocks and warps with divergence. Justify by identifying the lines of code that have generated the divergence for each case.

Yes, initially both if (i<n). If we go by the items processed by each block in the previous question, only block 24 processes items >= 50000.

Line 5 --> 50000 = Tid + 1024*24*2 --> Tid 848 (Warp 26 Block 24)

Line 7 --> 50000 = Tid + 1024*24*2 + 1024 -> Tid = all (all waprs) --> All values of I > 50000 so there is no divergence in this line.

f. State a performance disadvantage of this vector addition kernel, which computes two elements of the result vector per thread, compared to a kernel that only computes one element of the result vector per thread.

Contrary to what might be expected, there is no further divergence, as the behavior is the same. See "Compare non-divergent warps" section of the profiling.

Memory alignment problems in the kernel that computes 2 elements. Fewer

threads per block

One advantage would be that more work is done per block.

Detailed exercise response using a profiler

Kernel with only 1 element:

```
 global    void vecAddKernel_1 (float* A_d, float* B_d, float* C_d, int
n)
{
              int i = threadIdx.x + blockDim.x * blockIdx.x;
              if (i < n) { C_d[i] = A_d[i] + B_d[i]; }
}


vecAddKernel_1<<<ceil (n / 1024.0), 1024>>>> (A_d, B_d, C_d, n);
```

Runtime information:

vecAddKernel

```
Section: Launch Statistics
---------------------------------------------------------------------- --------------- ------------------------------
Block Size                                                                                                        1,024
Function Cache Configuration                                                                    cudaFuncCachePreferNone
Grid Size                                                                                                            25
Registers Per Thread                                                   register/thread                               16
Shared Memory Configuration Size                                                 Kbyte                            32.77
Driver Shared Memory Per Block                                              byte/block                                0
Dynamic Shared Memory Per Block                                             byte/block                                0
Static Shared Memory Per Block                                              byte/block                                0
Threads                                                                         thread                           25,600
Waves Per SM                                                                                                       0.62
---------------------------------------------------------------------- --------------- ------------------------------
WRN   The grid for this launch is configured to execute only 25 blocks, which is less than the GPU's 40
      multiprocessors. This can underutilize some multiprocessors. If you do not intend to execute this kernel
      concurrently with other workloads, consider reducing the block size to have at least one block per
      multiprocessor or increase the size of the grid to fully utilize the available hardware resources.
```

vecAddKernel_1

```
Section: Launch Statistics
---------------------------------------------------------------------- --------------- ------------------------------
Block Size                                                                                                        1,024
Function Cache Configuration                                                                    cudaFuncCachePreferNone
Grid Size                                                                                                            49
Registers Per Thread                                                   register/thread                               16
Shared Memory Configuration Size                                                 Kbyte                            32.77
Driver Shared Memory Per Block                                              byte/block                                0
Dynamic Shared Memory Per Block                                             byte/block                                0
Static Shared Memory Per Block                                              byte/block                                0
Threads                                                                         thread                           50,176
Waves Per SM                                                                                                       1.23
---------------------------------------------------------------------- --------------- ------------------------------
WRN   If you execute __syncthreads() to synchronize the threads of a block, it is recommended to have more than the
      achieved 1 blocks per multiprocessor. This way, blocks that aren't waiting for __syncthreads() can keep the
      hardware busy.
```

Ocuppancy

vecAddKernel

```
Section: Occupancy
---------------------------------------------------------------------- --------------- ------------------------------
Block Limit SM                                                                   block                             16
Block Limit Registers                                                            block                              4
Block Limit Shared Mem                                                           block                             16
Block Limit Warps                                                                block                              1
Theoretical Active Warps per SM                                                   warp                             32
Theoretical Occupancy                                                                %                            100
Achieved Occupancy                                                                   %                          96.40
Achieved Active Warps Per SM                                                      warp                          30.85
---------------------------------------------------------------------- --------------- ------------------------------
```

vecAddKernel_1

```
Section: Occupancy
---------------------------------------------------------------------- --------------- ------------------------------
Block Limit SM                                                                   block                             16
Block Limit Registers                                                            block                              4
Block Limit Shared Mem                                                           block                             16
Block Limit Warps                                                                block                              1
Theoretical Active Warps per SM                                                   warp                             32
Theoretical Occupancy                                                                %                            100
Achieved Occupancy                                                                   %                          95.02
Achieved Active Warps Per SM                                                      warp                          30.41
---------------------------------------------------------------------- --------------- ------------------------------
```

Compare non-divergent warps

```
vecAddKernel(float*, float*, float*, int), 2022-Dec-16 08:50:21, Context 1, Stream 7
  Section: Command line profiler metrics
  ---------------------------------------------------------------------- --------------- ------------------------------
  sm__sass_average_branch_targets_threads_uniform.pct                                  %                          99.98
  ---------------------------------------------------------------------- --------------- ------------------------------

vecAddKernel_1(float*, float*, float*, int), 2022-Dec-16 08:50:21, Context 1, Stream 7
  Section: Command line profiler metrics
  ---------------------------------------------------------------------- --------------- ------------------------------
  sm__sass_average_branch_targets_threads_uniform.pct                                  %                          99.98
  ---------------------------------------------------------------------- --------------- ------------------------------
```

vecAddKernel

```
Section: Source Counters
---------------------------------------------------------------------- --------------- ------------------------------
Branch Instructions Ratio                                                            %                           0.05
Branch Instructions                                                               inst                          6,327
Branch Efficiency                                                                    %                          99.98
Avg. Divergent Branches                                                                                          0.01
---------------------------------------------------------------------- --------------- ------------------------------
```

vecAddKernel_1

```
Section: Source Counters
---------------------------------------------------------------------- --------------- ------------------------------
Branch Instructions Ratio                                                            %                           0.04
Branch Instructions                                                               inst                          6,263
Branch Efficiency                                                                    %                          99.98
Avg. Divergent Branches                                                                                          0.01
---------------------------------------------------------------------- --------------- ------------------------------
```

Memory

```
vecAddKernel(float*, float*, float*, int), 2022-Dec-16 08:52:21, Context 1, Stream 7
  Section: Memory Workload Analysis
  ---------------------------------------------------------------------- --------------- ------------------------------
  Memory Throughput                                                          Gbyte/second                          48.57
  Mem Busy                                                                              %                           7.13
  Max Bandwidth                                                                         %                          15.43
  L1/TEX Hit Rate                                                                       %                              0
  L2 Hit Rate                                                                           %                          36.58
  Mem Pipes Busy                                                                        %                           5.08
  ---------------------------------------------------------------------- --------------- ------------------------------

vecAddKernel_1(float*, float*, float*, int), 2022-Dec-16 08:52:21, Context 1, Stream 7
  Section: Memory Workload Analysis
  ---------------------------------------------------------------------- --------------- ------------------------------
  Memory Throughput                                                          Gbyte/second                          37.95
  Mem Busy                                                                              %                           6.02
  Max Bandwidth                                                                         %                          12.11
  L1/TEX Hit Rate                                                                       %                              0
  L2 Hit Rate                                                                           %                          36.45
  Mem Pipes Busy                                                                        %                           6.24
  ---------------------------------------------------------------------- --------------- ------------------------------
```