1. If we need to use each thread to calculate one output element of a vector addition, what would be the expression for mapping the thread/block indices to data index:
2. i=threadIdx.x + threadIdx.y;
3. i=blockIdx.x + threadIdx.x;
4. i=blockIdx.x\*blockDim.x + threadIdx.x;
5. i=blockIdx.x \* threadIdx.x;
6. Assume that we want to use each thread to calculate two (adjacent) elements of a vector addition. What would be the expression for mapping the thread/block indices to i, the data index of the first element to be processed by a thread?
7. i=blockIdx.x\*blockDim.x + threadIdx.x +2;
8. i=blockIdx.x\*threadIdx.x\*2
9. i=(blockIdx.x\*blockDim.x + threadIdx.x)\*2
10. i=blockIdx.x\*blockDim.x\*2 + threadIdx.x

Every thread covers 2 consecutive elements. The starting data index is simply twice the global thread index. Another way to look at it is that all previous blocks cover (blockIdx.x\*blockDim.x)\*2. Within the block, each thread covers 2 consecutive elements so the beginning position for a thread is 2\*threadIdx.x.

Basically mapping the data to the threads where each thread takes in 2 data points.

1. We want to use each thread to calculate two elements of a vector addition. Each thread block processes 2\*blockDim.x consecutive elements that form two sections. All threads in each block will first process a section first, each processing one element. They will then all move to the next section, each processing one element. Assume that variable i should be the index for the first element to be processed by a thread. What would be the expression for mapping the thread/block indices to data index of the first element?
2. i=blockIdx.x\*blockDim.x + threadIdx.x +2;
3. i=blockIdx.x\*threadIdx.x\*2
4. i=(blockIdx.x\*blockDim.x + threadIdx.x)\*2
5. i=blockIdx.x\*blockDim.x\*2 + threadIdx.x
6. For a vector addition, assume that the vector length is 8000, each thread calculates one output element, and the thread block size is 1024 threads. The programmer configures the kernel launch to have a minimal number of thread blocks to cover all output elements. How many threads will be in the grid?
7. 8000
8. 8196
9. 8192
10. 8200
11. If we want to allocate an array of v integer elements in CUDA device global memory, what would be an appropriate expression for the second argument of the cudaMalloc() call?
12. n
13. v
14. n \* sizeof(int)
15. v \* sizeof(int)
16. If we want to allocate an array of n floating-point elements and have a floating-point pointer variable d\_A to point to the allocated memory, what would be an appropriate expression for the first argument of the cudaMalloc() call?
17. n
18. (void \*) d\_A
19. \*d\_A
20. (void \*\*) &d\_A
21. If we want to copy 3000 bytes of data from host array h\_A (h\_A is a pointer to element 0 of the source array) to device array d\_A (d\_A is a pointer to element 0 of the destination array), what would be an appropriate API call for this data copy in CUDA?
22. cudaMemcpy(3000, h\_A, d\_A, cudaMemcpyHostToDevice);
23. cudaMemcpy(h\_A, d\_A, 3000, cudaMemcpyDeviceTHost);
24. cudaMemcpy(d\_A, h\_A, 3000, cudaMemcpyHostToDevice);
25. cudaMemcpy(3000, d\_A, h\_A, cudaMemcpyHostToDevice);
26. How would one declare a variable err that can appropriately receive returned value of a CUDA API call?
27. int err;
28. cudaError err;
29. cudaError\_t err;
30. cudaSuccess\_t err;
31. A new summer intern was frustrated with CUDA. He has been complaining that CUDA is very tedious: he had to declare many functions that he plans to execute on both the host and the device twice, once as a host function and once as a device function. What is your response?

The intern most likely is creating inefficient code by double dipping between the host and device. When working with GPU’s you want the vast majority of the code to run in the GPU rather than the CPU; based on his response that he has to write double the amount of code lends to the idea that he’s creating redundant functions.