

Remember: don't use these questions as your sole guide for what to revise, because (along with the ones from the practice test) they're the only questions that are guaranteed **not** to be on the real test! Instead, I suggest using them to test how effectively your revision is going, and to get an idea of the question style / difficulty of the real test.

1. A particular GPU has the following performance specifications:
 - a. It is capable of performing calculations at a rate of 2,000 GFLOPs (giga-floating point operations per second).
 - b. It can transfer 200 gigabytes of data to/from global memory per second.

If a CUDA kernel is to fully utilize the available processing power of this GPU, what's the minimum required compute to global memory access (CGMA) ratio? Assume that the size of a floating point value is 4 bytes.

2. In general, it is preferable to statically allocate arrays in GPU global memory (e.g. by declaring the array with `__device__`) rather than dynamically allocating them (e.g. with `cudaMalloc`). However, static allocation is often not possible in practice – briefly explain why this is.
3. In a CUDA kernel, register spilling occurs when a local variable can't be stored in register memory and is instead stored in a special region of global memory. Although global memory is many times slower than register memory, register spilling does not always result in a big reduction in performance. Why is this?
4. The following table shows some properties relating to the different levels of CUDA Compute Ability (also called Compute Capability):

Technical specifications	Compute ability (version)													
	1.0	1.1	1.2	1.3	2.x	3.0	3.5	3.7	5.0	5.2	5.3	6.0	6.1	
Maximum number of threads per block	512				1024									
Warp size	32													
Maximum number of resident blocks per multiprocessor	8					16			32					
Maximum number of resident warps per multiprocessor	24		32		48		64							
Maximum number of resident threads per multiprocessor	768		1024		1536		2048							

With a Compute Capability 3.0 GPU and a block dimension of (50, 10, 1), what is the maximum number of blocks that can be resident in a multiprocessor? For this question assume that the kernel doesn't make use of any shared memory.

5. The following table shows some properties relating to the different levels of CUDA Compute Ability (also called Compute Capability):

Technical specifications	Compute ability (version)														
	1.0	1.1	1.2	1.3	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3	6.0	6.1	6.2
Maximum number of resident blocks per multiprocessor	8					16				32					
Maximum number of resident warps per multiprocessor	24		32		48		64								
Maximum number of resident threads per multiprocessor	768		1024		1536		2048								
Maximum amount of shared memory per multiprocessor	16 KB				48 KB				112 KB	64 KB	96 KB	64 KB		96 KB	64 KB
Maximum amount of shared memory per thread block	48 KB														

With a Compute Capability 6.0 GPU, a block dimension of (4, 4, 2) and a kernel that uses 1KB of shared memory per block, what is the maximum number of blocks that can be resident in a multiprocessor?

6. A CUDA streaming multiprocessor will stall if all of its resident warps of threads are in the Waiting state, meaning none of them can be executed. Describe one optimization that can be done to your code in order to reduce stalling.
7. A CUDA kernel contains the following code:

```
if (threadIdx.z == 0) {
    doSomething();
} else {
    doSomethingElse();
}
```

This kernel is launched on a grid with dimension (10, 10, 1), where the dimension of each block is (4, 4, 4). Assuming a standard warp size of 32 threads, in how many warps will execution diverge as a result of the `if` statement above? **Hint:** Recall that CUDA linearizes threads by ordering them by z co-ordinate first, then y co-ordinate, then x co-ordinate.

8. The following is a CUDA kernel that takes as input **NUM_THREADS** initial x positions and then calculates a new position for each by simulating **T** time steps (assume **T** is a **const unsigned int** that is defined elsewhere in the code). Each initial x value (and thread) has a unique value for some parameter that controls how the variable will change. The kernel uses Euler's method, but you don't need to know the details of that to answer this question. The function **derivative** involves performing ten floating point operations.

```
__device__ float d_xs[NUM_THREADS];
__constant__ float d_paramValues[NUM_THREADS];

const float dt = 0.01f;

__global__ void kernel()
{
    int globalId = blockIdx.x * blockDim.x + threadIdx.x;
    float x = d_xs[globalId];
    float paramValue = d_paramValues[globalId];

    for (int i = 0; i < T; i++) {
        x += dt * derivative(x, paramValue);
    }
    d_xs[globalId] = x;
}
```

What is the compute to global memory (CGMA) ratio of this kernel? Your answer should be some expression involving the number of steps (T).

9. In C programming it's very common to use a **struct** to store structured data. For example, the position of a particle in four dimensional space (co-ordinates w, x, y, z) could be represented by objects of the following type:

```
typedef struct { float w, x, y, z; } ParticlePosition;
```

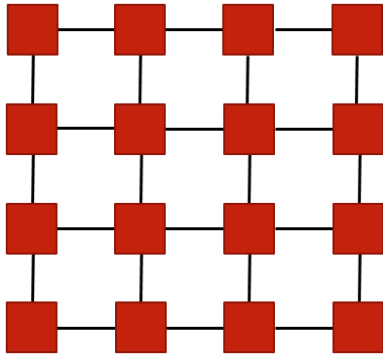
If an application involves simulating **N** particles, an array containing the particles' positions can be allocated in GPU memory with the following code:

```
__device__ ParticlePosition particles[N];
```

If the code is written this way, memory will be organised so that the four co-ordinates associated with the first particle are stored first, then the co-ordinates of the second particle, and so on. Assuming thread *i* processes the movement of particle *i*, why might this not be the optimal way of organising memory? What arrangement could be used instead?

10. Although the design of GPUs is based on a single instruction multiple data (SIMD) approach, there are a number of reasons why a GPU can't be considered a purely SIMD device. Briefly describe one of these reasons.
11. You are told that a program has a speed-up factor of 100 when executed in parallel on 20 processors. You run the program on one processor and it takes 1,500 seconds to complete. How long would you expect the program to take to run on 20 processors, and what would its efficiency be?
12. A program takes 100 seconds to run in parallel with two processors. Ten seconds of this time is spent loading data from disk, which cannot be parallelised, while the remaining 90 seconds consist of code that parallelises perfectly with linear speed-up. How long should you expect the program to take to run on four processors? According to Amdahl's law, what's the theoretical maximum speed-up that could be achieved by adding additional processors?
13. A common model for parallelising a relatively small program involves breaking the task up into multiple threads of execution, each of which can run on different CPU cores but within the same process. Is this an example of a shared memory or distributed memory approach?
14. Simulations of spiking neural networks often only involve transmitting very small amounts of data between different processes. However, for good performance the latency between any given pair of nodes should be small (i.e. messages should arrive quickly). When considering different supercomputers to run a spiking neural network simulation on, which of the following is likely to be the most important for performance: **bisection width**, **diameter** or **valency**? Briefly explain your answer; you should mention whether a smaller value or a larger value is preferable.
15. The diameter of a fat tree network is $2 \log_2 \frac{N+1}{2}$. If you have a fat tree network where it takes 0.5ms to send a message from one node to another, and there are $N=1000$ nodes, what is the maximum expected delay (to the nearest 0.1ms) between a pair of nodes? **Note:** in the actual class test you won't have to calculate \log_2 .

16. Another type of network that we haven't studied is a 2D square network. This is similar to a 2D torus but without the "wrap around" connections at the edges. The diagram below shows a 2D square network with the number of nodes $N=16$.



What is the **bisection width** of this specific network? What will the **valency** be for any 2D square network in general?