Q1. A CUDA kernel loads 4 bytes per thread from global memory and executes with:
Grid size = 512 blocks
Block size = 1024 threads per block
Kernel execution time = 1 millisecond

What is the achieved memory bandwidth?

Solution

Total threads = 512 blocks × 1024 threads = 524,288
Total data = 524,288 × 4 bytes = 2,097,152 bytes
Total Time = 1 ms = 0.001 s
Bandwidth = 2,097,152 / (0.001 × 10^9) = 2.10 GB/s

Q2. A CUDA kernel performs 2 FLOPs per thread. It is launched with:

Grid size = 256 blocks
Block size = 512 threads per block
Execution time = 2 milliseconds

What is the achieved FLOP throughput?

Solution
Total threads = 256 blocks × 512 threads = 131,072
Total FLOPs = 131,072 × 2 = 262,144 FLOPs
Total Time = 2 ms = 0.002 s
FLOP throughput = 262,144 / 0.002 = 131,072,000 FLOPs/sec = 131.07 MFLOP/s

Q3. Develop two version of CUDA program to perform matrix transpose:

- V1: naive implementation which does not use shared memory
- V2: optimized implementation which utilizes shared memory

Solution

V1

```
__global__ void transposeNaive(float* out, const float* in, int width, int height) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x < width && y < height) {
```

```
        int inIdx  = y * width + x;
        int outIdx = x * height + y;
        out[outIdx] = in[inIdx];
    }
}
```

V2

```
// V2: Optimized transpose using shared memory
__global__ void transposeShared(float* out, const float* in, int width, int height) {
    __shared__ float tile[32][32];  // 32x32 tile
    // __shared__ float tile[32][32 + 1];  // 32x32 tile + 1 column padding to avoid bank conflicts

    int xIndex = blockIdx.x * 32 + threadIdx.x;
    int yIndex = blockIdx.y * 32 + threadIdx.y;

    if (xIndex < width && yIndex < height) {
        int inIdx = yIndex * width + xIndex;
        tile[threadIdx.y][threadIdx.x] = in[inIdx];
    }

    __syncthreads();

    xIndex = blockIdx.y * 32 + threadIdx.x;
    yIndex = blockIdx.x * 32 + threadIdx.y;

    if (xIndex < height && yIndex < width) {
        int outIdx = yIndex * height + xIndex;
        out[outIdx] = tile[threadIdx.x][threadIdx.y];
    }
}
```

Host Code to call both v1 and v2

```
dim3 bd(32, 32);
dim3 gd((width + 31) / 32, (height + 31) / 32);

// Call v1
transposeNaive<<<gd, bd>>>(d_out, d_in, width, height);

// Call v2
transposeShared<<<gd, bd>>>(d_out, d_in, width, height);
```

Q4. Choose the correct answer. Mark answer ONLY on the bubble sheet. Use ink pen to fill in the sheet. Cutting, overwriting, partial filling will be considered as incorrect answer.

1. Consider the following kernel:

```
__global__ void coalescedAccess(int *d_arr) {
   int tid = threadIdx.x + blockIdx.x * blockDim.x;
   d_arr[tid] = tid;
}
```

Which statement about the above kernel is correct?

a) Memory accesses are coalesced because threads access consecutive indices.
b) Memory accesses are not coalesced because of warp divergence.
c) Memory accesses are not coalesced because of bank conflicts.
d) Memory accesses are coalesced only when blockDim.x is a power of 2.

2. The following kernel is designed to perform a reduction using shared memory:

```
__global__ void reduce(int *data) {
   __shared__ int smem[32];
   int tid = threadIdx.x;
   smem[tid] = data[tid];
   __syncthreads();
   if (tid % 2 == 0) smem[tid] += smem[tid + 1];
   __syncthreads();
   data[tid] = smem[tid];
}
```

What is the primary issue with this implementation?

a) The kernel will not compile because smem[tid + 1] may be out of bounds.
b) The memory access pattern causes shared memory bank conflicts.
c) The kernel requires more registers per thread than available.
d) The use of __syncthreads() is incorrect in a reduction kernel.

3. What is a key advantage of using Unified Memory (cudaMallocManaged()) in CUDA?

a) It eliminates the need for explicit memory transfers between host and device.
b) It provides lower latency than global memory.
c) It requires manual memory deallocation on the GPU.
d) It is only supported on CUDA devices with compute capability 8.0+.

4. What will be the output of the following CUDA kernel?

```
__global__ void shared_memory_example() {
    __shared__ int smem;
    if (threadIdx.x == 0) smem = 10;
    __syncthreads();
    printf("Thread %d: smem = %d\n", threadIdx.x, smem);
}
```

a) All threads print smem = 10
b) Some threads may print an uninitialized value
c) Only thread 0 prints smem = 10
d) Compilation error

5. Consider the following CUDA kernel launch:

```
dim3 gridDim(2, 2);
dim3 blockDim(16, 16);
kernel<<<gridDim, blockDim>>>();
```

How many total threads will execute this kernel?

a) 64
b) 128
c) 512
d) 1024

6. Which of the following strategies helps reduce global memory latency?
a) Using atomic operations
b) Using more thread divergence
c) Prefetching data into shared memory
d) Reducing the number of registers

7. What is memory coalescing in CUDA?
a) A method for reducing register usage
b) A technique for optimizing global memory accesses
c) A way to increase shared memory
d) A strategy for increasing clock speed

8. Which of the following is NOT a valid CUDA execution configuration syntax?
a) kernel<<<blocks, threads>>>(args)
b) kernel<<<blocks, threads, shared_memory>>>(args)
c) kernel<<<blocks, threads, shared_memory, stream>>>(args)
d) kernel<<<threads, blocks>>>(args)

9. In CUDA, what is a warp?

a) A group of 32 threads executing in parallel
b) A block of memory allocated for a kernel
c) A single instruction executed by the CPU
d) A shared memory segment for a thread block

10. What is the problem with this kernel?

```
__global__ void bankConflict(int *data) {
    __shared__ int smem[32];
    int tid = threadIdx.x;
    smem[tid * 2] = data[tid];
    __syncthreads();
    data[tid] = smem[tid * 2];
}
```

a) It will cause memory coalescing issues.
b) It will cause shared memory bank conflicts.
c) The memory accesses are misaligned.
d) The kernel will not compile.

11. What determines the maximum number of active warps per multiprocessor (SM) in CUDA?

a) Number of registers per thread and shared memory usage
b) Number of threads per block only
c) The number of global memory accesses
d) The size of the L2 cache

12. What is a drawback of using Unified Memory (cudaMallocManaged())?

a) Requires explicit memory copying between CPU and GPU
b) Causes potential page faults and migration overhead
c) Increases shared memory bank conflicts
d) Reduces available registers per thread

13. What is the best way to reduce warp divergence in the following kernel?

```
__global__ void warpDivergence(int *data) {
    int tid = threadIdx.x;
    if (tid % 2 == 0)
        data[tid] += 10;
    else
        data[tid] -= 5;
}
```

a) Use __syncthreads() before the conditional statements
b) Replace the if-else with arithmetic operations
c) Increase the number of blocks in the kernel launch
d) Use __shared__ memory instead of global memory

14. A kernel has theoretical bandwidth = 900 GB/s, but the measured bandwidth = 450 GB/s. What is the likely reason?

a) Memory coalescing issues
b) Too many floating-point operations
c) Excessive shared memory usage
d) Too many warps per SM

15. What is the issue in this code?

```
int main() {
    int *d_data;
    cudaMalloc(&d_data, 256 * sizeof(int));

    cudaStream_t stream;
    cudaStreamCreate(&stream);

    kernel<<<1, 256, 0, stream>>>(d_data, 42);

    cudaStreamDestroy(stream);
    cudaFree(d_data);
}
```

a) The stream is destroyed before the kernel completes execution.
b) All kernels in different streams execute sequentially.
c) The kernel cannot execute in a non-default stream.
d) cudaStreamSynchronize() is missing, so the kernel won’t execute.