COL 380/730/7880	QUIZ 2	March 24, 2026

Name _____ *EntryNo* _____

Note (i) Complete your answers in the provided space.
(ii) Justify all answers. Be clear, brief and precise. There will no later opportunity to clarify further.
(iii) Write your name on top of each sheet of the paper.
(iv) Use the rough sheet for rough work. Answers in the provided space must be legible.

Problem 1: 5 Marks

A GPU's Shared Memory is divided into 32 banks, each with a width of 32 bits. Consider a kernel where each thread in a warp accesses an element in shared memory using the following indexing logic:

$$\text{Address} = \text{Base} + ((\text{threadIdx.x} \times \text{stride}) \pmod N)$$

. If $stride = 2$ and the data type is `float32`, describe the degree of bank conflicts. How does this change if the data type is changed to `float64`.

Problem 2: 3 Marks

Occupancy (the ratio of active warps running on a streaming multiprocessor (SM) to the maximum supported warps) is not performance. Explain with three distinct reasons why lower occupancy kernel may still outperform the higher occupancy one.

Problem 3: 3 Marks

In a program execution, `Block 0` writes $data[i] = value$; and then sets $flag = 1$; `Block 1` spins until $flag == 1$ and then reads $data[i]$. Explain why the above access pattern is unsafe. How will you correctly implement it?

Problem 4: 4 Marks

Consider two warps in the same thread block that perform an `atomicAdd` to a single shared memory location. Warp A executes the atomic operation before calling *__syncthreads*(); Warp B executes its atomic operation after the *__syncthreads*() barrier.

Which of the following statements are correct? Select all that apply. For the ones that are incorrect, provide brief explanation.

1. Both atomic operations are guaranteed to be visible to all threads in the block once they have all passed the *__syncthreads*() barrier.
2. `atomicAdd` to shared memory is an intra-SM operation; therefore, no inter-SM (L2 or global) coherence protocols are triggered.
3. The hardware scheduler guarantees that Warp A's write is visible to Warp B because they reside in the same thread block.
4. Because `atomicAdd` is an atomic operation, it inherently includes a block-wide synchronization, making the *__syncthreads*() call redundant in this scenario.

Problem 5: 6 Marks

A developer runs the following kernel and gets a wrong answer. Identify all bugs, classify each as a correctness bug or a performance bug.

```
__global__ void reduce(float *d_in, float *d_out, int n) {
    extern __shared__ float sdata[];
    int tid = threadIdx.x;
    int i   = blockIdx.x * blockDim.x + threadIdx.x;
    sdata[tid] = (i < n) ? d_in[i] : 0.0f;
    for (int s = 1; s < blockDim.x; s *= 2) {
        if (tid % (2*s) == 0)
            sdata[tid] += sdata[tid + s];
        __syncthreads();
    }
    if (tid == 0) d_out[blockIdx.x] = sdata[0];
}
```

Problem 6: 4 Marks

For the following program what would be the output of the print statement and why?

```
  __global__ void fill(int *d) { *d = 99; }

int main() {
    int h = 0, *d;
    cudaStream_t s;
    cudaStreamCreate(&s);
    cudaMalloc(&d, sizeof(int));
    fill <<< 1, 1, 0, s >>>(d);
    cudaMemcpyAsync(&h, d, sizeof(int), cudaMemcpyD2H, s);
    printf("%d\n", h);
    cudaStreamSynchronize(s);
    cudaStreamDestroy(s);
    cudaFree(d);
}
```