National University of Computer and Emerging Sciences

Islamabad Campus

High Performance Computing with GPUs (CS4110)

Course Instructor(s):
Dr. Imran Ashraf
Section(s): A, B, C

Sessional-II Exam

Total Time (Hrs):	**1**
Total Marks:	**55**
Total Questions:	**4**

Date: Nov 6, 2025

Roll No	**Course Section**	**Student Signature**

Do not write below this line.

1. **Attempt all the questions.**
2. **Each question should be solved in the specified number of lines.**
3. **A bonus mark will be awarded for solving all the questions and all parts in order.**

CLO 2: Identify application hotspots based on the application profile using realistic work-load. **[Analyze, C4]**

Q1: **[10 marks]**

You are simulating the gravitational interaction of N particles in space. Each particle **i** has position **r**$_i$=(x_i,y_i,z_i), velocity **v**$_i$, mass **m**$_i$. The gravitational acceleration on particle i due to all other particles is given by:

$$\vec{a}_i = G \sum_{j \neq i} \frac{m_j(\vec{r}_j - \vec{r}_i)}{(|\vec{r}_j - \vec{r}_i|^2 + \varepsilon^2)^{3/2}}$$

where G is the gravitational constant and ε is a small softening term to avoid division by zero. Following code implements this simulation on CPU.

```
void computeAccelerationCPU( float3 *pos, float3 *vel, float3 *acc, float *mass, int N, float G, float
eps, float dt)
{
   for (int i = 0; i < N; i++) {
      float3 a = {0.0f, 0.0f, 0.0f};
      for (int j = 0; j < N; j++) {
            if (i == j) continue;
            float3 r;
            r.x = pos[j].x - pos[i].x;
            r.y = pos[j].y - pos[i].y;
            r.z = pos[j].z - pos[i].z;
            float distSqr = r.x * r.x + r.y * r.y + r.z * r.z + eps * eps;
            float invDist = rsqrtf(distSqr);
            float invDist3 = invDist * invDist * invDist;
            float F = G * mass[j] * invDist3;
            a.x += F * r.x;
            a.y += F * r.y;
            a.z += F * r.z;
        }
        acc[i] = a;

        // Euler update
        vel[i].x += a.x * dt;
        vel[i].y += a.y * dt;
        vel[i].z += a.z * dt;
        pos[i].x += vel[i].x * dt;
        pos[i].y += vel[i].y * dt;
        pos[i].z += vel[i].z * dt;
    }
}
```

Hint: In CUDA, float3 is a built-in vector type that represents a 3-component floating-point vector.

1) Do you need to utilize shared memory in this application to improve performance (Yes/No)? Why (2 lines) and Identify where (2 lines)?
2) Do you need atomic operations in this application for correct execution (Yes/No)? Why (2 lines) and identify where (2 lines)?

SOLUTION:

A)

- Yes.
- (WHY) Shared memory can reduce redundant global memory reads of particle data when each thread computes forces for one particle.
- (WHERE) Use shared memory to load blocks of particle positions and masses (pos[j], mass[j]) so that all threads in a block reuse them during the inner loop.

B)

- No.
- (WHY) Each thread updates only its own particle's acceleration, velocity, and position—no shared data is written concurrently.
- (WHERE) Hence, no atomic operations are needed anywhere in the kernel.

CLO 4: Analyze performance of an application running on an HPC system to improve compute and memory performance. **[Evaluate, C5]**

Q2: **[4+4+2 = 10 marks]**

A CUDA kernel performs a vector operation as shown below:

```
__global__ void vecFMA(float *A, float *B, float *C, int N, int stride) {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int idx = i * stride;
   if (idx < N) {
      C[idx] = A[idx] * B[idx] + C[idx];
   }
}
```

Each warp contains 32 threads, and each element is a 4-byte float.

The GPU has a **128-byte global memory transaction size**.

1) For **stride = 1**, estimate how many global memory transactions are needed for each warp to read **A**, assuming perfectly coalesced access.
2) For **stride = 8**, estimate how many memory transactions are needed per warp for reading **A**. Assume each thread reads one float, and no caching occurs.
3) Compute the ratio of effective bandwidth utilization (throughput efficiency) between the stride=1 and stride=8 cases.

SOLUTION:

1. 1: one 128-byte transaction (32 threads × 4 B = 128 B, perfectly coalesced).
2. 8: with stride=8 each 128-byte block supplies 4 threads, so 32/4 = 8 transactions.
3. 8×: stride=1 uses 128 B useful / 128 B fetched = 100% ; stride=8 uses 128 B useful / (8×128 B fetched) = 12.5% ⇒ 100%/12.5% = 8× (stride=1 is 8× more efficient).

CLO 1: Describe the terms related to HPC systems, GPU architecture and GPU programming models. **[Understand, C2]**

Q3: **[6+4+4+2+4 = 20 marks]**

Provide short answers to each of the following questions. No marks will be awarded if the answer exceeds the mentioned number of lines.

1) Provide three use-case which justify shared memory usage (2 lines per case).
2) Give two reasons for adding extra **padding**" elements to arrays allocated in GPU global memory? Each reason should not be more than 2 lines.
3) Give two potential disadvantages associated with increasing the amount of work done in each CUDA thread, such as loop unrolling techniques, using fewer threads in total? Each disadvantage should not be more than 2 lines.
4) What is programmer-managed memory (2 lines)?
5) Provide two advantages of a programmer-managed memory over a hardware managed cache (2 lines per advantage)?

SOLUTION:

1) Shared Memory Use Cases

1. Global Memory Data Reuse: A block can load a section of global memory into shared memory, letting multiple threads reuse the data with significantly lower latency.
2. Faster atomic operations in shared memory as compared to global memory.
3. Use of shared memory as intermediate temporary memory results in coalesced global memory access.

B) Reasons for GPU Global Memory Padding

1. Avoid Shared memory bank conflicts
2. Optimize Global Memory Access (Coalescing): Padding can ensure that the array elements accessed by a warp are contiguous in global memory, allowing for coalesced memory access.

C) Disadvantages of Increasing Per-Thread Work

1. More complex thread kernels require more registers per thread, potentially limiting the number of active warps and hiding memory latency less effectively.
2. Increased Divergence and Synchronization Overhead: If unrolled loops introduce complex control flow or more thread synchronization points, it can increase warp divergence and overall overhead.

D) Programmer-Managed Memory

Programmer-managed memory (like shared memory) is explicitly managed by the developer for data storage and reuse, providing direct control over its usage and lifespan within a thread block.

E) Advantages of Programmer-Managed Memory over Hardware-Managed Cache

1. Predictable Performance: The programmer controls what to place in this fast memory and when it is evicted, allowing for deterministic performance optimization unlike an automatic cache.
2. Explicit Data Reuse Strategy: Enables custom, data reuse strategies that may be more optimal than a general-purpose hardware cache policy.

National University of Computer and Emerging Sciences
Islamabad Campus

CLO 1: Describe the terms related to HPC systems, GPU architecture and GPU programming models. **[Understand, C2]**

Q4: **[15 marks]**

Choose the correct answer. Solve this part on the bubble sheet attached at the end of the question paper.

1. What is the main purpose of the Uniform Memory Model in NVIDIA GPUs?

A. To give each GPU thread its own private memory region

B. To enable a single shared virtual address space between CPU and GPU

C. To unify constant and texture memory

D. To eliminate all cache hierarchies in GPU

2. Under the Uniform Memory Model, which CUDA feature allows both CPU and GPU to access the same memory allocation without explicit copies?

A. Zero-copy memory (cudaHostAllocMapped)

B. Managed memory (cudaMallocManaged)

C. Device memory (cudaMalloc)

D. Constant memory (__constant__)

3. What is the main performance penalty when relying on on-demand page migration in Unified Memory?

A. Increased arithmetic latency on CUDA cores

B. Page faults that stall execution as data migrates between host and device

C. Shared memory bank conflicts

D. Reduced warp occupancy due to thread divergence

4. How can a programmer minimize page-fault latency in Unified Memory allocations?

A. Use smaller page sizes to increase migration granularity

B. Prefetch data explicitly using cudaMemPrefetchAsync() before kernel execution

C. Disable L2 cache to speed up memory access

D. Launch more threads per block to hide latency

5. What is the primary characteristic of pinned (page-locked) memory on the host?

A. It can be swapped out by the operating system at any time.

B. It resides permanently in physical memory and cannot be paged out.

C. It exists only during GPU kernel execution.

D. It must be allocated inside GPU global memory.

6. Why is pinned memory typically faster for host–device transfers compared to pageable memory?

A. It uses higher clock frequencies for memory access.

B. It allows the GPU to perform Direct Memory Access (DMA) without first staging through a temporary buffer.

C. It bypasses the L2 cache hierarchy.

D. It enables asynchronous kernel launches.

7. Which CUDA API function is used to allocate pinned host memory?

A. `cudaMalloc()`

B. `cudaMallocHost()`

C. `cudaHostAlloc()`

D. Both B and C

8. What is a potential disadvantage of using too much pinned memory?

A. It reduces L2 cache capacity on the GPU.

B. It can lead to system instability due to reduced pageable memory available for the OS.

C. It slows down PCIe bandwidth.

D. It disables concurrent kernel execution.

9. Which of the following statements about the default stream (stream 0) is TRUE in modern CUDA under the per-thread default stream model?

A. It blocks operations in all other streams.

B. It executes asynchronously and independently of other streams.

C. It forces serialization across all streams globally.

D. It is only used for host-to-device memory transfers.

10. What happens when you launch multiple asynchronous operations (copies or kernels) into the same stream?

A. The GPU executes them in parallel, unordered.

B. They execute in the order issued, with implicit synchronization.

C. They execute randomly depending on warp scheduling.

D. They block other streams automatically.

11. The purpose of CUDA cores inside each SM is to:

A. Manage memory allocation

B. Perform arithmetic and logic operations for threads

C. Handle kernel scheduling

D. Execute host-side operations

12. What is the main function of the warp scheduler in NVIDIA architecture?

A. Manage shared memory accesses

B. Schedule and issue instructions to CUDA cores

C. Allocate registers to threads

D. Handle PCIe communication

13. Which of the following correctly matches memory hierarchy from fastest to slowest?

A. Global → Shared → Registers

B. Registers → Shared → Global

C. Shared → Registers → Global

D. Constant → Registers → Shared

14. What is the main purpose of atomic operations in CUDA?

A. To increase thread divergence

B. To ensure that multiple threads can safely update shared data without race conditions

C. To synchronize host and device execution

D. To accelerate floating-point math operations

15. Which of the following is a valid CUDA atomic operation?

A. atomicAdd()

B. atomicSub()

C. atomicCAS()

D. All of the above

SOLUTION:

1. B
2. B
3. B
4. B
5. B
6. B
7. D
8. B
9. B
10. B
11. B
12. B
13. B
14. B
15. D