CEN310 Parallel Programming

Week-11 (Advanced GPU Programming)

Spring Semester, 2024-2025

Overview

Topics

1. CUDA Memory Model
2. Shared Memory Optimization
3. Thread Synchronization
4. Performance Optimization Techniques

Objectives

- Understand CUDA memory hierarchy
- Learn shared memory usage
- Master thread synchronization
- Implement optimization strategies

Memory Types

- Global Memory
- Shared Memory
- Constant Memory
- Texture Memory
- Registers

Memory Access Patterns

```
// Coalesced memory access example
__global__ void coalesced_access(float* data, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        // Coalesced access pattern
        float value = data[idx];
        // Process value
        data[idx] = value * 2.0f;
    }
```

2. Shared Memory Optimization

Using Shared Memory

```
__global__ void matrix_multiply(float* A, float* B, float* C, int N) {
    __shared__ float sharedA[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float sharedB[BLOCK_SIZE][BLOCK_SIZE];

    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    float sum = 0.0f;

    for(int tile = 0; tile < N/BLOCK_SIZE; tile++) {
        // Load data into shared memory
        sharedA[threadIdx.y][threadIdx.x] =
            A[row * N + tile * BLOCK_SIZE + threadIdx.x];
        sharedB[threadIdx.y][threadIdx.x] =
            B[(tile * BLOCK_SIZE + threadIdx.y) * N + col];

        __syncthreads();

        // Compute using shared memory
        for(int k = 0; k < BLOCK_SIZE; k++) {
            sum += sharedA[threadIdx.y][k] * sharedB[k][threadIdx.x];
        }

        __syncthreads();
    }

    C[row * N + col] = sum;
}
```

3. Thread Synchronization

Synchronization Methods

- Block-level synchronization
- Grid-level synchronization
- Atomic operations

Example: Atomic Operations

```
__global__ void histogram(int* data, int* hist, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        atomicAdd(&hist[data[idx]], 1);
    }
}
```

4. Performance Optimization

Optimization Techniques

1. Memory Coalescing
2. Bank Conflict Avoidance
3. Occupancy Optimization
4. Loop Unrolling

Example: Bank Conflict Resolution

```
// Bad: Bank conflicts
__shared__ float shared_data[BLOCK_SIZE][BLOCK_SIZE];

// Good: Padded to avoid bank conflicts
__shared__ float shared_data[BLOCK_SIZE][BLOCK_SIZE + 1];
```

Advanced Memory Management

Unified Memory

```
// Allocate unified memory
float* unified_data;
cudaMallocManaged(&unified_data, size);

// Access from host or device
// No explicit transfers needed
kernel<<<grid, block>>>(unified_data);

// Free unified memory
cudaFree(unified_data);
```

Stream Processing

Concurrent Execution

```
cudaStream_t stream1, stream2;
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);

// Asynchronous operations in different streams
kernel1<<<grid, block, 0, stream1>>>(data1);
kernel2<<<grid, block, 0, stream2>>>(data2);

cudaStreamSynchronize(stream1);
cudaStreamSynchronize(stream2);

cudaStreamDestroy(stream1);
cudaStreamDestroy(stream2);
```

Dynamic Parallelism

Nested Kernel Launch

```
__global__ void child_kernel(float* data) {
    // Child kernel code
}

__global__ void parent_kernel(float* data) {
    if(threadIdx.x == 0) {
        child_kernel<<<grid, block>>>(data);
        cudaDeviceSynchronize();
    }
}
```

Lab Exercise

Tasks

1. Implement matrix multiplication with shared memory
2. Compare performance with global memory version
3. Analyze memory access patterns
4. Optimize for different GPU architectures

Performance Metrics

- Execution time
- Memory throughput
- Occupancy
- Cache hit rate

Resources

Documentation

- CUDA C++ Programming Guide
- CUDA Best Practices Guide
- GPU Computing Webinars

Tools

- Nsight Compute
- CUDA Profiler
- Visual Studio GPU Debugger

Questions & Discussion


RECEP TAYYIP
ERDOĞAN
ÜNİVERSİTESİ