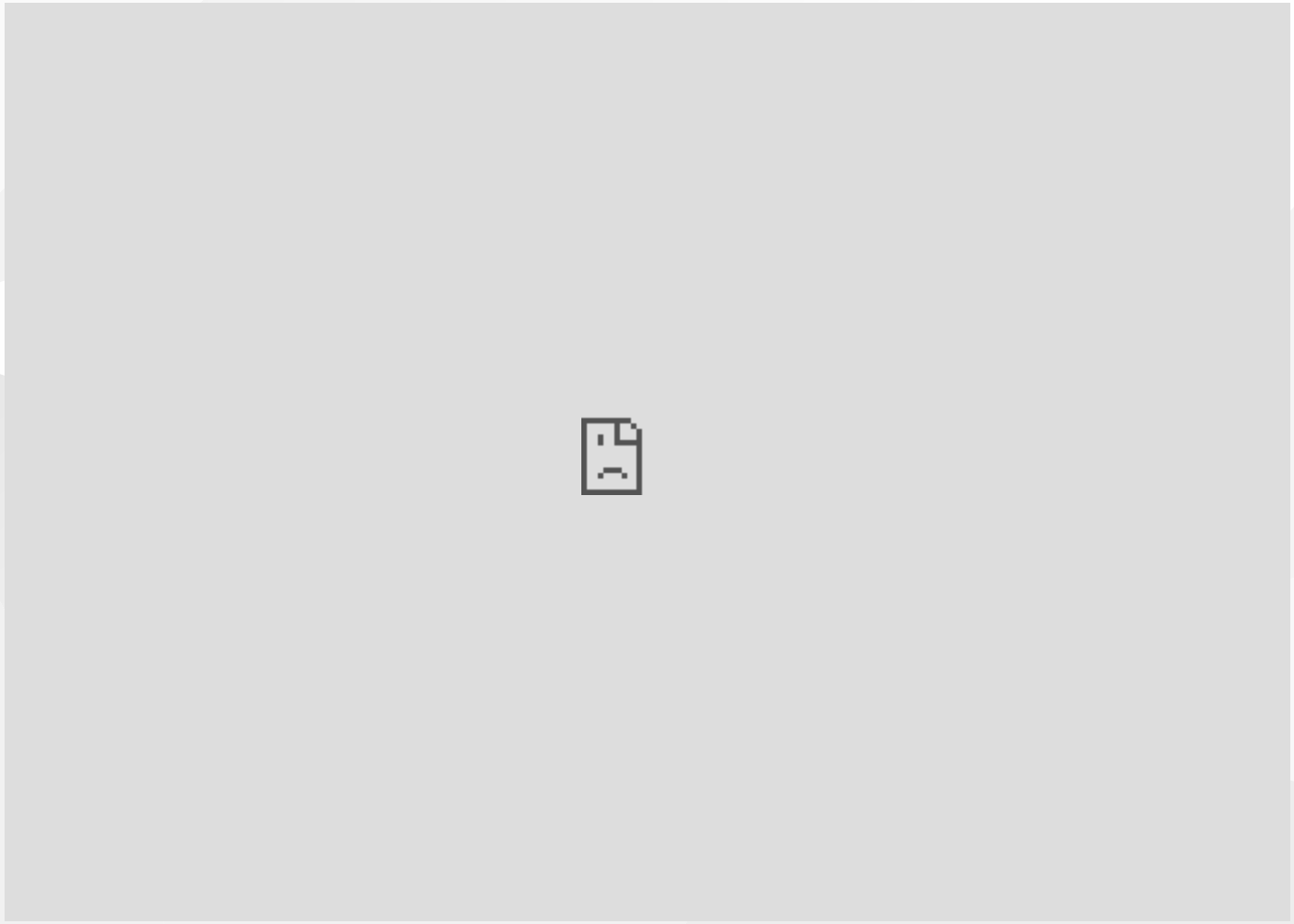
CEN310 Parallel Programming

Week-1

Course Introduction and Development Environment Setup

Download

- PDF
- DOC
- SLIDE
- PPTX



Outline (1/3)

1. Course Overview
    - Course Description
    - Learning Outcomes
    - Assessment Methods
    - Course Topics
2. Development Environment Setup
    - Required Hardware
    - Required Software
    - Installation Steps

Outline (2/3)

3. Introduction to Parallel Programming
    - What is Parallel Programming?
    - Why Parallel Programming?
    - Basic Concepts
4. First Parallel Program
    - Hello World Example
    - Compilation Steps
    - Running and Testing

Outline (3/3)

5. Understanding Hardware
    - CPU Architecture
    - Memory Patterns
6. Performance and Practice
    - Parallel Patterns
    - Performance Measurement
    - Homework
    - Resources

1. Course Overview

Course Description

This course introduces fundamental concepts and practices of parallel programming, focusing on:

- Designing and implementing efficient parallel algorithms
- Using modern programming frameworks
- Understanding parallel architectures
- Analyzing and optimizing parallel programs

Learning Outcomes (1/2)

After completing this course, you will be able to:

1. Design and implement parallel algorithms using OpenMP and MPI
2. Analyze and optimize parallel program performance
3. Develop solutions using various programming models

Learning Outcomes (2/2)

4. Apply parallel computing concepts to real-world problems
5. Evaluate and select appropriate parallel computing approaches based on:
    - Problem requirements
    - Hardware constraints
    - Performance goals

Assessment Methods

Assessment	Weight	Due Date
Midterm Project Report	60%	Week 8
Quiz-1	40%	Week 7
Final Project Report	70%	Week 14
Quiz-2	30%	Week 13

Course Topics (1/2)

1. Parallel computing concepts
    - Basic principles
    - Architecture overview
    - Programming models
2. Algorithm design and analysis
    - Design patterns
    - Performance metrics
    - Optimization strategies

Course Topics (2/2)

3. Programming frameworks
    - OpenMP
    - MPI
    - GPU Computing
4. Advanced topics
    - Performance optimization
    - Real-world applications
    - Best practices

Why Parallel Programming? (1/2)

Historical Evolution

- Moore's Law limitations
- Multi-core revolution
- Cloud computing era
- Big data requirements

Industry Applications

- Scientific simulations
- Financial modeling
- AI/Machine Learning
- Video processing

Why Parallel Programming? (2/2)

Performance Benefits

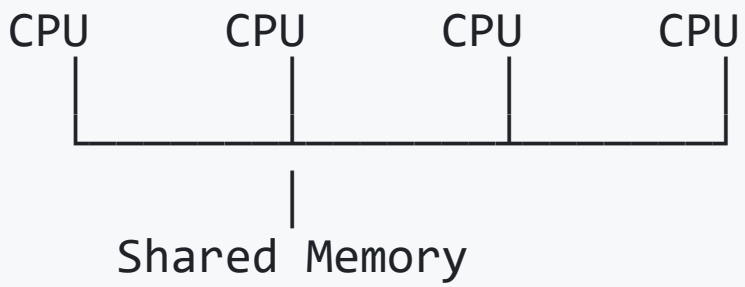
- Reduced execution time
- Better resource utilization
- Improved responsiveness
- Higher throughput

Challenges

- Synchronization overhead
- Load balancing
- Debugging complexity
- Race conditions

Parallel Computing Models (1/2)

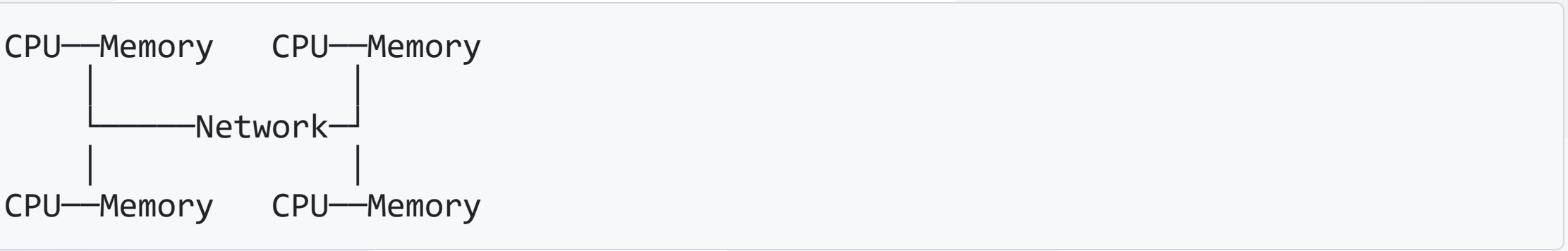
Shared Memory



- All processors access same memory
- Easy to program
- Limited scalability
- Example: OpenMP

Parallel Computing Models (2/2)

Distributed Memory



- Each processor has private memory
- Better scalability
- More complex programming
- Example: MPI

Memory Architecture Deep Dive (1/3)

Cache Hierarchy

```
// Example showing cache effects
void demonstrateCacheEffects() {
    const int SIZE = 1024 * 1024;
    int* arr = new int[SIZE];

    // Sequential access (cache-friendly)
    Timer t1;
    for(int i = 0; i < SIZE; i++) {
        arr[i] = i;
    }
    double sequential_time = t1.elapsed();

    // Random access (cache-unfriendly)
    Timer t2;
    for(int i = 0; i < SIZE; i++) {
        arr[(i * 16) % SIZE] = i;
    }
    double random_time = t2.elapsed();

    printf("Sequential/Random time ratio: %f\n",
           random_time/sequential_time);
}
```

Memory Architecture Deep Dive (2/3)

False Sharing Example

```
#include <omp.h>

// Bad example with false sharing
void falseSharing() {
    int data[4];
    #pragma omp parallel for
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 1000000; j++) {
            data[i]++; // Adjacent elements share cache line
        }
    }
}

// Better version avoiding false sharing
void avoidFalseSharing() {
    struct PaddedInt {
        int value;
        char padding[60]; // Separate cache lines
    };
    PaddedInt data[4];

    #pragma omp parallel for
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 1000000; j++) {
            data[i].value++;
        }
    }
}
```

Memory Architecture Deep Dive (3/3)

NUMA Awareness

```
// NUMA-aware allocation
void numaAwareAllocation() {
    #pragma omp parallel
    {
        // Each thread allocates its own memory
        std::vector<double> local_data(1000000);

        // Process local data
        #pragma omp for
        for(int i = 0; i < local_data.size(); i++) {
            local_data[i] = heavyComputation(i);
        }
    }
}
```

OpenMP Fundamentals (1/4)

Basic Parallel Regions

```
#include <omp.h>

void basicParallelRegion() {
    #pragma omp parallel
    {
        // This code runs in parallel
        int thread_id = omp_get_thread_num();

        #pragma omp critical
        std::cout << "Thread " << thread_id << " starting\n";

        // Do some work
        heavyComputation();

        #pragma omp critical
        std::cout << "Thread " << thread_id << " finished\n";
    }
}
```

OpenMP Fundamentals (2/4)

Work Sharing Constructs

```
void workSharing() {
    const int SIZE = 1000000;
    std::vector<double> data(SIZE);

    // Parallel for loop
    #pragma omp parallel for schedule(dynamic, 1000)
    for(int i = 0; i < SIZE; i++) {
        data[i] = heavyComputation(i);
    }

    // Parallel sections
    #pragma omp parallel sections
    {
        #pragma omp section
        { task1(); }

        #pragma omp section
        { task2(); }
    }
}
```

OpenMP Fundamentals (3/4)

Data Sharing

```
void dataSharing() {
    int shared_var = 0;
    int private_var = 0;

    #pragma omp parallel private(private_var) \
                         shared(shared_var)
    {
        private_var = omp_get_thread_num(); // Each thread has its copy

        #pragma omp critical
        shared_var += private_var; // Updates shared variable
    }
}
```

OpenMP Fundamentals (4/4)

Synchronization

```
void synchronization() {
    #pragma omp parallel
    {
        // Barrier synchronization
        #pragma omp barrier

        // Critical section
        #pragma omp critical
        {
            // Exclusive access
        }

        // Atomic operation
        #pragma omp atomic
        counter++;
    }
}
```

Practical Workshop (1/3)

Matrix Multiplication

```
void matrixMultiply(const std::vector<std::vector<double>>& A,
                    const std::vector<std::vector<double>>& B,
                    std::vector<std::vector<double>>& C) {
    int N = A.size();

    #pragma omp parallel for collapse(2)
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            double sum = 0.0;
            for(int k = 0; k < N; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
}
```

Practical Workshop (2/3)

Performance Comparison

```
void comparePerformance() {
    const int N = 1000;
    auto A = generateRandomMatrix(N);
    auto B = generateRandomMatrix(N);
    auto C1 = createEmptyMatrix(N);
    auto C2 = createEmptyMatrix(N);

    // Sequential version
    Timer t1;
    matrixMultiplySequential(A, B, C1);
    double sequential_time = t1.elapsed();

    // Parallel version
    Timer t2;
    matrixMultiply(A, B, C2);
    double parallel_time = t2.elapsed();

    printf("Speedup: %f\n", sequential_time/parallel_time);
}
```

Practical Workshop (3/3)

Exercise Tasks

1. Implement matrix multiplication
2. Measure performance with different matrix sizes
3. Try different scheduling strategies
4. Plot performance results

2. Development Environment

Required Hardware

- Multi-core processor
- 16GB RAM (recommended)
- 100GB free disk space
- Windows 10/11 (version 2004+)

Required Software

1. Visual Studio Community 2022
2. Windows Subsystem for Linux (WSL2)
3. Ubuntu distribution
4. Git for Windows

```
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

First OpenMP Program (15 minutes)

1. **Create Test File**
   - In VS Code, create new file: `test.cpp`
   - Add this code:

```
#include <iostream>
#include <omp.h>

int main() {
    // Get total available threads
    int max_threads = omp_get_max_threads();
    printf("System has %d processors available\n", max_threads);
```

3. Introduction to Parallel Programming

What is Parallel Programming? (1/2)

Parallel programming is the technique of writing programs that:

- Execute multiple tasks simultaneously
- Utilize multiple computational resources
- Improve performance through parallelization

What is Parallel Programming? (2/2)

Key Concepts:

- Task decomposition
- Data distribution
- Load balancing
- Synchronization

4. First Parallel Program

Hello World Example

```
#include <iostream>
#include <omp.h>

int main() {
    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();
        int total_threads = omp_get_num_threads();

        printf("Hello from thread %d of %d!\n",
               thread_id, total_threads);
    }
    return 0;
}
```

Compilation Steps

Visual Studio:

```
# Create new project
mkdir parallel_hello
cd parallel_hello

# Compile with OpenMP
cl /openmp hello.cpp
```

Running and Testing

Windows:

```
hello.exe
```

Linux/WSL:

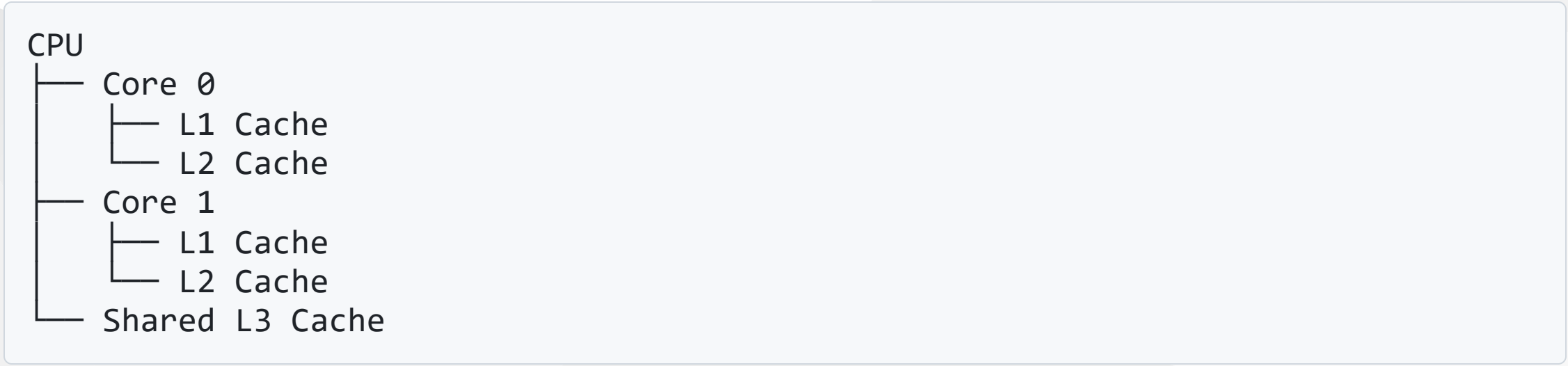
```
./hello
```

Expected Output:

```
Hello from thread 0 of 4!
Hello from thread 2 of 4!
Hello from thread 3 of 4!
Hello from thread 1 of 4!
```

5. Understanding Hardware

CPU Architecture



Memory Access Patterns

```
void measureMemoryAccess() {
    const int SIZE = 1000000;
    std::vector<int> data(SIZE);

    // Sequential access
    auto start = std::chrono::high_resolution_clock::now();
    for(int i = 0; i < SIZE; i++) {
        data[i] = i;
    }
    auto end = std::chrono::high_resolution_clock::now();

    // Random access
    start = std::chrono::high_resolution_clock::now();
    for(int i = 0; i < SIZE; i++) {
        data[(i * 16) % SIZE] = i;
    }
    end = std::chrono::high_resolution_clock::now();
}
```

6. Parallel Patterns

Data Parallelism Example

```
#include <omp.h>
#include <vector>

void vectorAdd(const std::vector<int>& a,
               const std::vector<int>& b,
               std::vector<int>& result) {
    #pragma omp parallel for
    for(int i = 0; i < a.size(); i++) {
        result[i] = a[i] + b[i];
    }
}
```

Task Parallelism Example

```
#pragma omp parallel sections
{
    #pragma omp section
    {
        // Task 1: Matrix multiplication
    }

    #pragma omp section
    {
        // Task 2: File processing
    }
}
```

7. Performance Measurement

Using the Timer Class

```
class Timer {
    std::chrono::high_resolution_clock::time_point start;
public:
    Timer() : start(std::chrono::high_resolution_clock::now()) {}

    double elapsed() {
        auto end = std::chrono::high_resolution_clock::now();
        return std::chrono::duration<double>(end - start).count();
    }
};
```

Measuring Parallel Performance

```
void measureParallelPerformance() {
    const int SIZE = 100000000;
    std::vector<double> data(SIZE);

    Timer t;
    #pragma omp parallel for
    for(int i = 0; i < SIZE; i++) {
        data[i] = std::sin(i) * std::cos(i);
    }
    std::cout << "Time: " << t.elapsed() << "s\n";
}
```

8. Homework

Assignment 1: Environment Setup

1. Screenshots of installations
2. Version information
3. Example program results
4. Issue resolution documentation

Assignment 2: Performance Analysis

1. Process & thread ID printing
2. Execution time measurements
3. Performance graphs
4. Analysis report

9. Resources

Documentation

- OpenMP API Specification
- Visual Studio Parallel Programming
- WSL Documentation

Books and Tutorials

- "Introduction to Parallel Programming"
- "Using OpenMP"
- Online courses

Next Week Preview

We will cover:

- Advanced parallel patterns
- Performance analysis
- OpenMP features
- Practical exercises

10. Advanced OpenMP Features

Nested Parallelism (1/2)

```
#include <omp.h>

void nestedParallelExample() {
    omp_set_nested(1); // Enable nested parallelism

    #pragma omp parallel num_threads(2)
    {
        int outer_id = omp_get_thread_num();

        #pragma omp parallel num_threads(2)
        {
            int inner_id = omp_get_thread_num();
            printf("Outer thread %d, Inner thread %d\n",
                   outer_id, inner_id);
        }
    }
}
```

Nested Parallelism (2/2)

Expected Output:

```
Outer thread 0, Inner thread 0
Outer thread 0, Inner thread 1
Outer thread 1, Inner thread 0
Outer thread 1, Inner thread 1
```

Benefits:

- Hierarchical parallelism
- Better resource utilization
- Complex parallel patterns

Task-Based Parallelism (1/3)

```
void taskBasedExample() {
    #pragma omp parallel
    {
        #pragma omp single
        {
            #pragma omp task
            heavyTask1();

            #pragma omp task
            heavyTask2();

            #pragma omp taskwait
            printf("All tasks completed\n");
        }
    }
}
```

Task-Based Parallelism (2/3)

Fibonacci Example

```
int parallel_fib(int n) {
    if (n < 30) return fib_sequential(n);

    int x, y;
    #pragma omp task shared(x)
    x = parallel_fib(n - 1);

    #pragma omp task shared(y)
    y = parallel_fib(n - 2);

    #pragma omp taskwait
    return x + y;
}
```

Task-Based Parallelism (3/3)

Task Priority

```
void priorityTasks() {
    #pragma omp parallel
    {
        #pragma omp single
        {
            #pragma omp task priority(0)
            lowPriorityTask();

            #pragma omp task priority(100)
            highPriorityTask();
        }
    }
}
```

Loop Optimization (1/3)

Loop Scheduling

```
void demonstrateScheduling() {
    const int SIZE = 1000000;

    // Static scheduling
    #pragma omp parallel for schedule(static)
    for(int i = 0; i < SIZE; i++)
        work_static(i);

    // Dynamic scheduling
    #pragma omp parallel for schedule(dynamic, 1000)
    for(int i = 0; i < SIZE; i++)
        work_dynamic(i);

    // Guided scheduling
    #pragma omp parallel for schedule(guided)
    for(int i = 0; i < SIZE; i++)
        work_guided(i);
}
```

Loop Optimization (2/3)

Loop Collapse

```
void matrixOperations() {
    const int N = 1000;
    double matrix[N][N];

    // Without collapse
    #pragma omp parallel for
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            matrix[i][j] = compute(i, j);

    // With collapse
    #pragma omp parallel for collapse(2)
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            matrix[i][j] = compute(i, j);
}
```

Loop Optimization (3/3)

SIMD Directives

```
void simdExample() {
    const int N = 1000000;
    float a[N], b[N], c[N];

    #pragma omp parallel for simd
    for(int i = 0; i < N; i++) {
        c[i] = a[i] * b[i];
    }
}
```

12. Common Parallel Programming Patterns

Pipeline Pattern (1/2)

```
struct Data {
    // ... data members
};

void pipelinePattern() {
    std::queue<Data> queue1, queue2;

    #pragma omp parallel sections
    {
        #pragma omp section // Stage 1
        {
            while(hasInput()) {
                Data d = readInput();
                queue1.push(d);
            }
        }

        #pragma omp section // Stage 2
        {
            while(true) {
                Data d = queue1.pop();
                process(d);
                queue2.push(d);
            }
        }

        #pragma omp section // Stage 3
        {
            while(true) {
                Data d = queue2.pop();
                writeOutput(d);
            }
        }
    }
}
```

Pipeline Pattern (2/2)

Benefits:

- Improved throughput
- Better resource utilization
- Natural for streaming data

Challenges:

- Load balancing
- Queue management
- Termination conditions

13. Debugging Parallel Programs

Common Issues (1/2)

1. Race Conditions

```
// Bad code
int counter = 0;
#pragma omp parallel for
for(int i = 0; i < N; i++)
    counter++; // Race condition!

// Fixed code
int counter = 0;
#pragma omp parallel for reduction(+:counter)
for(int i = 0; i < N; i++)
    counter++;
```

Common Issues (2/2)

2. Deadlocks

```
// Potential deadlock
#pragma omp parallel sections
{
    #pragma omp section
    {
        #pragma omp critical(A)
        {
            #pragma omp critical(B)
            { /* ... */ }
        }
    }

    #pragma omp section
    {
        #pragma omp critical(B)
        {
            #pragma omp critical(A)
            { /* ... */ }
        }
    }
}
```

14. Real-World Applications

Image Processing Example

```
void parallelImageProcessing(unsigned char* image,
                             int width, int height) {
    #pragma omp parallel for collapse(2)
    for(int y = 0; y < height; y++) {
        for(int x = 0; x < width; x++) {
            int idx = (y * width + x) * 3;

            // Apply Gaussian blur
            float sum_r = 0, sum_g = 0, sum_b = 0;
            float weight_sum = 0;

            for(int ky = -2; ky <= 2; ky++) {
                for(int kx = -2; kx <= 2; kx++) {
                    int ny = y + ky;
                    int nx = x + kx;

                    if(ny >= 0 && ny < height &&
                       nx >= 0 && nx < width) {
                        float weight = gaussian(kx, ky);
                        int nidx = (ny * width + nx) * 3;

                        sum_r += image[nidx + 0] * weight;
                        sum_g += image[nidx + 1] * weight;
                        sum_b += image[nidx + 2] * weight;
                        weight_sum += weight;
                    }
                }
            }

            // Store result
            image[idx + 0] = sum_r / weight_sum;
            image[idx + 1] = sum_g / weight_sum;
            image[idx + 2] = sum_b / weight_sum;
        }
    }
}
```

Monte Carlo Simulation

```
double parallelMonteCarlo(int iterations) {
    long inside_circle = 0;

    #pragma omp parallel reduction(+:inside_circle)
    {
        unsigned int seed = omp_get_thread_num();

        #pragma omp for
        for(int i = 0; i < iterations; i++) {
            double x = (double)rand_r(&seed) / RAND_MAX;
            double y = (double)rand_r(&seed) / RAND_MAX;

            if(x*x + y*y <= 1.0)
                inside_circle++;
        }
    }

    return 4.0 * inside_circle / iterations;
}
```

15. Advanced Workshop

Project: Parallel Sort Implementation

1. Sequential Quicksort
2. Parallel Quicksort
3. Performance Comparison
4. Visualization Tools

Workshop Tasks (1/3)

```
// Sequential Quicksort
void quicksort(int* arr, int left, int right) {
    if(left < right) {
        int pivot = partition(arr, left, right);
        quicksort(arr, left, pivot - 1);
        quicksort(arr, pivot + 1, right);
    }
}
```

Workshop Tasks (2/3)

```
// Parallel Quicksort
void parallel_quicksort(int* arr, int left, int right) {
    if(left < right) {
        if(right - left < THRESHOLD) {
            quicksort(arr, left, right);
            return;
        }

        int pivot = partition(arr, left, right);

        #pragma omp task
        parallel_quicksort(arr, left, pivot - 1);

        #pragma omp task
        parallel_quicksort(arr, pivot + 1, right);

        #pragma omp taskwait
    }
}
```

Workshop Tasks (3/3)

Performance Analysis Tools:

```
void analyzePerformance() {
    const int SIZES[] = {1000, 10000, 100000, 1000000};
    const int THREADS[] = {1, 2, 4, 8, 16};

    for(int size : SIZES) {
        for(int threads : THREADS) {
            omp_set_num_threads(threads);

            // Run and measure
            auto arr = generateRandomArray(size);
            Timer t;

            #pragma omp parallel
            {
                #pragma omp single
                parallel_quicksort(arr.data(), 0, size-1);
            }

            double time = t.elapsed();
            printf("Size: %d, Threads: %d, Time: %f\n",
                   size, threads, time);
        }
    }
}
```

Project Template

Download or clone the template project:

```
git clone https://github.com/ucoruh/cpp-openmp-template
# or create manually:
mkdir parallel-programming
cd parallel-programming
```

Create this structure:

```
parallel-programming/
├── CMakeLists.txt
├── src/
│   ├── main.cpp
│   └── include/
│       └── config.h
├── build/
│   ├── windows/
│   └── linux/
└── scripts/
    ├── build-windows.bat
```

Cross-Platform Development Environment (2/5)

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)
project(parallel-programming)

# C++17 standard
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)

# Find OpenMP
find_package(OpenMP)
if(OpenMP_CXX_FOUND)
    message(STATUS "OpenMP found")
else()
    message(FATAL_ERROR "OpenMP not found")
endif()

# Add executable
add_executable(${PROJECT_NAME}
    src/main.cpp
)

# Include directories
target_include_directories(${PROJECT_NAME}
    PRIVATE
        ${CMAKE_CURRENT_SOURCE_DIR}/src/include
)

# Link OpenMP
target_link_libraries(${PROJECT_NAME}
    PRIVATE
        OpenMP::OpenMP_CXX
)
```

```
:: Create build directory
mkdir build\windows 2>nul
cd build\windows

:: CMake configuration
cmake -G "Visual Studio 17 2022" -A x64 ..\..

:: Debug build
cmake --build . --config Debug

:: Release build
cmake --build . --config Release

cd ..\..

echo Build completed!
pause
```

build-linux.sh:

```
#!/bin/bash

# Create build directory
mkdir -p build/linux
cd build/linux
```

```
// Platform check
#if defined(_WIN32)
    #define PLATFORM_WINDOWS
#elif defined(__linux__)
    #define PLATFORM_LINUX
#else
    #error "Unsupported platform"
#endif

// OpenMP check
#ifdef _OPENMP
    #define HAVE_OPENMP
#endif
```

main.cpp:

```
#include <iostream>
#include <vector>
#include <omp.h>
#include "config.h"

int main() {
    // OpenMP version check
    #ifdef _OPENMP
        std::cout << "OpenMP Version: "
                  << _OPENMP << std::endl;
    #else
        std::cout << "OpenMP not supported" << std::endl;
        return 1;
    #endif

    // Set thread count
```

Common Issues and Solutions

1. **CMake OpenMP Issues:**
   - Windows: Reinstall Visual Studio
   - Linux: `sudo apt install libomp-dev`
2. **WSL Connection Issues:**

```
wsl --shutdown
wsl --update
```

3. **Build Errors:**
   - Delete build directory
   - Delete CMakeCache.txt
   - Rebuild project
4. **VS2022 WSL Target Missing:**
   - Run VS2022 as administrator

Additional Resources

- Visual Studio Documentation
- WSL Documentation
- CMake Tutorial
- OpenMP Documentation

For questions and help:

- GitHub Issues
- Email
- Office hours

$$End - Of - Week - 1$$