Engenharia de Sistemas da Computação

João Barbosa and Vitor Oliveira

Universidade do Minho

2021

Introduction - Syllabus - Grades

Syllabus

- C++11/C++17 POSIX Threads Programming Model && Mutual Exclusion
- Basics of Concurrency in C++ 11/17/20
- SIMD/SPMD/STMD Programming Models
- Distributed Memory Model Parallel Programming
- HPC Clusters Architecture
- HPC Program performance analysis and debugging

Grading

- Two projects
  - Basic raytracer with accelaration structure - April 6th (35%)
  - Performance Analysis - May 18th (35%)
  - Maximum of two students per group
  - No deadline extensions
  - Report in a "Paper" Format
- Exam at the end of the semester (30%)

C++11/17 Thread Programming Model

POSIX Execution Threads

- OS Dependent Execution Model (UNIX only)
- Allows multiple time overlapping control flows in shared memory (thread)
- Standard POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995)
- Defines API for:
  - Thread creation, join, etc...
  - Mutual exclusion (MUTEX)
  - Conditional Variables
  - Synchronization

C++ 11/17 Threads

- OS Independent Execution Model (Resource Acquisition Is Initialization)
  - UNIX - Pthreads
  - Windows - Windows Threads
- Allows multiple time overlapping control flows in shared memory (thread)
- Defines API for:
  - Thread creation, join, etc...
  - Mutual exclusion (MUTEX)
  - Conditional Variables
  - Synchronization
- Introduces parallelism primitives for C++ and allows compiler to “reason” about optimizations

C++ 11/17 Threads

Why not OpenMP?

- Pros:
  - Easy to implement on pragmas
  - Thread Pool matches the number of cores
  - Dynamic Scheduling
  - Implicit shared variables and explicit privates
  - Good Support in Unix systems
- Cons:
  - Not everything fits in a for loop fork and join
  - Good graph algorithms are "hard" to implement
  - Irregular algorithms are hard to map and to choose a good scheduling strategy
  - Ultimatly maps to POSIX Threads
  - *"Easy to induce a false sense of performance"*

C++ 11/17 Threads - Example

```
A = INT[N]
SUM = 0
(...)
function LocalSum(START_BLOCK,END_BLOCK,
                  A,SUM)

    for I in START_BLOCK .. END_BLOCK
        LOCALSUM = LOCALSUM + A[I]

    SUM += LOCALSUM
```

C++ 11/17 Threads - OpenMP Example

```
A = INT[N]
SUM = 0
(...)
function LocalSum(START_BLOCK,END_BLOCK,
                    A,SUM)
#pragma omp parallel for private(I)
    reduce(localsum:+)
    schedule(dynamic, CHUNK)
for I in START_BLOCK .. END_BLOCK
  LOCALSUM += A[I]
SUM = LOCALSUM
```

```
schedule(dynamic, CHUNK)
```

C++ 11/17 Threads - Example

```
A = INT[N]
SUM = 0
(...)
function LocalSum(START_BLOCK,END_BLOCK,
                  A,SUM)

    for I in START_BLOCK .. END_BLOCK
        LOCALSUM = LOCALSUM + A[I]
    Aquire(Mutex)
    SUM += LOCALSUM
    Release(Mutex)
```

C++ 11/17 Threads - Example

```
#include <thread>
#include <mutex>
#include <vector>
#define N (1u << 16)

int main(int argc, char* argv) {
    float A(N);
    float sum = 0;
    std::vector<std::threads> threads;
    (...)
}
```

C++ 11/17 Threads - Example

```
for(auto start=0u; start<N; start+=512u){
    uint end = std::min(start+512,N);
    threads.push_back(std::thread(
        [&](const float A(),
            const uint start,
            const uint end) {

        for(auto i = start; i < end; i++) {
            A(i) = float(i) / float(N);
        }

        },A,start,end));
    )
}
for(auto &thr : threads) {
    thr.join();
}
```

C++ 11/17 Threads - Example

```
std::mutex protect_sum;
for(auto start=0u; start<N; start+=512u){
    uint end = std::min(start+512,N);
    threads.push_back(std::thread(
        [&](const float A[],
            const uint start,
            const uint end) {
        float localsum;
        for(auto i = start; i < end; i++) {
            localsum+=A(i);
        }
        std::lock_guard lk(protect_sum);
        sum += localsum;
        },A,start,end));
    )
}
for(auto &thr : threads) {
    thr.join();
}
(...)
```

C++ 11/17 Threads - Example

- Access search6
- module use */home/jbarbosa/software/modulefiles*
- module load cmake gcc

Try to implement the fastest parallel sum using C++11/17 Threads (30min)

C++ 11/17 Threads - Example

Problems:

- Too many threads $2^{16}/2^7 = 2^9$
  - Creating and Destroying threads is expensive
  - Typically we want one thread per core
- Too many locks $2^{16}/2^7 = 2^9$
  - Locking is expensive

Possible solutions:

- Atomics
- Task based programming

C++ 11/17 Threads - Exercise

- Implement a prefix sum (reduction)
- The problem that we discussed using the locks

C++ 11/17 ASYNC - Example

```
std::vector<std::future<float>> tasks;
for(auto start=0u; start<N; start+=512u){
    uint end = std::min(start+512,N);
    threads.push_back(std::async(
        [&](const float A(),
            const uint start,
            const uint end) -> float {
        float localsum;
        for(auto i = start; i < end; i++) {
            localsum+=A(i);
        }
        return localsum;
        },A,start,end));
    )
}
for(auto &thr : threads) {
    sum += thr.get();
}
```

C++ 11/17 ASYNC

- Creates a thread pool that equals the number of cores
- Issues a *task* using a FIFO scheduling approach
- Can get a result from a *task* through a ***std::future***

C++ 11/17 ASYNC - Exercise

- Implement a prefix sum (reduction)
- The problem that we discussed using the futures

C++ 11/17 ASYNC - Advantages

- Allows task based parellism
- e.g. Traverse a graph to compute a single shortest path
- Not everything can be implemented using for loops (OpenMP)

Single Source Shortest Path Problem