

# AMATH 483/583

## Introduction to High Performance Computing

### Lecture 14: OpenMP

Andrew Lumsdaine  
Northwest Institute for Advanced Computing  
Pacific Northwest National Laboratory  
University of Washington  
Seattle, WA

# Overview

- Quiz
- Introduction
- OpenMP programming model
- Parallel regions
- Parallel for
- Reduction
- Race conditions

# Two Norm Function (Sequential)

```
double two_norm(const Vector& x) {  
    double sum = 0.0;  
    for (size_t i = 0; i < x.num_rows(); ++i) {  
        sum += x(i) * x(i);  
    }  
    return std::sqrt(sum);  
}
```

# Two Norm (Helper Function)

```
double two_norm_part(const PartitionedVector& x, size_t p) {
    double sum = 0.0;
    for (size_t i = x.partitions_[p]; i < x.partitions_[p+1]; ++i) {
        sum += x(i) * x(i);
    }
    return sum;
}

double two_norm_rx(const PartitionedVector& x) {
    std::vector<std::future<double>> futures_;
    for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
        futures_.push_back(std::async(std::launch::async, two_norm_part, std::cref(x), p));
    }

    double sum = 0.0;
    for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
        sum += futures_[p].get();
    }
    return std::sqrt(sum);
}
```

# Two Norm (Lambda)

```
double two_norm_l(const PartitionedVector& x) {
    std::vector<std::future<double>> futures_;
    for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
        futures_.emplace_back(std::async(std::launch::async, [&](size_t p) {
            double sum = 0.0;
            for (size_t i = x.partitions_[p]; i < x.partitions_[p+1]; ++i) {
                sum += x(i) * x(i);
            }
            return sum;
        }, p));
    }

    double sum = 0.0;
    for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
        sum += futures_[p].get();
    }
    return std::sqrt(sum);
}
```

# Two Norm (Lambda)

```
double two_norm(const Vector& x) {  
    double sum = 0.0;  
    for (size_t i = 0; i < x.num_rows(); ++i) {  
        sum += x(i) * x(i);  
    }  
    return std::sqrt(sum);  
}
```

This  
(straightforward)

Became this (not  
so straightforward)

Different parts  
of vector

Finding  
Concurrency

Partitioned vector

Algorithm  
Structure

Rewrite algorithm

Supporting  
Structures

PartitionedVector

Implementation  
Mechanisms

Fork/join

C++ tasks/futures

```
double two_norm(const PartitionedVector& x) {  
    std::vector<std::future<double>> futures_;  
    for (size_t p = 0; p < x.partitions_.size()-1; ++p) {  
        futures_.emplace_back(std::async(std::launch::async, [&](size_t p)  
            {  
                double sum = 0.0;  
                for (size_t i = x.partitions_[p]; i < x.partitions_[p+1]; ++i) {  
                    sum += x(i) * x(i);  
                }  
                return sum;  
            }, p));  
    }  
    double sum = 0.0;  
    for (size_t p = 0; p < x.partitions_.size()-1; ++p) {  
        sum += futures_[p].get();  
    }  
    return std::sqrt(sum);  
}
```

Would need  
to do for all

Twice as  
much code

Different /  
separate code

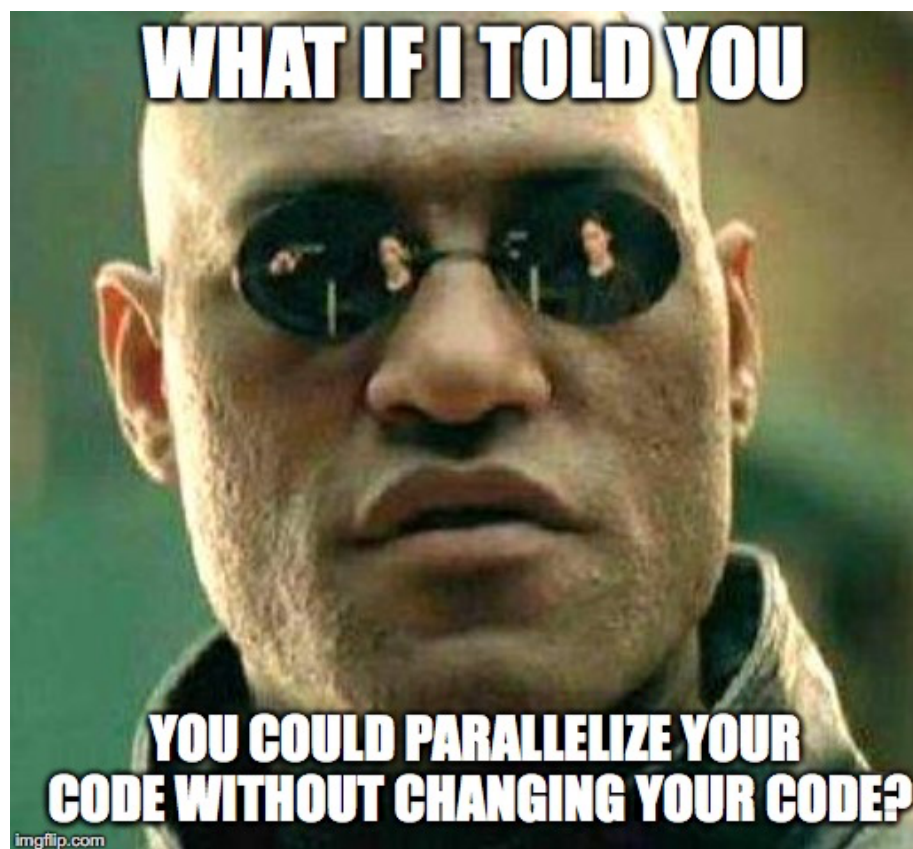
NORTHWEST INSTITUTE for ADVANCED COMPUTING

AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

Pacific Northwest  
NATIONAL LABORATORY  
Partially Operated by Battelle  
for the U.S. Department of Energy

W  
UNIVERSITY of  
WASHINGTON

# What if I told you



```
double two_norm(const Vector& x) {  
    double sum = 0.0;  
    for (size_t i = 0; i < x.num_rows(); ++i) {  
        sum += x(i) * x(i);  
    }  
    return std::sqrt(sum);  
}
```

This does not  
change

OpenMP™



# OpenMP



- **Open Multi-Processing**
- Application Program Interface (API) used to explicitly direct ***multi-threaded, shared memory*** parallelism

- Three primary API components:

- Compiler directives
- Runtime library routines
- Environment variables

Requires no  
code changes

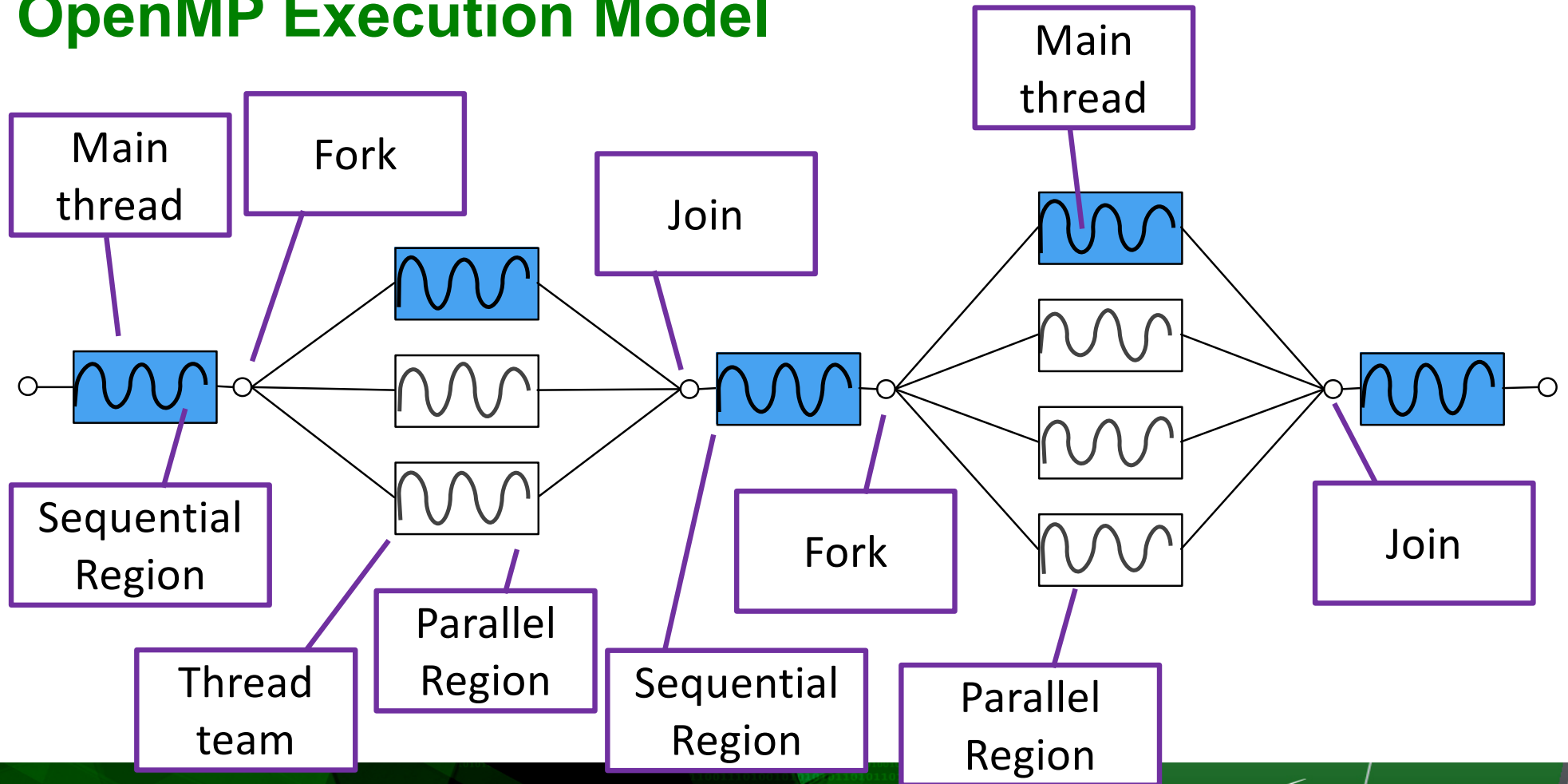
Some  
additions

Only for parallel  
version

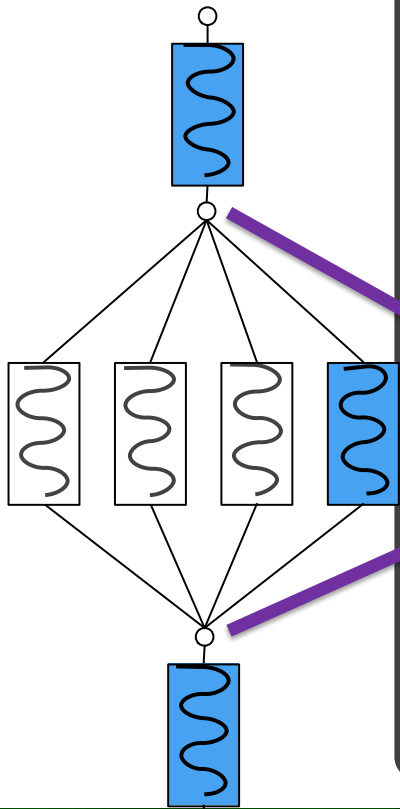
Requires no  
code changes



# OpenMP Execution Model



# Hello OpenMP v.0



```
#include <iostream>
```

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

```
int main () {
```

```
#pragma omp parallel
```

```
{
```

```
std::cout << "Hello OpenMP World!" << std::endl;
```

```
}
```

```
return 0;
```

```
}
```

# Programming with OpenMP

- How do we start a parallel region
- How do we end a parallel region
- What can we do with / in a parallel region
- Do we need to worry about race conditions and if so what do we do about them
- How do we optimize
- Do we really not need to change our code
- What else can we do with OpenMP
- Example(s)

# Querying environment

```
#include <omp.h>

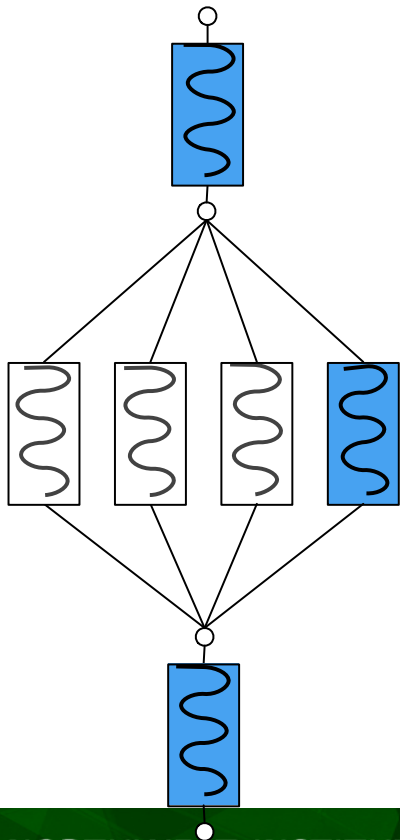
int main(int argc, char* argv[]) {

    size_t numthreads = omp_get_num_threads();
    size_t maxthreads = omp_get_max_threads();
    std::cout << "Number of threads: " << numthreads << std::endl;
    std::cout << "Max threads: " << maxthreads << std::endl;

    return 0;
}
```

# Querying the environment

```
lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[606] % ./pi_omp_8.exe
```



I

# Querying the environment

```
#include <omp.h>

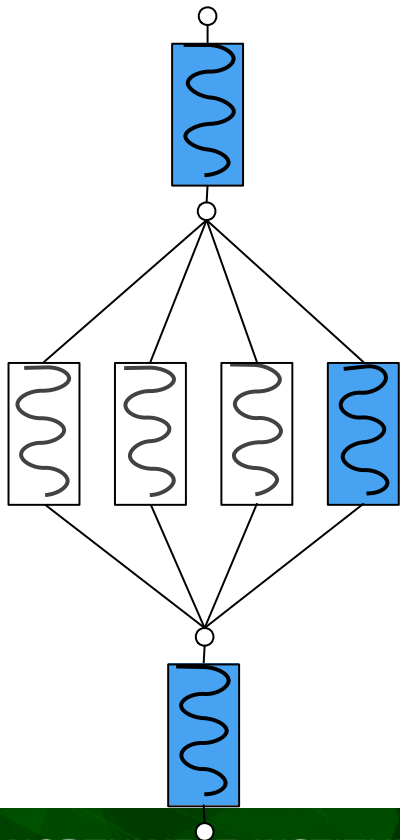
int main(int argc, char* argv[]) {

    size_t maxthreads = omp_get_max_threads();
    std::cout << "Max threads: " << maxthreads << std::endl;
    #pragma omp parallel
    {
        size_t numthreads = omp_get_num_threads();
        std::cout << "Number of threads: " + std::to_string(numthreads) + "\n";
    }

    return 0;
}
```

# Querying the environment

```
lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[607] %
```



I



# Hello OpenMP v.0

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

int main () {

#pragma omp parallel
{
    std::cout << "Hello OpenMP World!" << std::endl;
}

return 0;
}
```

lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[594] %

# Hello OpenMP v.1

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

int main () {

#pragma omp parallel
{
    std::cout << "Hello OpenMP World!\n";
}

return 0;
}
```



A terminal window with a title bar showing the user 'lums658@WE34888' and the directory '~/Teaching/amath583s19/l...'. The prompt is 'WE34888:L14/src[594] %' and the command entered is './hello\_omp\_0.exe'. The cursor is at the end of the command line.

# Hello OpenMP

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

int main () {

#pragma omp parallel
{
    std::cout << "Hello OpenMP World!" << std::endl;
}

return 0;
}
```

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

int main () {

#pragma omp parallel
{
    std::cout << "Hello OpenMP World!\n";
}

return 0;
}
```

Explain

lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[594] %

# Hello OMP

lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[594] %

```
#include <omp.h>

int main () {

#pragma omp parallel
{
    size_t tid = omp_get
    std::cout << "Hello

    if (tid == 0) {
        size_t nthreads =
        std::cout << "Numb

    }
}

return 0;
}
```

Comments?

# OMP pi 1

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h               = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```

# Output

lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[599] %

# What Happened?

Race

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h               = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```



# Before

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h               = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```

# After

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h               = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel for
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```

# Output

```
lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[600] % ./pi_omp_2.exe
```

# Before

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h              = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel for
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```

# After

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h               = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel reduction(+:pi)
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```

# Output

```
lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[602] %
```

I

# Before

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h               = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel reduction(+:pi)
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```



# After

```
int main(int argc, char* argv[]) {
    size_t intervals      = 1024 * 1024;
    if (argc >= 2) intervals = std::stol(argv[1]);
    double h               = 1.0 / (double)intervals;

    double pi = 0.0;

    #pragma omp parallel for reduction(+:pi)
    for (size_t i = 0; i < intervals; ++i) {
        pi += (h * 4.0) / (1.0 + (i * h * i * h));
    }

    std::cout << "pi is approximately " << std::setprecision(15) << pi <<
    ↪ std::endl;
    std::cout << "error is " << std::abs(PI25DT - pi) << std::endl;

    return 0;
}
```

# Output



A terminal window with a title bar containing three window control buttons and a folder icon. The title bar text is "lums658@WE34888: ~/Teaching/amath583s19/l...". The terminal content shows a green prompt "WE34888:L14/src[605] %" followed by a cursor. A mouse cursor is visible in the center of the terminal area.

```
lums658@WE34888: ~/Teaching/amath583s19/l...  
WE34888:L14/src[605] %
```

# Two Norm Function (Sequential)

```
double two_norm(const Vector& x) {  
    double sum = 0.0;  
    for (size_t i = 0; i < x.num_rows(); ++i) {  
        sum += x(i) * x(i);  
    }  
    return std::sqrt(sum);  
}
```

## Two Norm Function (Open MP)

```
double two_norm(const Vector& x) {  
    double sum = 0.0;  
    #pragma omp parallel reduction(+:sum)  
    for (size_t i = 0; i < x.num_rows(); ++i) {  
        sum += x(i) * x(i);  
    }  
    return std::sqrt(sum);  
}
```

# Thank you!

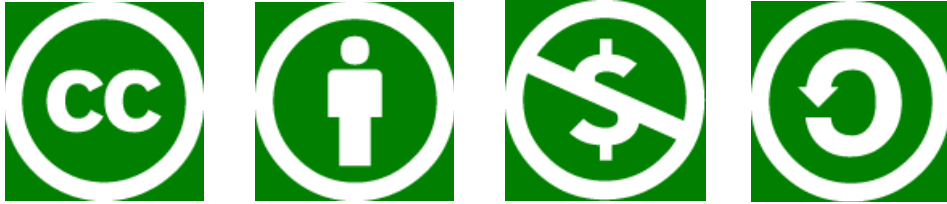
*NORTHWEST INSTITUTE for ADVANCED COMPUTING*

AMATH 483/583 High-Performance Scientific Computing Spring 2019  
University of Washington by Andrew Lumsdaine

  
**Pacific Northwest**  
NATIONAL LABORATORY  
Partially Operated by Battelle  
for the U.S. Department of Energy

  
**UNIVERSITY of  
WASHINGTON**

# Creative Commons BY-NC-SA 4.0 License



© Andrew Lumsdaine, 2017-2019

Except where otherwise noted, this work is licensed under

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

