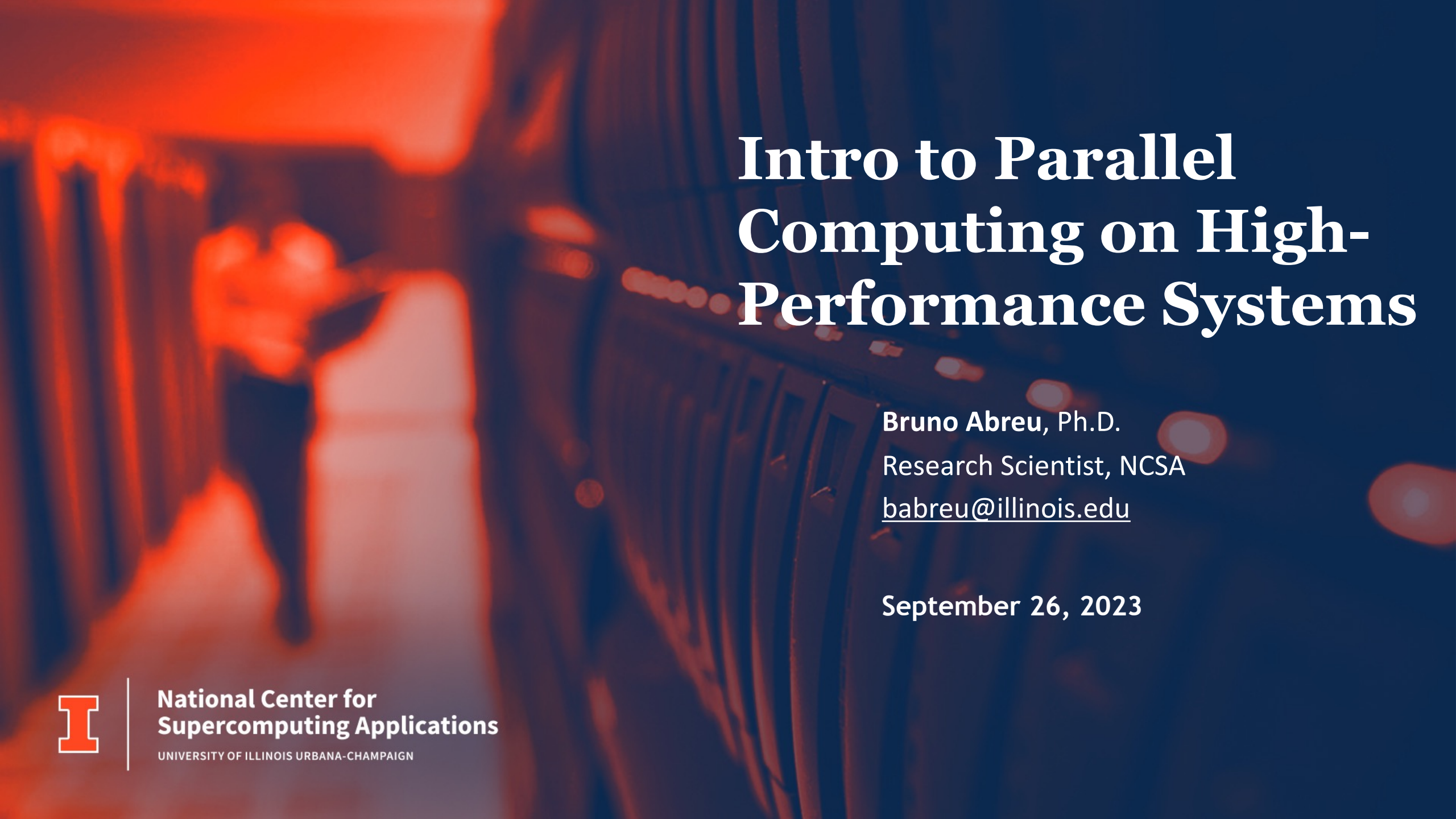


Intro to Parallel Computing on High-Performance Systems

Bruno Abreu, Ph.D.
Research Scientist, NCSA
babreu@illinois.edu

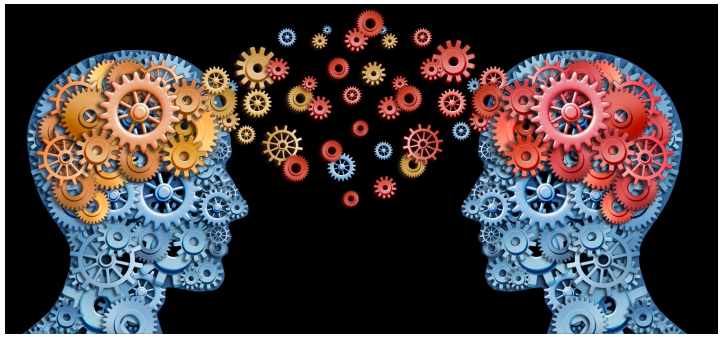
September 26, 2023

National Center for Supercomputing Applications
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Outline

Part A: **Parallel Computing Overview** (45 min)

- Why parallelism?
- Task and Data parallelism
- Parallel computers
- Threads, processes and more



Part B: **How to Parallelize a Code** (75 min)

Hello, Parallel World!
- Parallel regions
- Compilation and execution

Task Parallelism
- Parallel sections
- Measuring performance

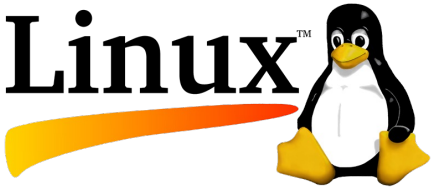
Data Parallelism
- Parallel loops
- Performance and scaling

Tools and resources

During the next 90 minutes, you will be directly or indirectly learning how to use:

- Command Line Interface (CLI)
- Linux essential commands
- Git and GitHub
- OpenMP

- Batch job processing
- Illinois Campus Cluster Program

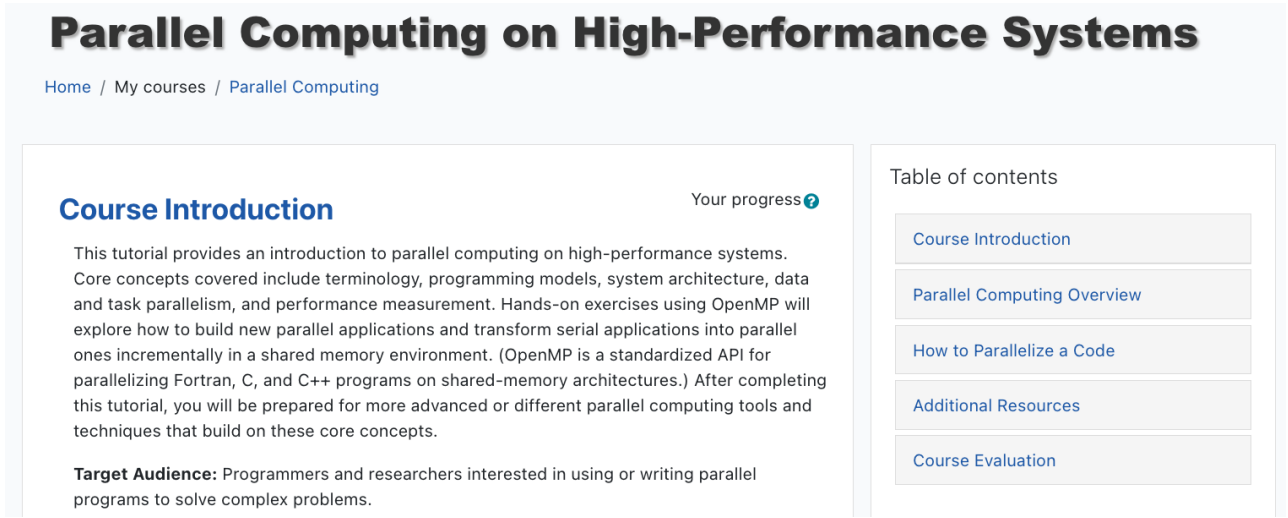


HPC Training Moodle

HPC-Moodle

www.hpc-training.org

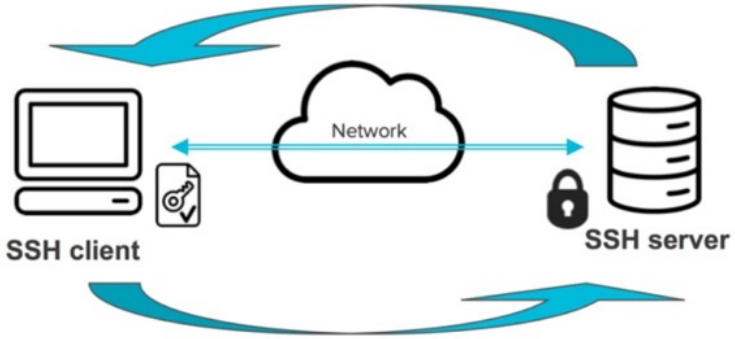
- Self-paced Tutorials
- Training Workshops
- Digital Badges



- Intro to OpenMP
- Intro to MPI
- Intro to Visualization
- Getting Started on the Illinois Campus Cluster
- Developing Effective Training Webinars

We develop live training sessions (like this one!) on top of existing self-paced courses. If you want to revisit today's contents and/or learn more about other HPC tools and frameworks, this is the place!

Environment access



ssh

Illinois Campus Cluster Program
AT THE UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

macOS®

"Terminal" on *Spotlight Search* or *Finder*

Windows

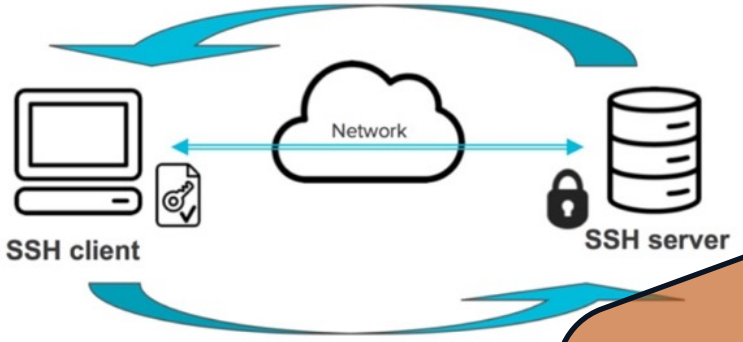
Start -> Windows System -> Command Prompt

Linux

Ctrl+Alt+T

- `ssh -l <NetID> cc-login1.campuscluster.Illinois.edu`

Environment access



POLL

ssh

Illinois Campus Cluster Program
AT THE UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

macOS®

"Terminal" on *Spotlight Search* or *Finder*

Windows

Start -> Windows System -> Command Prompt

Linux

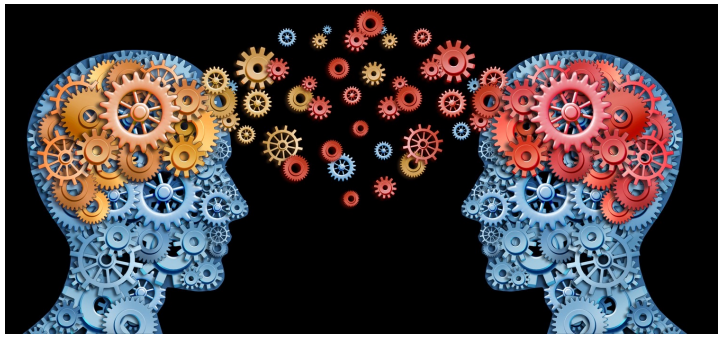
Ctrl+Alt+T

- `ssh -l <NetID> cc-login1.campuscluster.Illinois.edu`

Outline

Part A: **Parallel Computing Overview** (45 min)

- Why parallelism?
- Task and Data parallelism
- Parallel computers
- Threads, processes and more



Part B: **How to Parallelize a Code** (75 min)

Hello, Parallel World!
- Parallel regions
- Compilation and execution

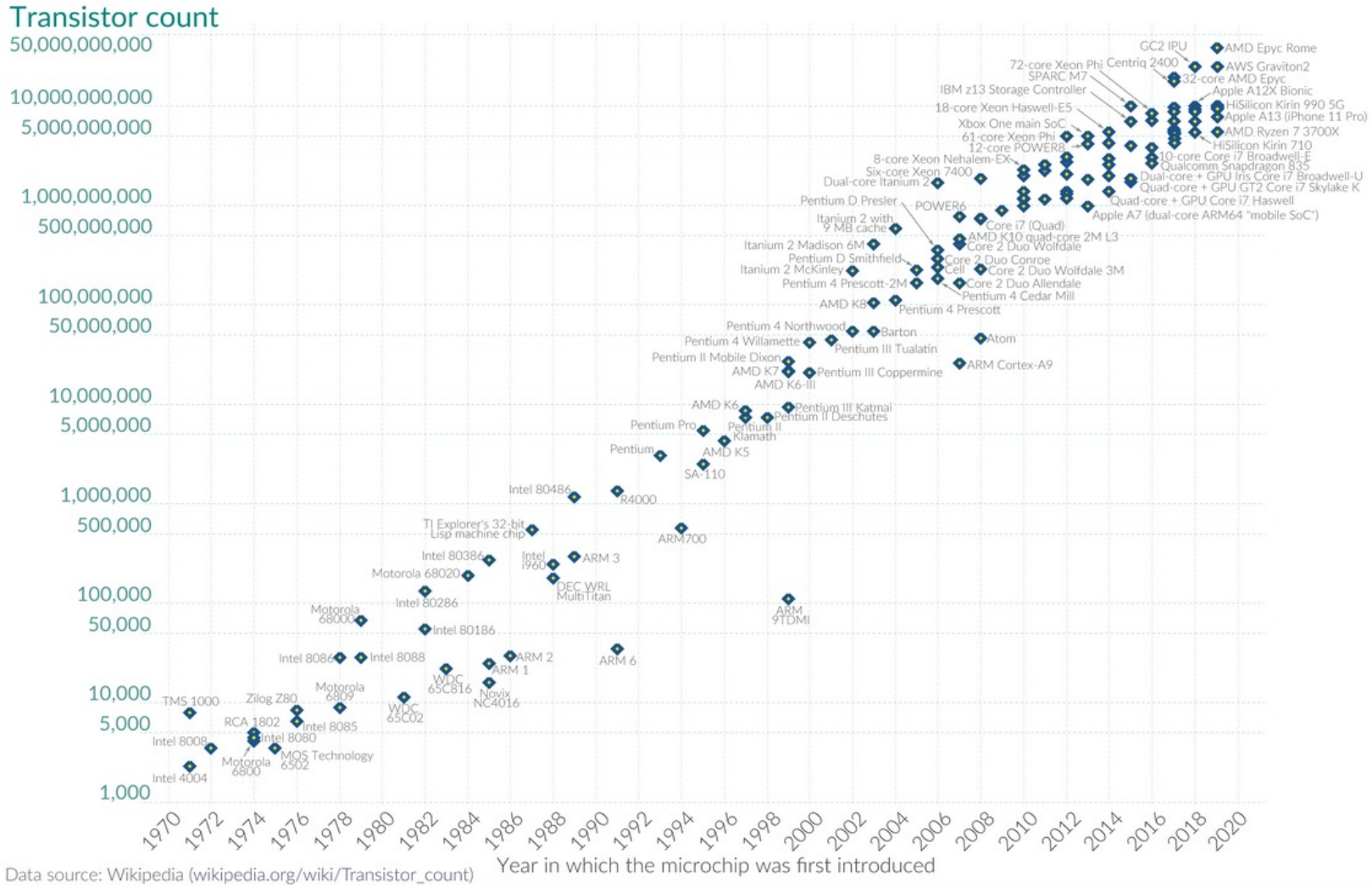
Task Parallelism
- Parallel sections
- Measuring performance

Data Parallelism
- Parallel loops
- Performance and scaling

Parallelism: why bother?

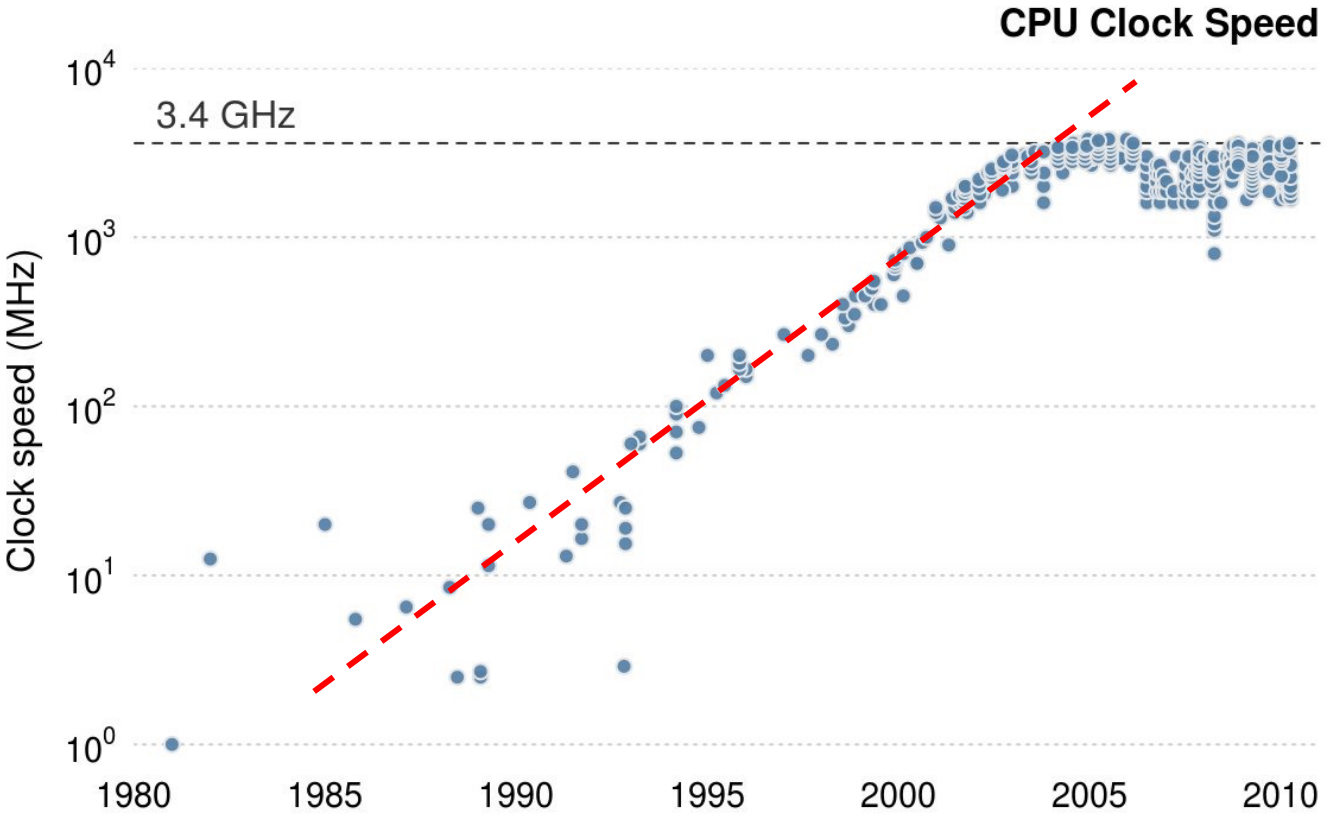
Moore's Law (1965)
"The number of transistors in a dense integrated circuit doubles every two years"

https://ourworldindata.org/grapher/transistors-per-microprocessor



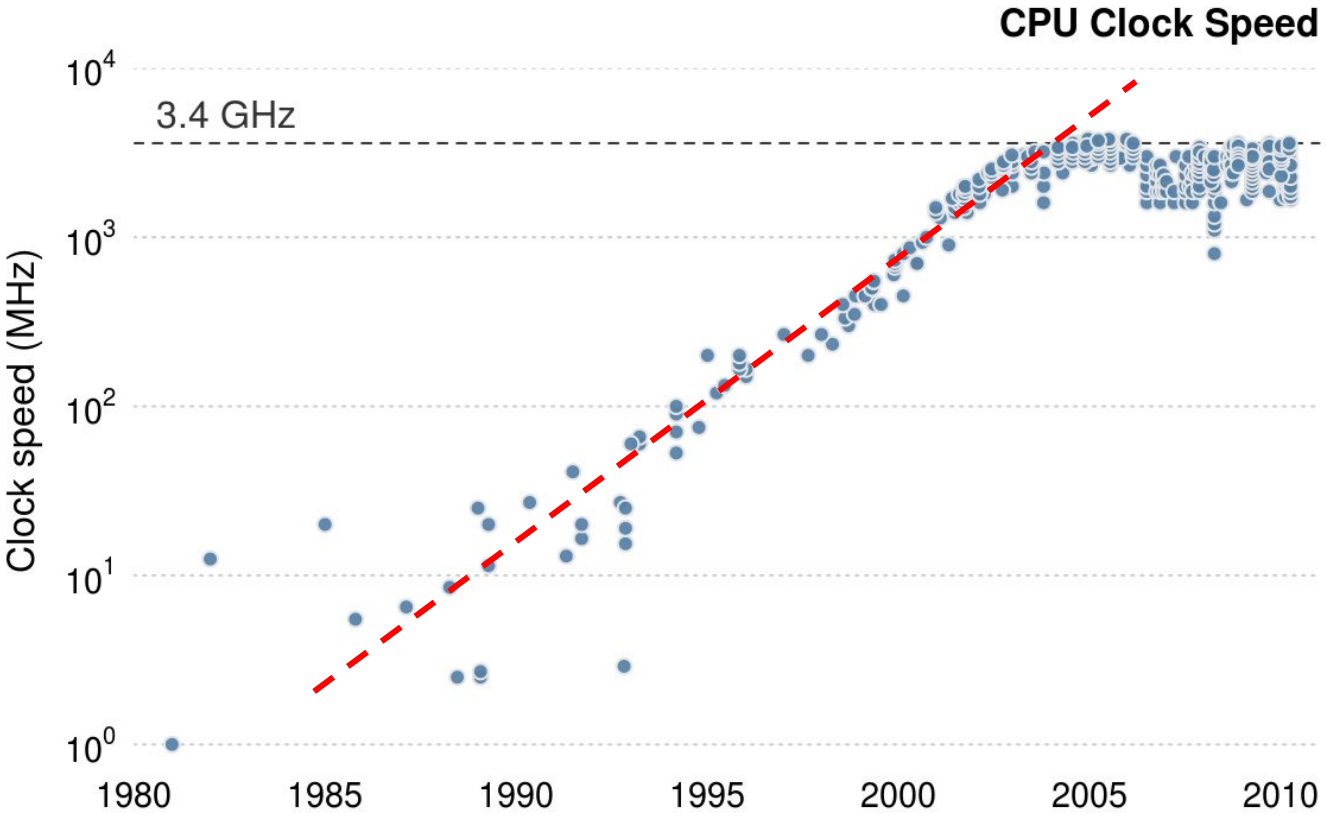
Parallelism: clock speeds

Number of cycles a CPU executes per second



Parallelism: clock speeds

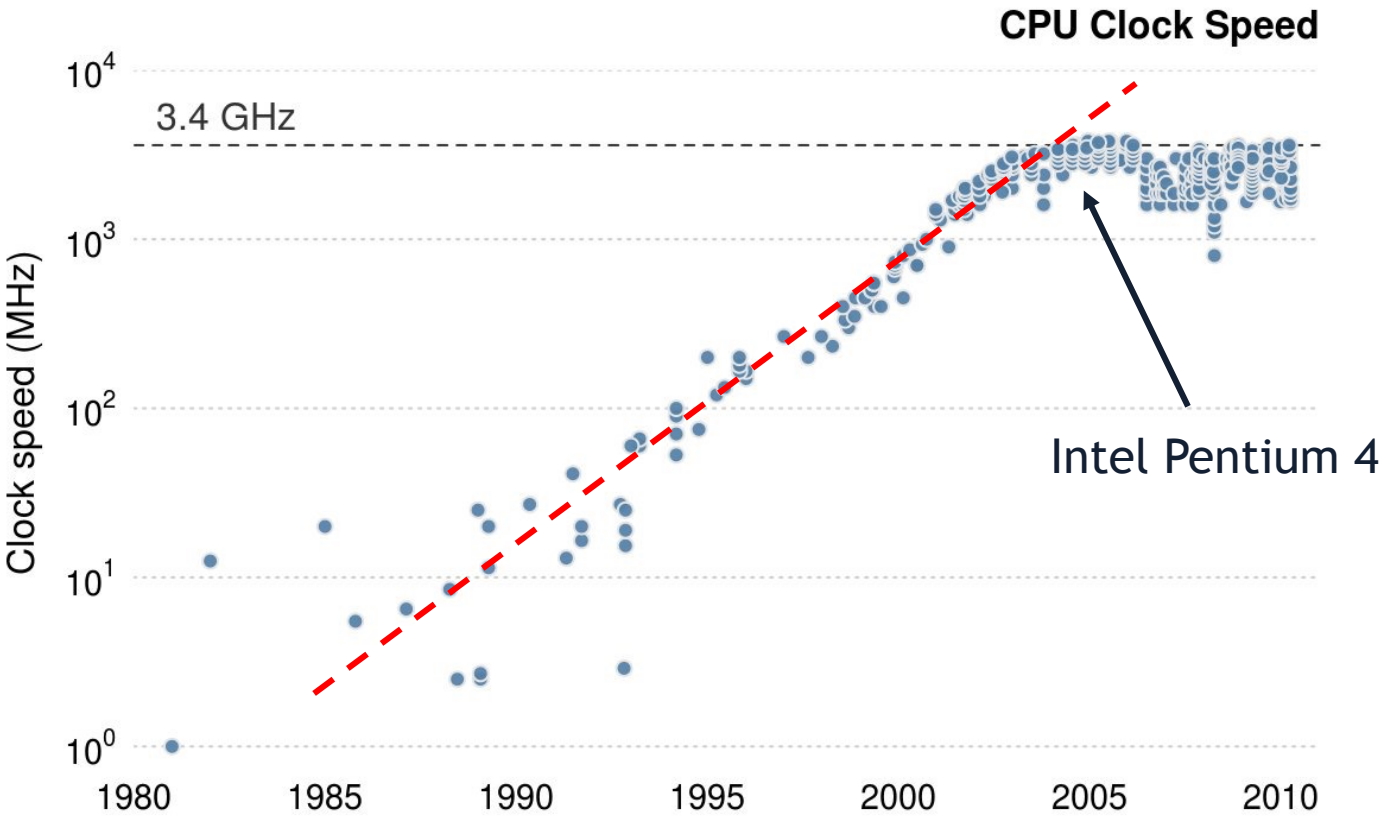
Number of cycles a CPU executes per second



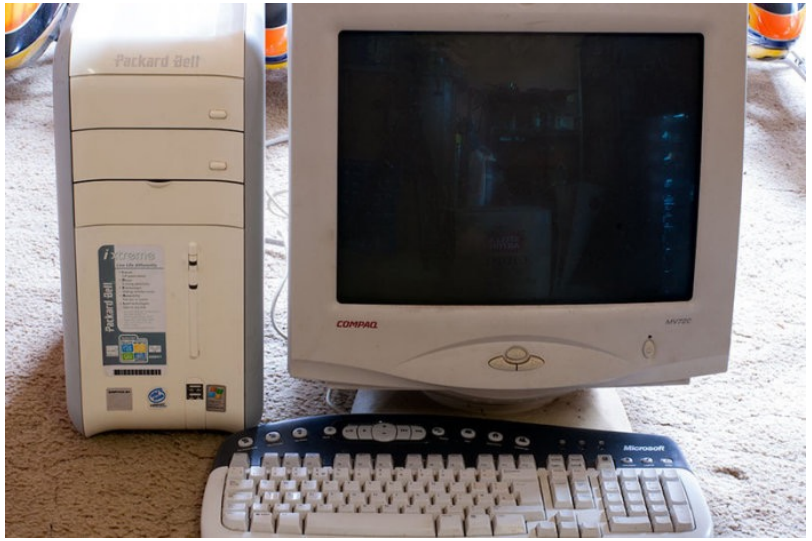
The speed at which a single CPU can process instructions has remained the same since the early 2000s!

Parallelism: clock speeds

Number of cycles a CPU executes per second



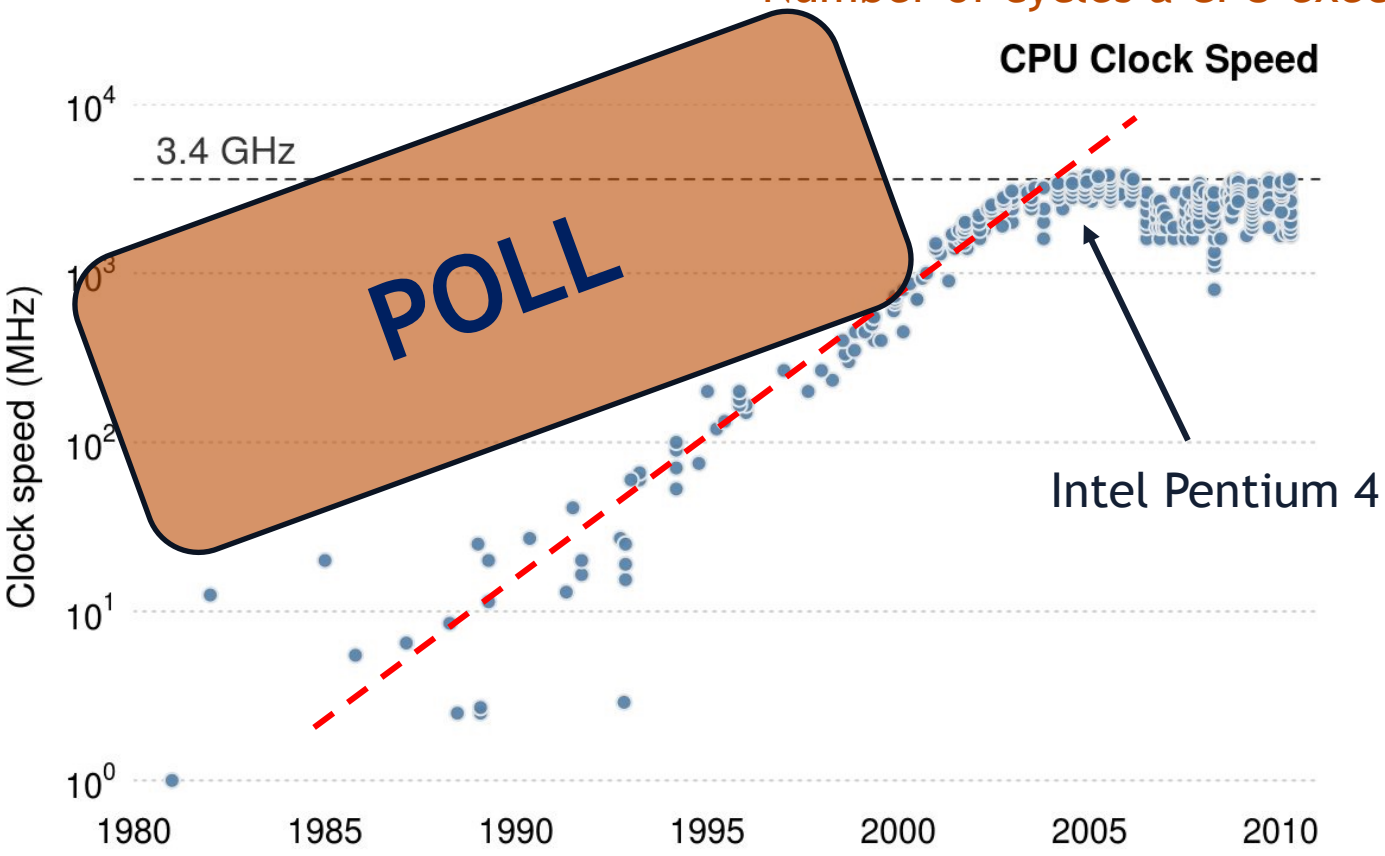
The speed at which a single CPU can process instructions has remained the same since the early 2000s!



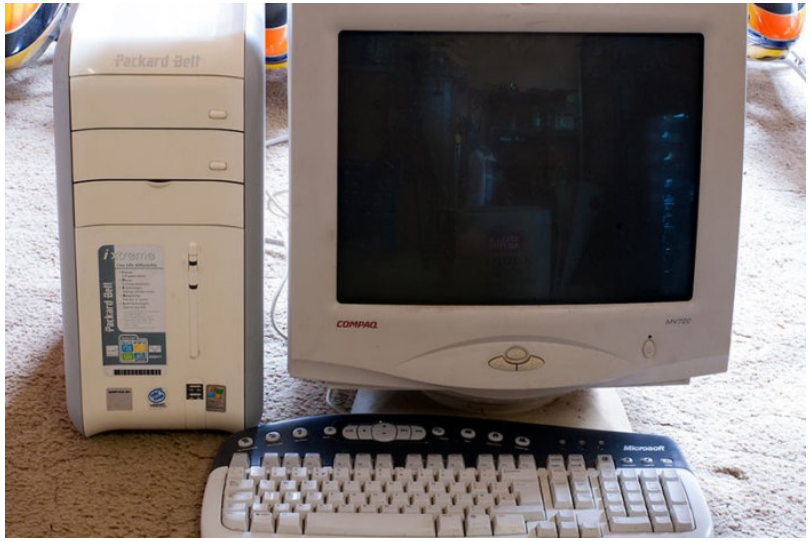
This one was sold ~ 2 years ago for $80

Parallelism: clock speeds

Number of cycles a CPU executes per second



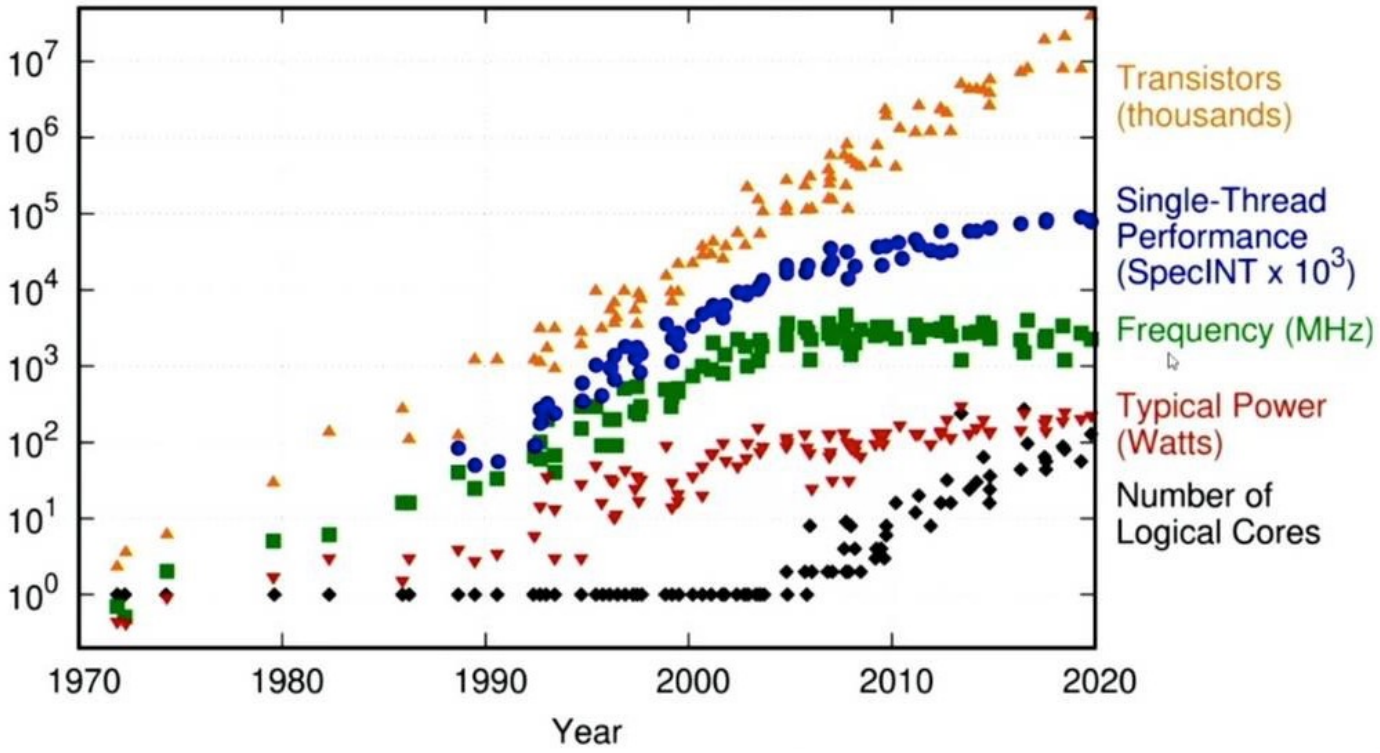
The speed at which a single CPU can process instructions has remained the same since the early 2000s!



This one was sold ~ 2 years ago for $80

Shift to multi-core architecture

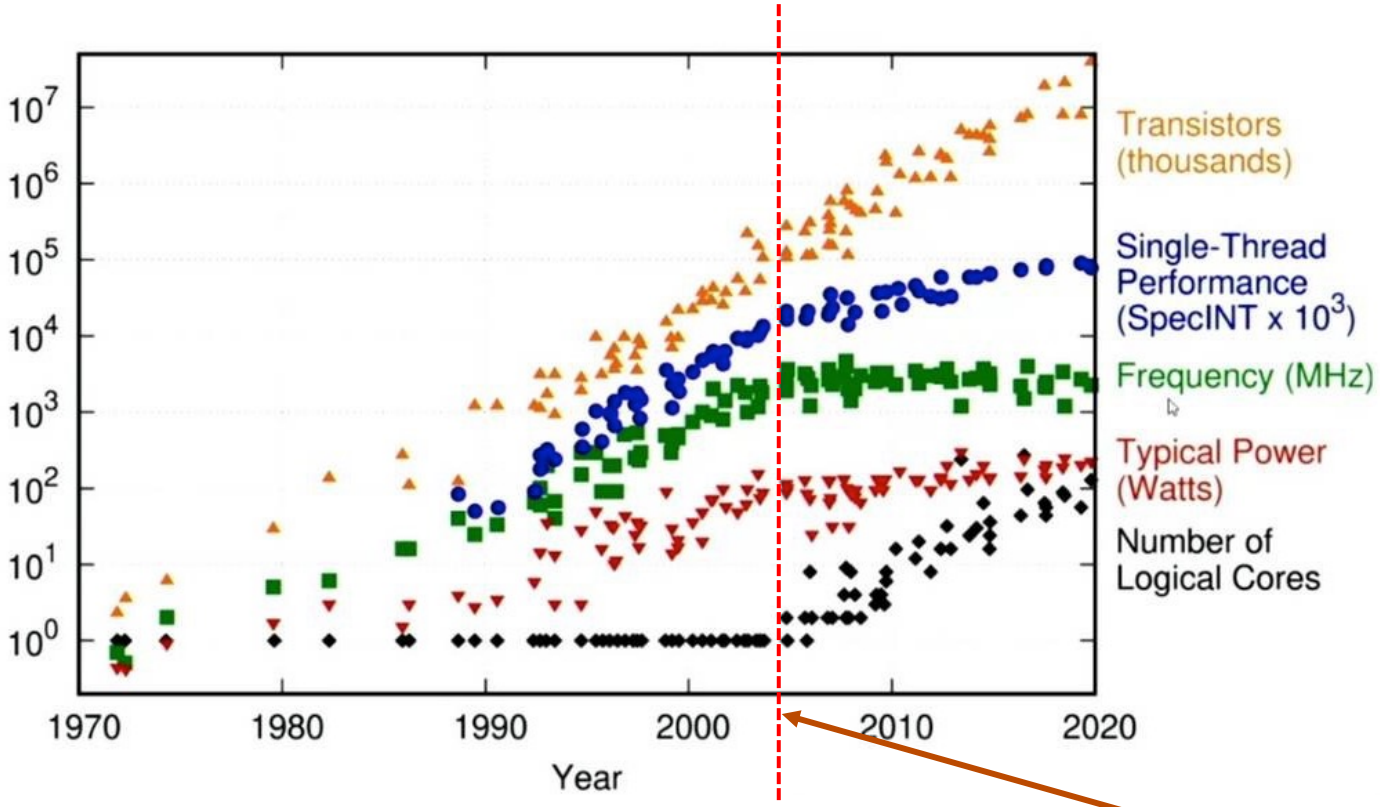
48 years of Microprocessors Trend Data



https://zenodo.org/record/3947824#.YYrKHlNMH0p

Shift to multi-core architecture

48 years of Microprocessors Trend Data

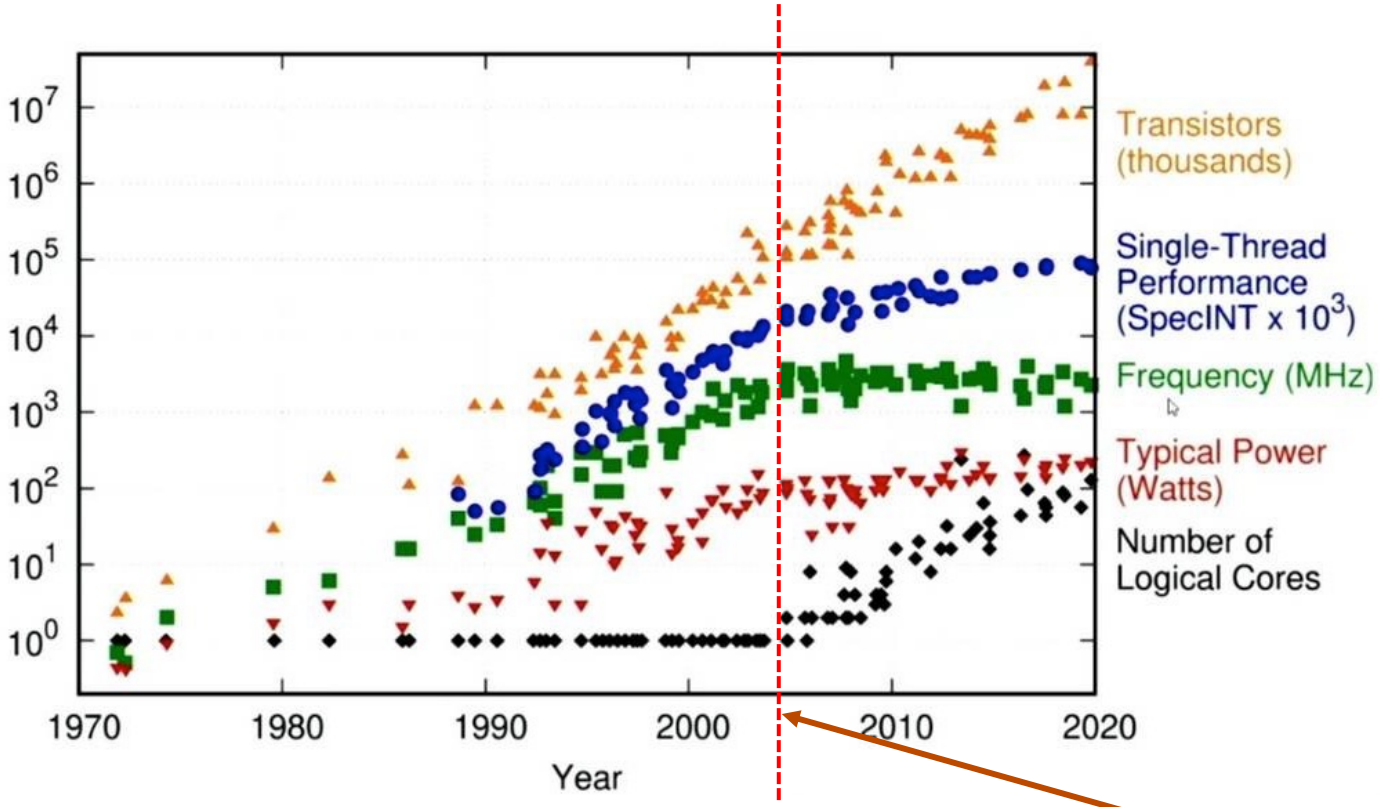


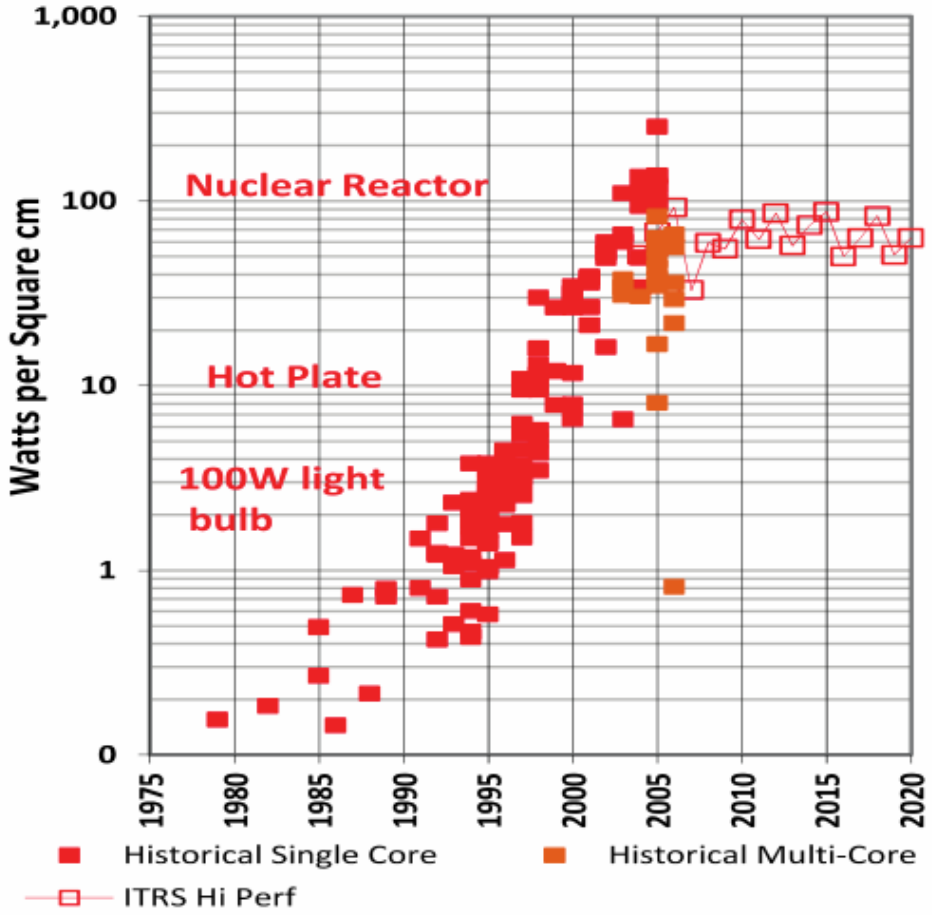
https://zenodo.org/record/3947824#.YYrKHlNMH0p

Onset of multi-core architecture

Shift to multi-core architecture

48 years of Microprocessors Trend Data





https://zenodo.org/record/3947824#.YYrKHlNMH0p

Onset of multi-core architecture

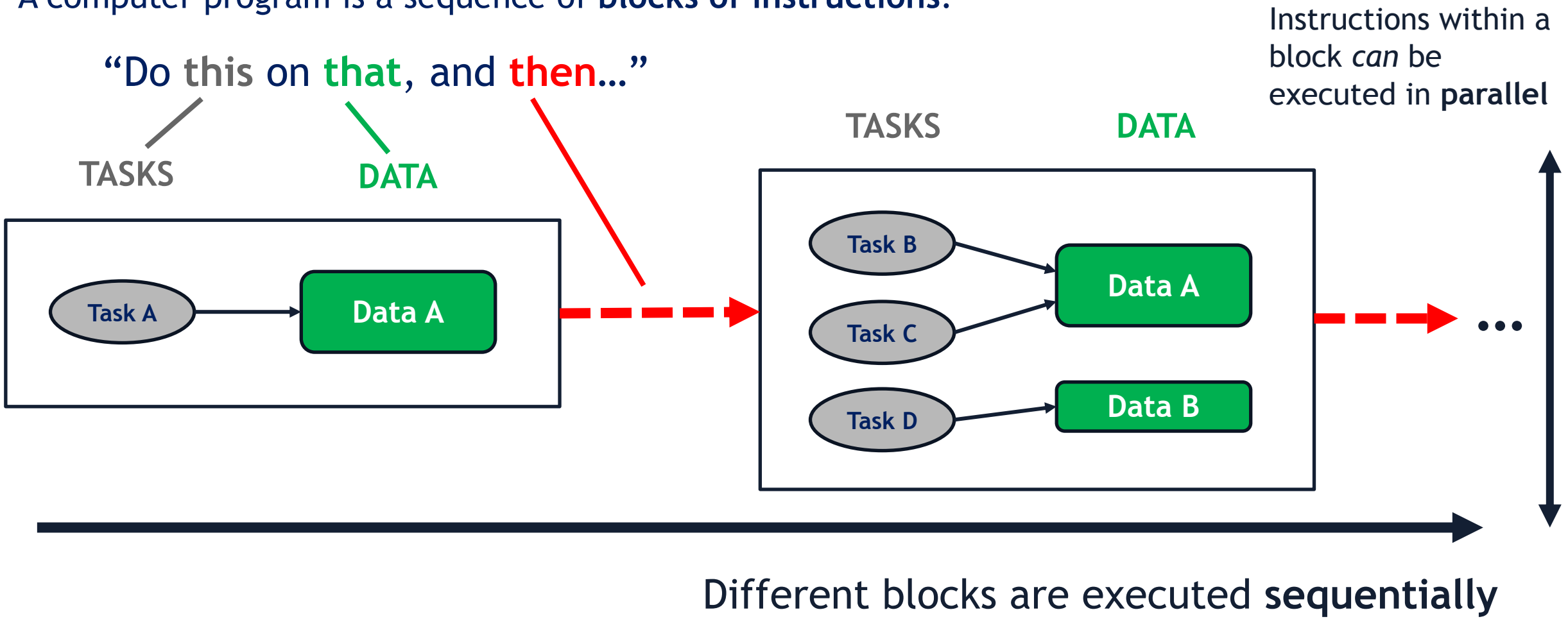
Sequential vs. Parallel Processes

Parallel

Sequential

Instruction blocks

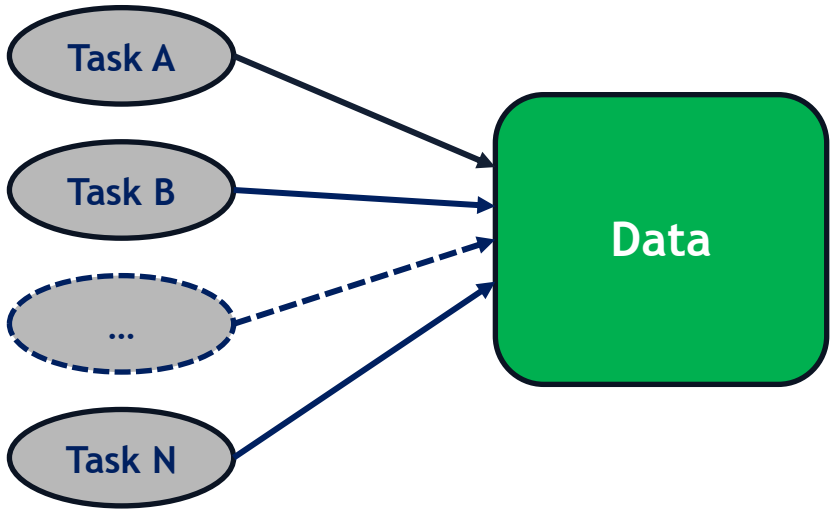
A computer program is a sequence of **blocks of instructions:**



Data & Task Parallelism

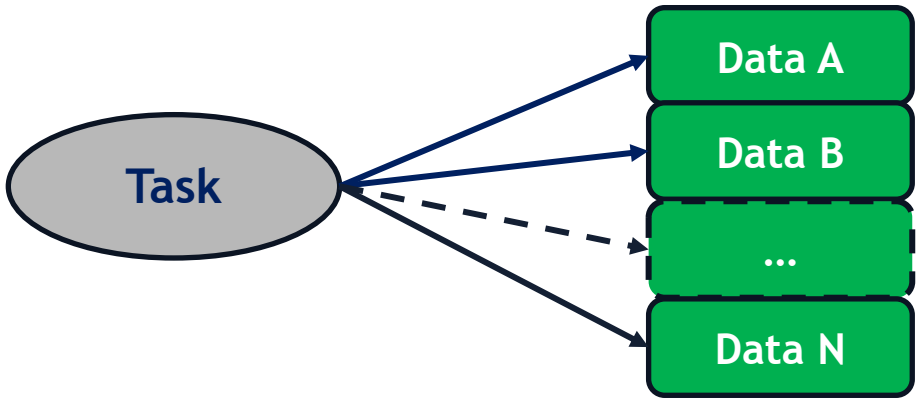
- An instruction is a pair (TASK, DATA)
- There are two “pure” ways of executing instructions in parallel:

Task Parallelism



Execute different tasks over the same data

Data Parallelism



Execute the same task over different chunks of data

Terminology

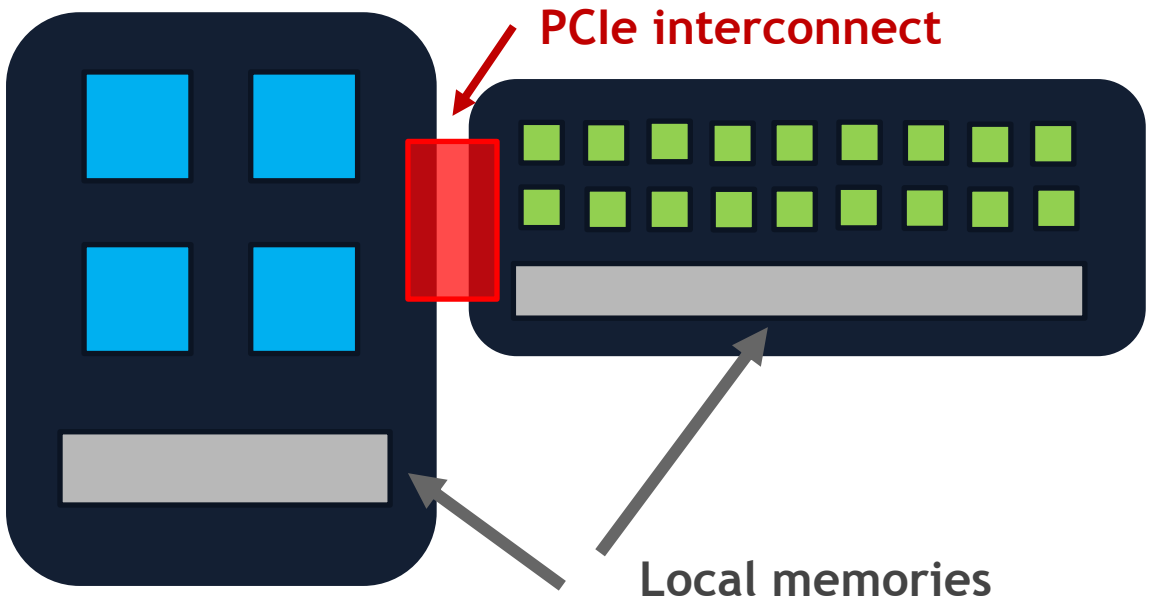
Jargon	Description
Peak performance	Maximum speed at which the computer can operate
Clock rate (GHz)	Unit of measure of a processor's speed, reflecting the natural clock rate at which it can operate
Computer cycle	Shortest time in which a unit of work can be performed
Instructions per second	How quickly the computer can issue instructions (e.g. memory reads/writes, logical operations, floating point operations, integer operations, branch instructions)
FLOPS	Floating point operations per second (e.g. subtract, add, divide, multiply)
Speedup	Measures the benefit of parallelism, showing how a program scales as more processors are employed
Benchmark	Used to rate and compare performance of parallel computers (see Top500 Supercomputers)

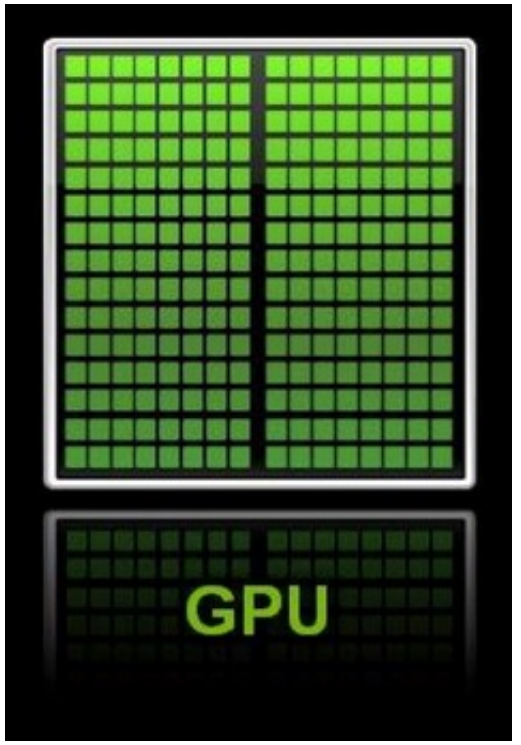
Processing Units

Moderate number of powerful processors

Large number of moderate processors

The usual setup these days…

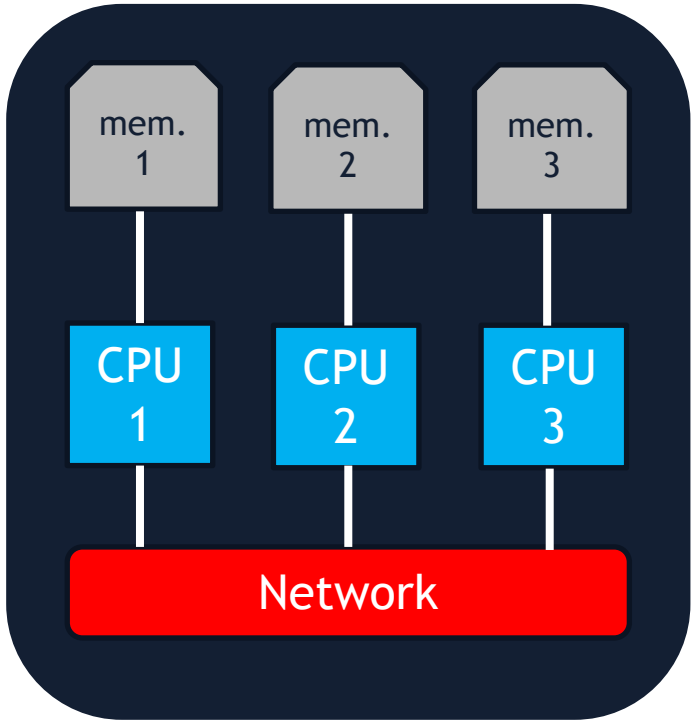




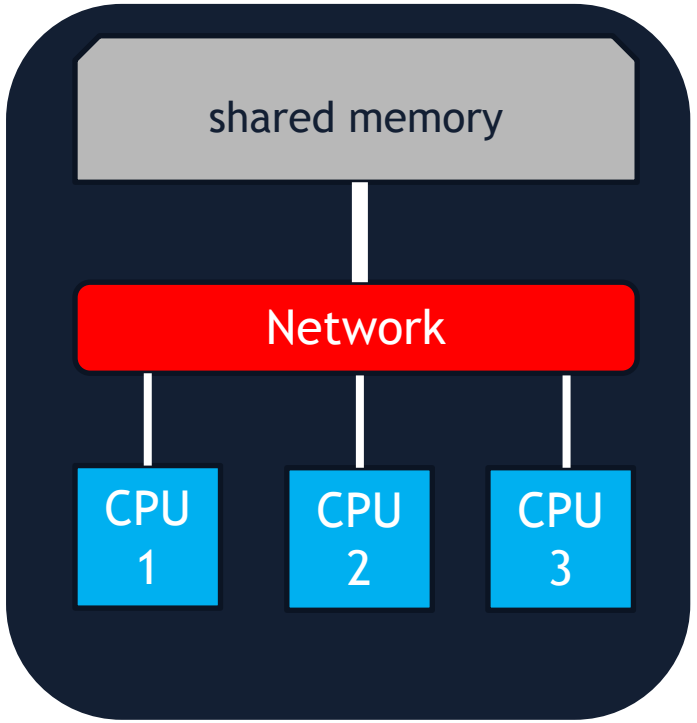
https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

Memory

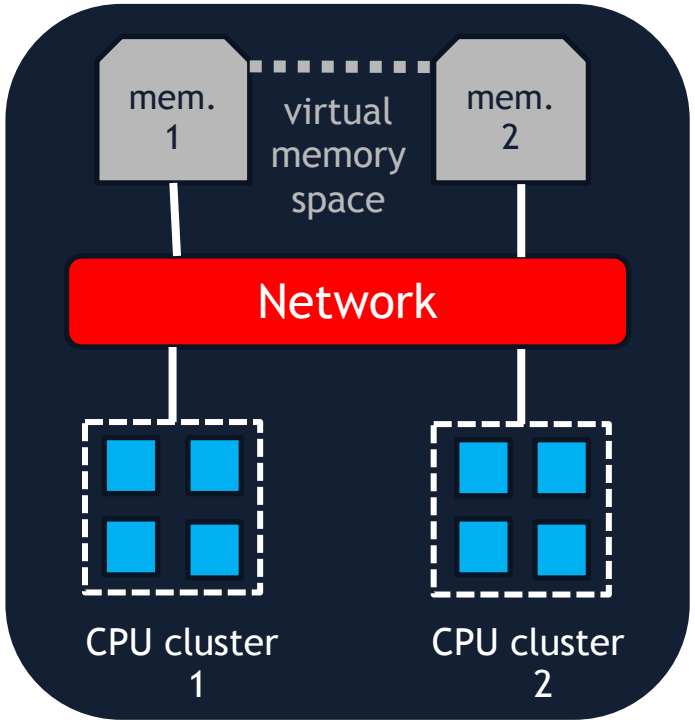
Distributed Memory



Shared Memory



Distributed Shared Memory



Flow of control

Flynn's Taxonomy

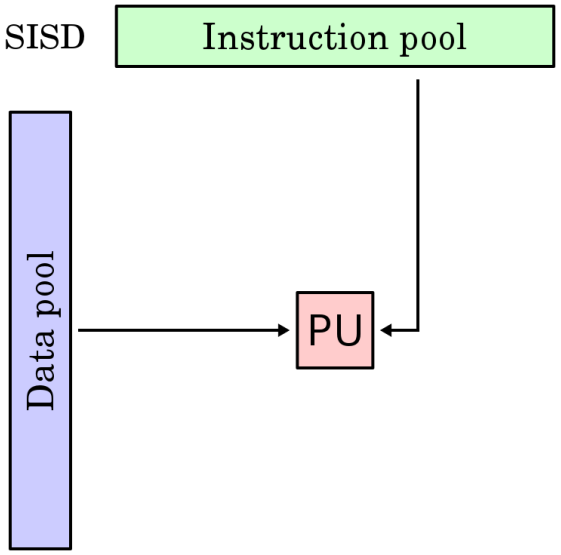
Describes how streams of instructions interact with streams of data

Instruction Streams	Data Streams: Single	Data Streams: Multiple
Single	SISD	SIMD
Multiple	MISD	MIMD

Flow of control: SISD

Flynn's Taxonomy

Describes how streams of instructions interact with streams of data



Instruction Streams \ Data Streams	Single	Multiple
Single	SISD	SIMD
Multiple	MISD	MIMD

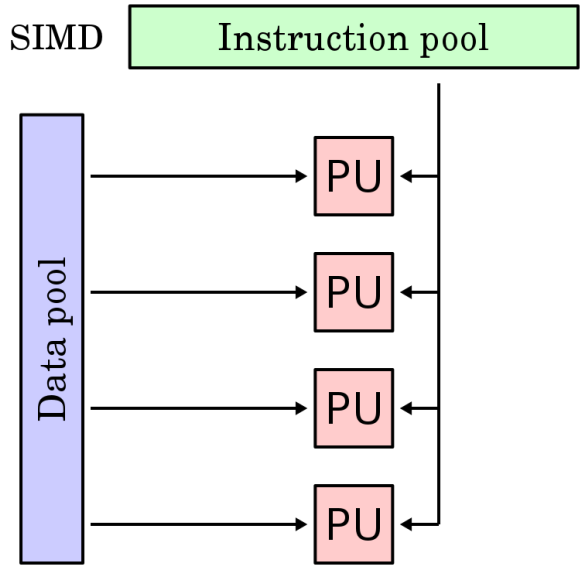
SISD is naturally not parallelizable…

- Personal computers until ~2010

Flow of control: SIMD

Flynn's Taxonomy

Describes how streams of instructions interact with streams of data



Instruction Streams	Data Streams: Single	Data Streams: Multiple
Single	SISD	SIMD
Multiple	MISD	MIMD

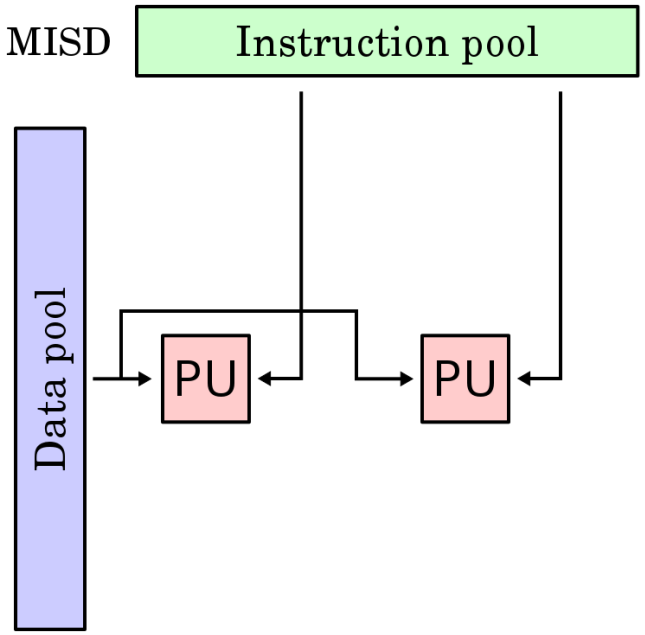
SIMD is parallelizable: it can **scale**

- Array (vector) processors: AVX (e.g. Intel Knights Landing, AMD Bulldozer)

Flow of control: MISD

Flynn's Taxonomy

Describes how streams of instructions interact with streams of data



Instruction Streams

Data Streams

Instruction Streams \ Data Streams	Single	Multiple
Single	SISD	SIMD
Multiple	MISD	MIMD

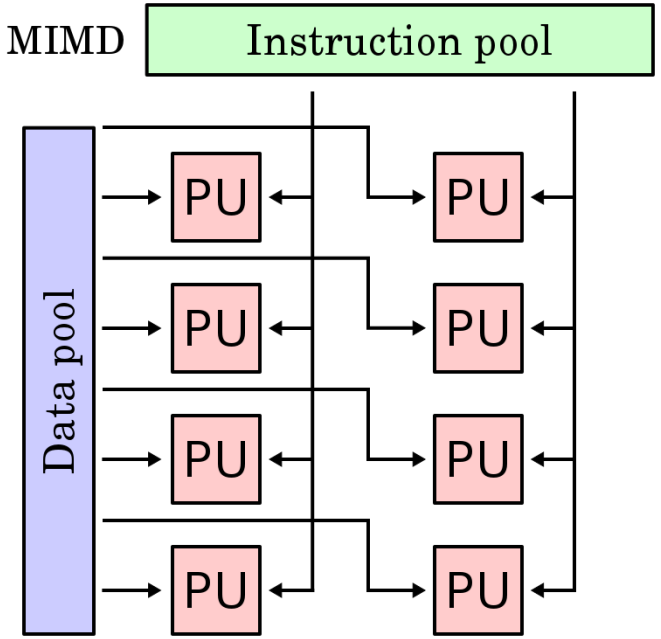
MISD is parallelizable: it can **scale**, but it is not very common

- Generally used for fault tolerance (e.g. NASA Space Shuttle flight control computer)

Flow of control: MIMD

Flynn's Taxonomy

Describes how streams of instructions interact with streams of data



Instruction Streams	Data Streams: Single	Data Streams: Multiple
Single	SISD	SIMD
Multiple	MISD	MIMD

MIMD is highly parallelizable: it can **scale** either way

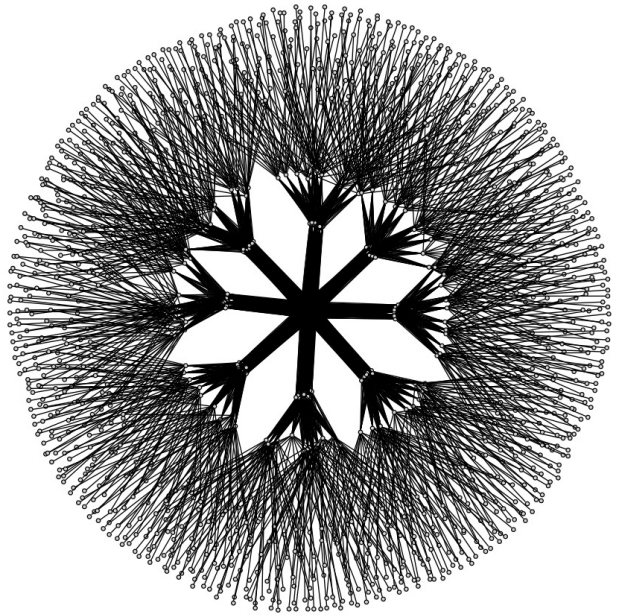
- Every supercomputer on Top500 since 2010

https://en.wikipedia.org/wiki/Flynn%27s_taxonomy

Interconnection networks

http://unixer.de/research/topologies/

Platform, wires and cables that define how multiple processors are connected to each other and to the memory units



Atlas @ Lawrence Livermore National Laboratory
1142 nodes
192 24-port InfiniBand SDR
3-staged folded Clos topology

Network Characteristics

Diameter: maximum distance that data must travel for two processors to communicate

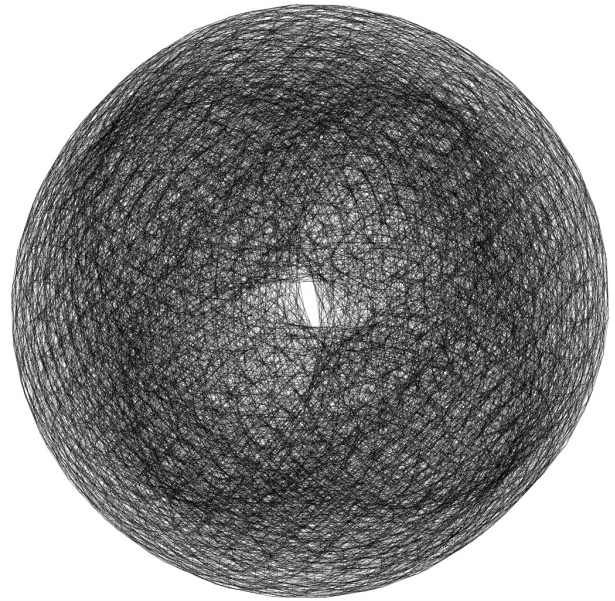
Bandwidth: amount of data that can be sent through the network within a certain time

Latency: delay on a network while a data packet is being stored and forwarded

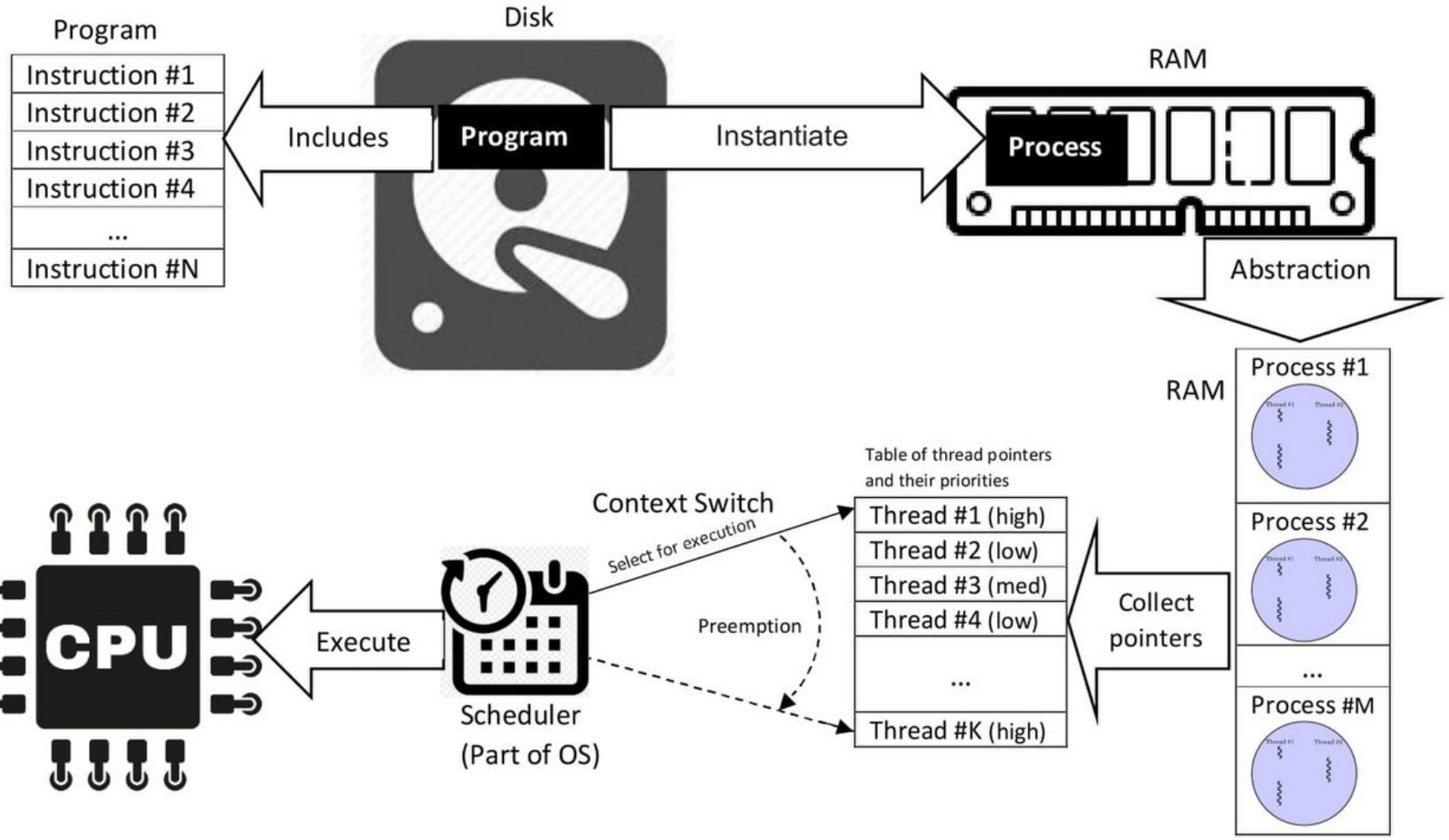
Degree: number of communicating wires coming out of each processor

Topology: configuration of the network that determines how processors are connected

Jaguar Cray XT-5 partition @ Oak Ridge National Lab
18851 nodes
SeaStar 2 25x32x24 3D-Torus

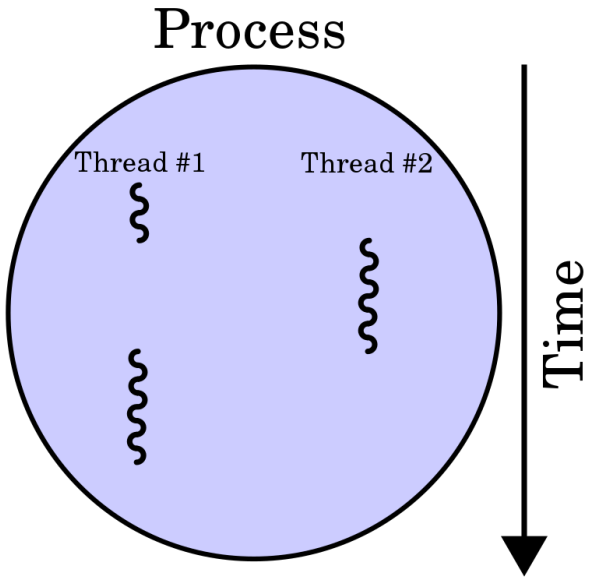


Processes, threads, and more

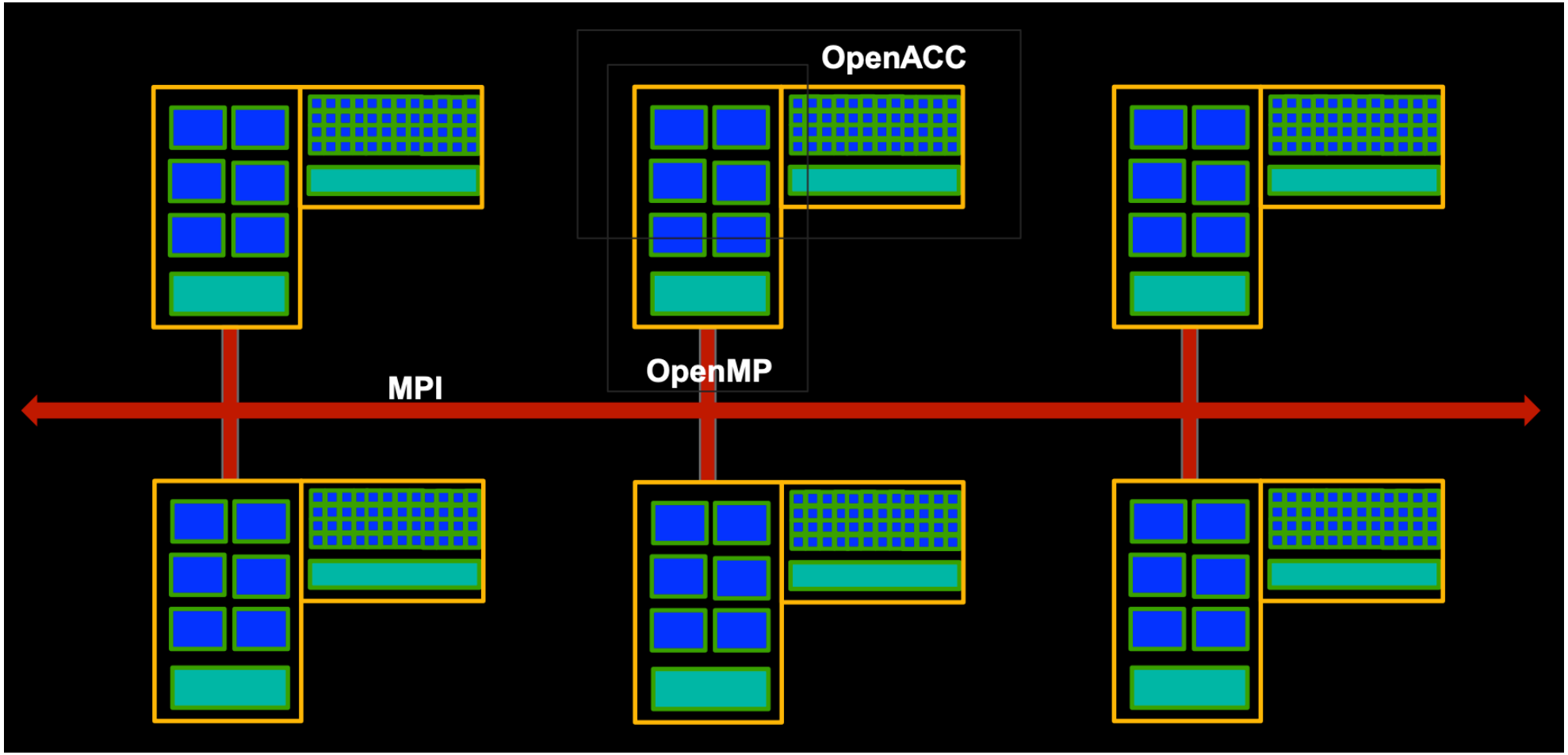


Thread: smallest sequence of instructions that the OS scheduler can manage

Process: composed of threads. Memory space allocated when program is loaded from disk.



Frameworks

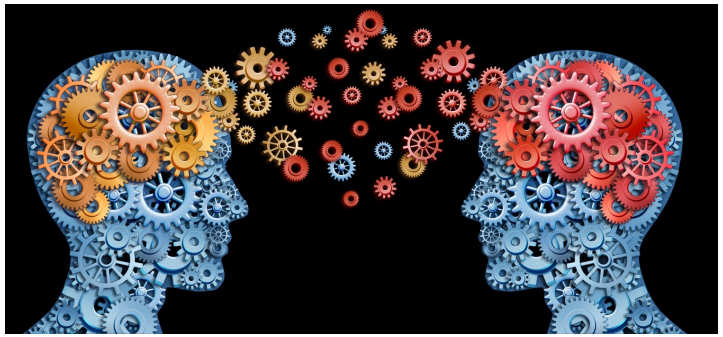


Courtesy of John Urbanic @ PSC

Outline

Part A: **Parallel Computing Overview** (45 min)

- Why parallelism?
- Task and Data parallelism
- Parallel computers
- Threads, processes and more



Part B: **How to Parallelize a Code** (75 min)

Hello, Parallel World!
- Parallel regions
- Compilation and execution

Task Parallelism
- Parallel sections
- Measuring performance

Data Parallelism
- Parallel loops
- Performance and scaling

Cloning the repository

The exercises for today
can be found here:

https://github.com/babreu-ncsa/IntroToPC/tree/uiuc-icc

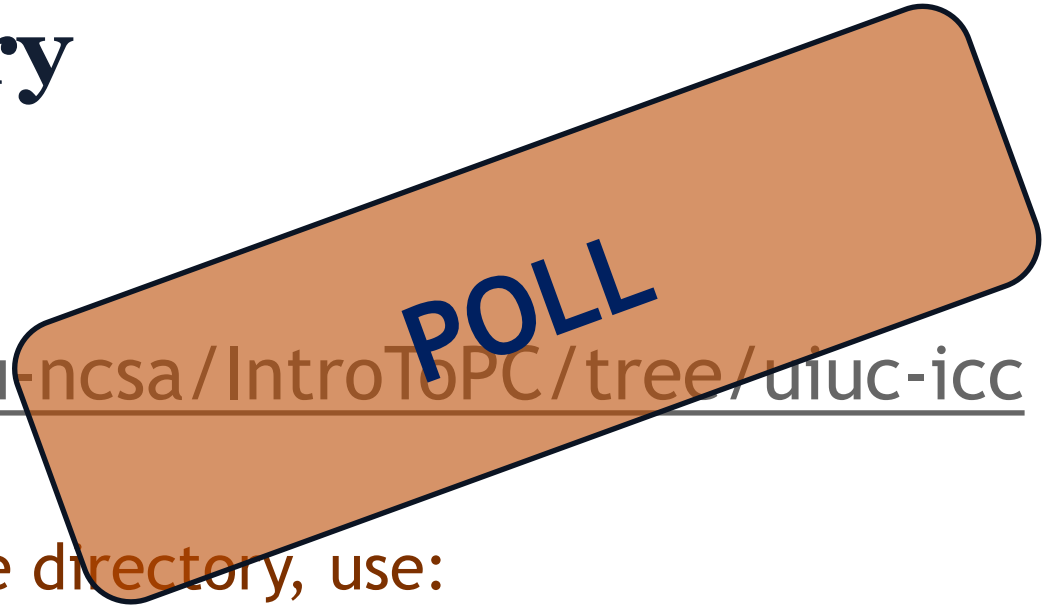
To copy these files to your ICC home directory, use:

- `module load git`
- `git clone --branch uiuc-icc --single-branch https://github.com/babreu-ncsa/IntroToPC.git`

Cloning the repository

The exercises for today
can be found here:

https://github.com/babreu-ncsa/IntroToPC/tree/uiuc-icc



To copy these files to your ICC home directory, use:

```
git clone --branch uiuc-icc --single-branch
https://github.com/babreu-ncsa/IntroToPC.git
```

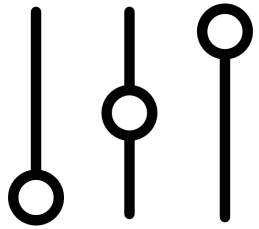
Why OpenMP?

OpenMP

"OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran."

https://www.openmp.org/

- Compiler directives
- Library routines
- Environment variables



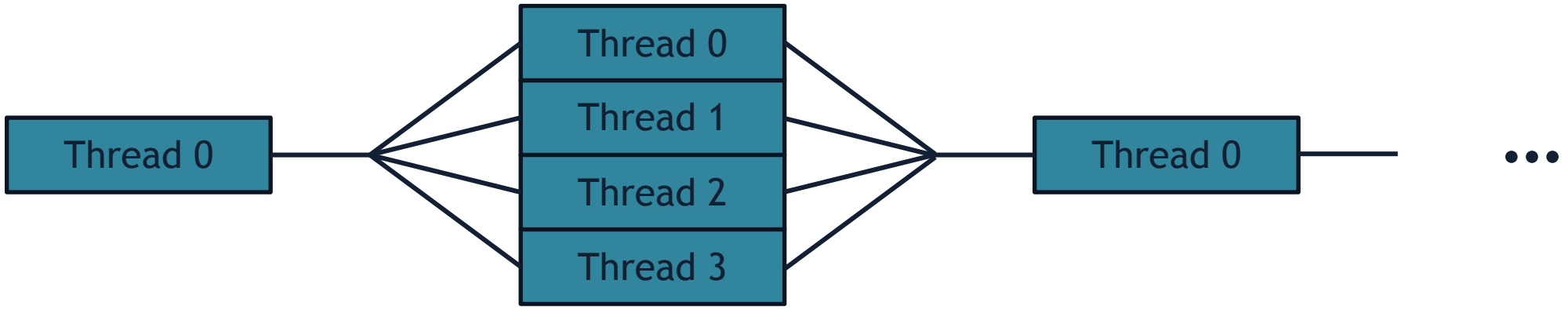
Application runtime behavior

- Simple, robust, mature framework
- Incorporated into every major compiler
- Incremental parallelism
- Does not require message passing
- Allows for reasonable speedups with very little work

- Not fully scalable by itself
- Does not require redesigning or creating new applications within the parallel computing paradigm
- Performance usually limited by sequential blocks

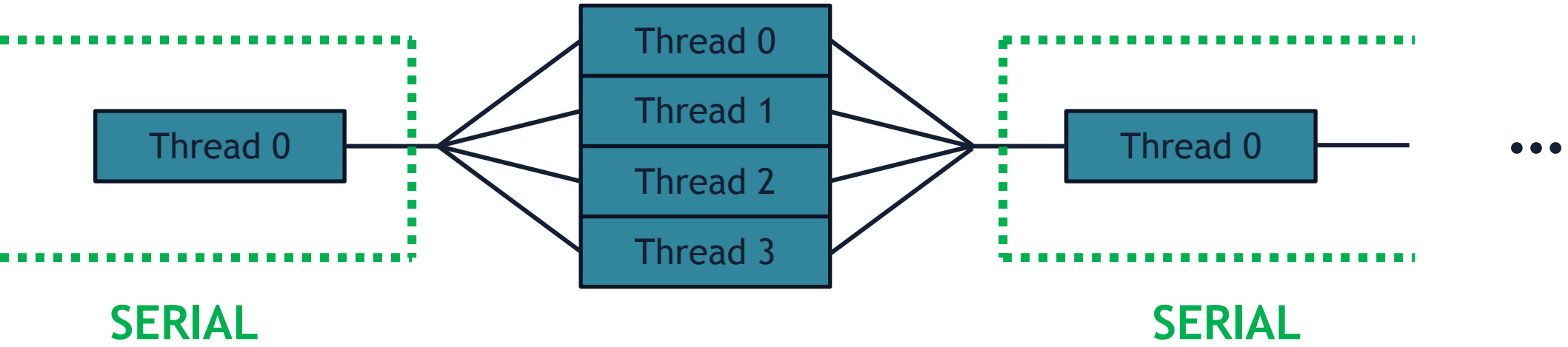
Parallel regions

These are blocks of code where instructions can be executed independently by different threads



Parallel regions

These are blocks of code where instructions can be executed independently by different threads

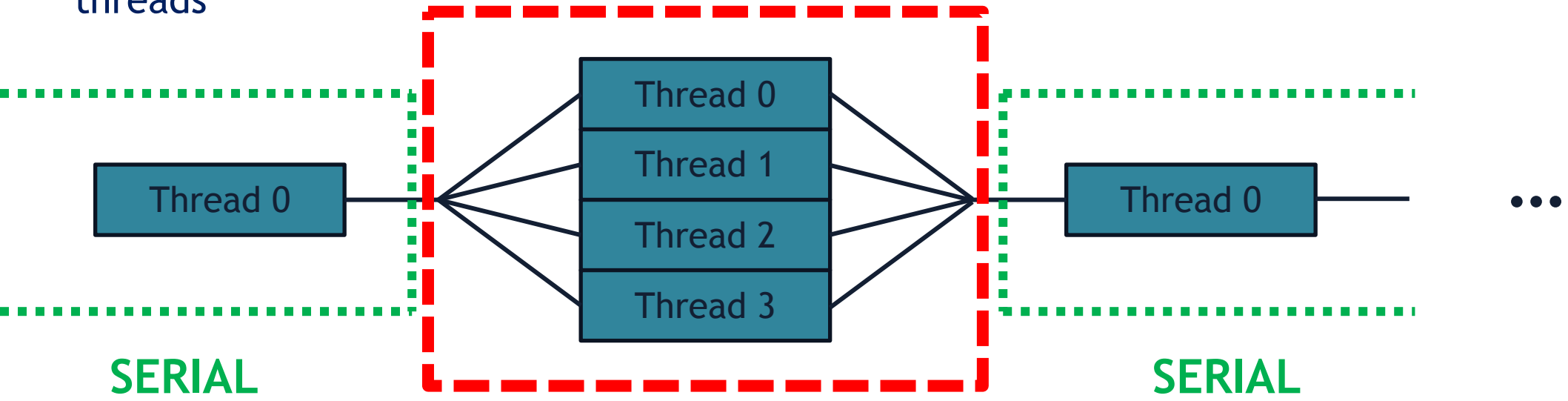


Other names for
Thread 0:

- Main
- Parent
- Manager

Parallel regions

These are blocks of code where instructions can be executed independently by different threads

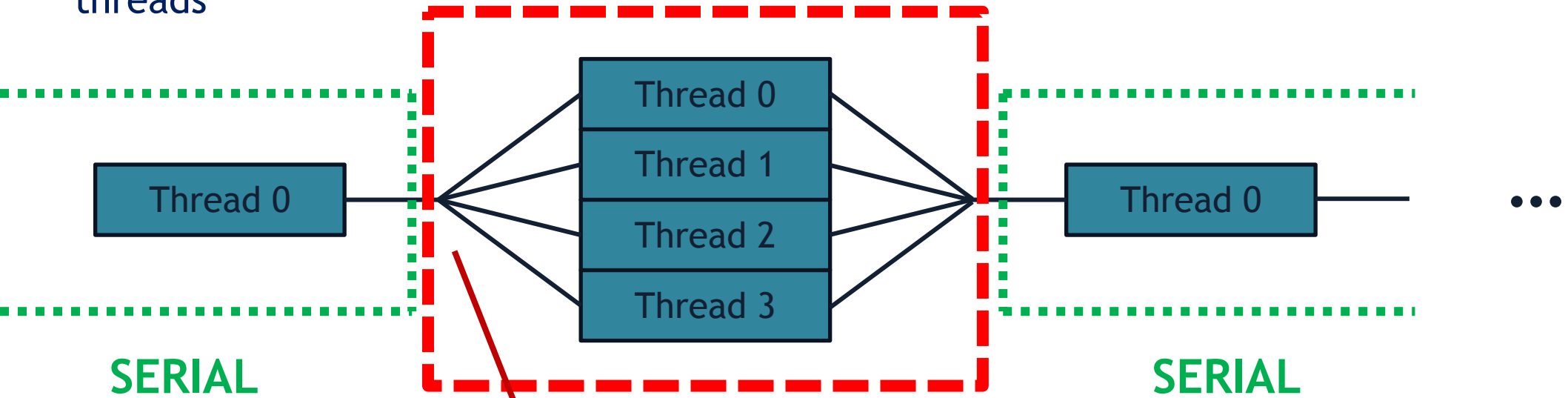


Other names for **Thread 0**:

- Main
- Parent
- Manager

Parallel regions

These are blocks of code where instructions can be executed independently by different threads



Other names for **Thread 0**:

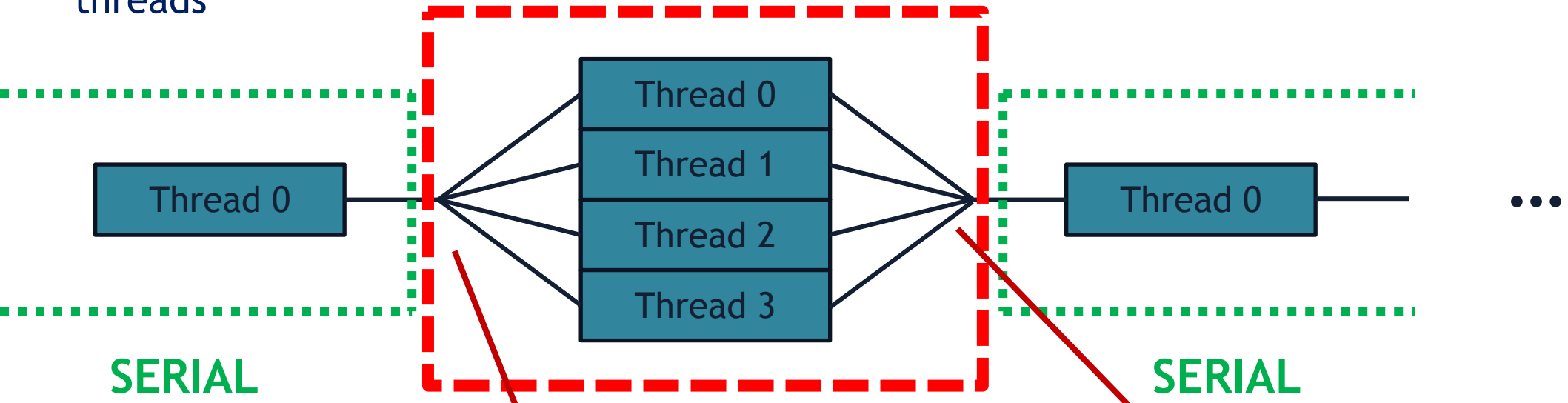
- Main
- Parent
- Manager

PARALLEL

A team of threads is created at the beginning of the parallel region, and we distribute instructions to each of them

Parallel regions

These are blocks of code where instructions can be executed independently by different threads



Other names for **Thread 0**:

- Main
- Parent
- Manager

PARALLEL

A team of threads is created at the beginning of the parallel region, and we distribute instructions to each of them

As threads finish their instructions, they **wait at the end of the region until the entire team is done**. The team is then destroyed, and the next block is executed

Parallel regions: syntax

C/C++

```
#include "omp.h"
.
.
... serial code ...
.
.
#pragma omp parallel
{
    ... instructions ...
}
.
.
... serial code ...
.
.
```

Fortran

```
use :: omp_lib
.
.
... serial code ...
.
.
!$omp parallel
   .
    ... instructions ...
   .
!$omp end parallel
.
.
... serial code ...
.
.
```

Parallel regions: syntax

OpenMP library header

C/C++

```
1   #include "omp.h"
2   .
3   .
4   ... serial code ...
5   .
6   .
7   #pragma omp parallel
8   {
9       ... instructions ...
10  }
11  .
12  .
13  ... serial code ...
14  .
15  .
```

Fortran

```
1   use :: omp_lib
2   .
3   .
4   ... serial code ...
5   .
6   .
7   !$omp parallel
8       .
9        ... instructions ...
10      .
11  !$omp end parallel
12  .
13  .
14  ... serial code ...
15  .
16  .
```

Parallel regions: syntax

OpenMP library header

OpenMP **parallel** directive

C/C++

```
1   #include "omp.h"
2   .
3   .
4   ... serial code ...
5   .
6   .
7   #pragma omp parallel
8   {
9       ... instructions ...
10  }
11  .
12  .
13  ... serial code ...
14  .
15  .
```

Fortran

```
1   use :: omp_lib
2   .
3   .
4   ... serial code ...
5   .
6   .
7   !$omp parallel
8       .
9        ... instructions ...
10      .
11  !$omp end parallel
12  .
13  .
14  ... serial code ...
15  .
16  .
```

Parallel regions: syntax

OpenMP library header

OpenMP **parallel** directive

PARALLEL BLOCKS

C/C++

```
1   #include "omp.h"
2   .
3   .
4   ... serial code ...
5   .
6   .
7   #pragma omp parallel
8   {
9       ... instructions ...
10  }
11  .
12  .
13  ... serial code ...
14  .
15  .
```

Fortran

```
1   use :: omp_lib
2   .
3   .
4   ... serial code ...
5   .
6   .
7   !$omp parallel
8       .
9         ... instructions ...
10      .
11  !$omp end parallel
12  .
13  .
14  ... serial code ...
15  .
16  .
```

Hello World: serial code

Find the code here:

https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/serial/hello_serial.cpp

Hello World: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/serial/hello_serial.cpp

```
26  #include <iostream>
27  #include <string>
28
29  using namespace std;
30
31  int main()
32  {
33      string hello;
34      hello = "Hello, Parallel World!";
35
36      cout << hello << "\n";
37
38      return 0;
39  }
```

Hello World: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/serial/hello_serial.cpp

```
26  #include <iostream>
27  #include <string>
28
29  using namespace std;
30
31  int main()
32  {
33      string hello;
34      hello = "Hello, Parallel World!";
35
36      cout << hello << "\n";
37
38      return 0;
39  }
```

Allocate message in memory

Hello World: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/serial/hello_serial.cpp

```
26  #include <iostream>
27  #include <string>
28
29  using namespace std;
30
31  int main()
32  {
33      string hello;
34      hello = "Hello, Parallel World!";
35
36      cout << hello << "\n";
37
38      return 0;
39  }
```

Allocate message in memory

Print message to standard output

Hello World: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/serial/hello_serial.cpp

```
26  #include <iostream>
27  #include <string>
28
29  using namespace std;
30
31  int main()
32  {
33      string hello;
34      hello = "Hello, Parallel World!";
35
36      cout << hello << "\n";
37
38      return 0;
39  }
```

What do we want?

Make each available thread print the hello message

Hello World: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/serial/hello_serial.cpp

```
26  #include <iostream>
27  #include <string>
28
29  using namespace std;
30
31  int main()
32  {
33      string hello;
34      hello = "Hello, Parallel World!";
35
36      cout << hello << "\n";
37
38      return 0;
39  }
```

What do we want?

Make each available thread print the hello message

Insert OpenMP **parallel** directives

Hello World: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/serial/hello_serial.cpp

```
26  #include <iostream>
27  #include <string>
28
29  using namespace std;
30
31  int main()
32  {
33      string hello;
34      hello = "Hello, Parallel World!";
35
36      cout << hello << "\n";
37
38      return 0;
39  }
```

POLL What do we want?

Make each available thread print the hello message

Insert OpenMP **parallel** directives

Hello World: parallel code

Find the code here:

https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/omp/hello_omp.cpp

Hello World: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/omp/hello_omp.cpp

```
26  #include <iostream>
27  #include <string>
28  #include "omp.h"
29
30  using namespace std;
31
32  int main()
33  {
34      string hello;
35      hello = "Hello, Parallel World!";
36      #pragma omp parallel
37      {
38          cout << hello << endl;
39      }
40      return 0;
41  }
```

Hello World: parallel code

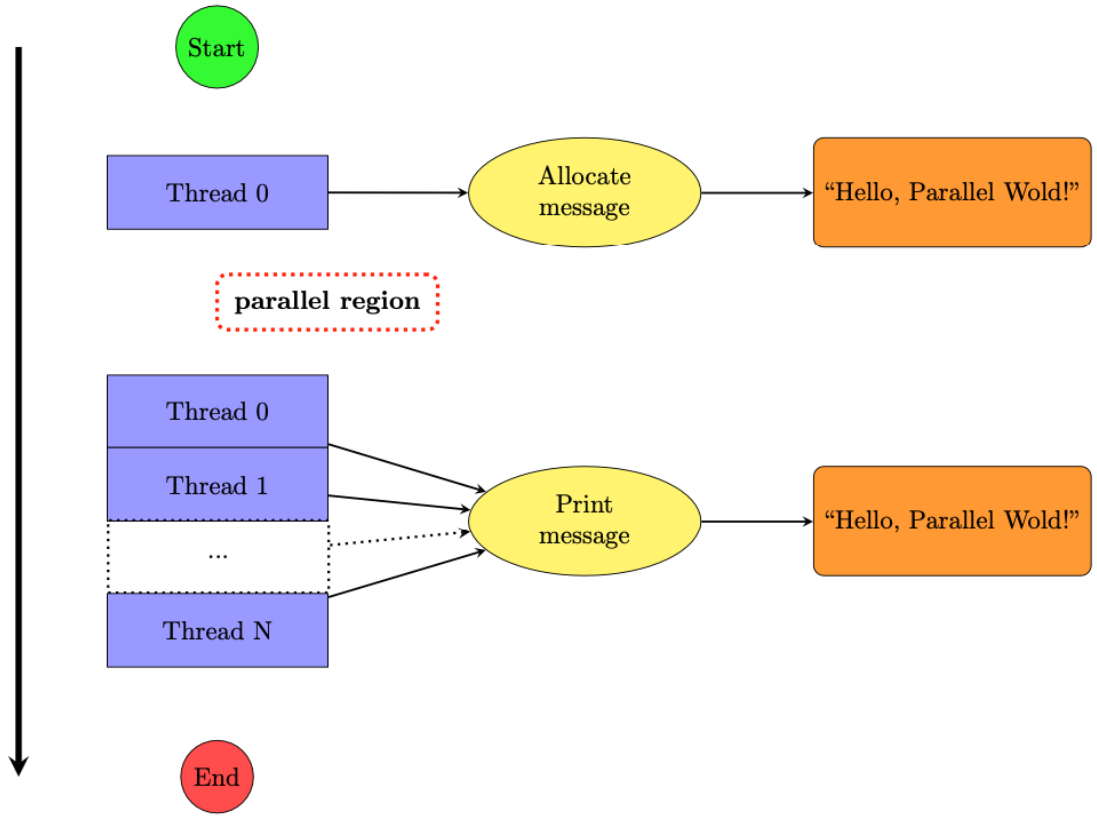
Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/omp/hello_omp.cpp

```
26  #include <iostream>
27  #include <string>
28  #include "omp.h"
29
30  using namespace std;
31
32  int main()
33  {
34      string hello;
35      hello = "Hello, Parallel World!";
36      #pragma omp parallel
37      {
38          cout << hello << endl;
39      }
40      return 0;
41  }
```

Hello World: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/omp/hello_omp.cpp

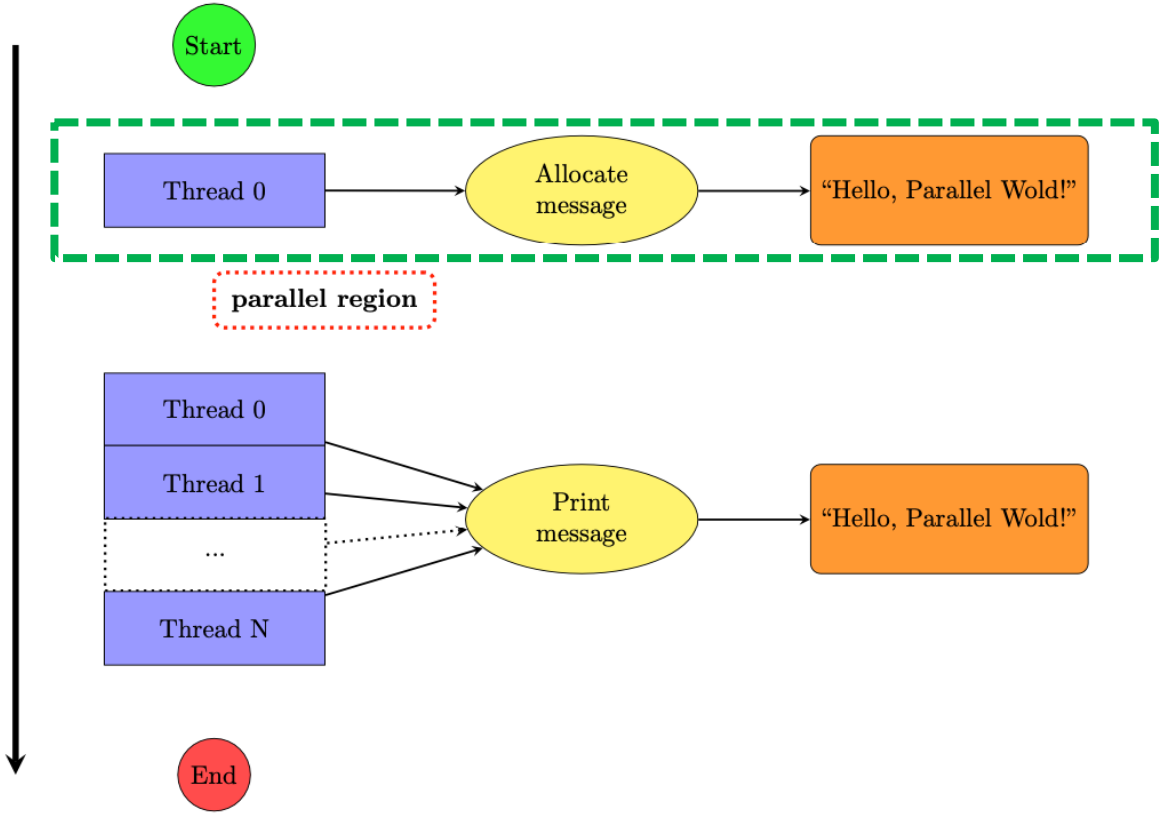
```
26  #include <iostream>
27  #include <string>
28  #include "omp.h"
29
30  using namespace std;
31
32  int main()
33  {
34      string hello;
35      hello = "Hello, Parallel World!";
36      #pragma omp parallel
37      {
38          cout << hello << endl;
39      }
40      return 0;
41  }
```



Hello World: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/omp/hello_omp.cpp

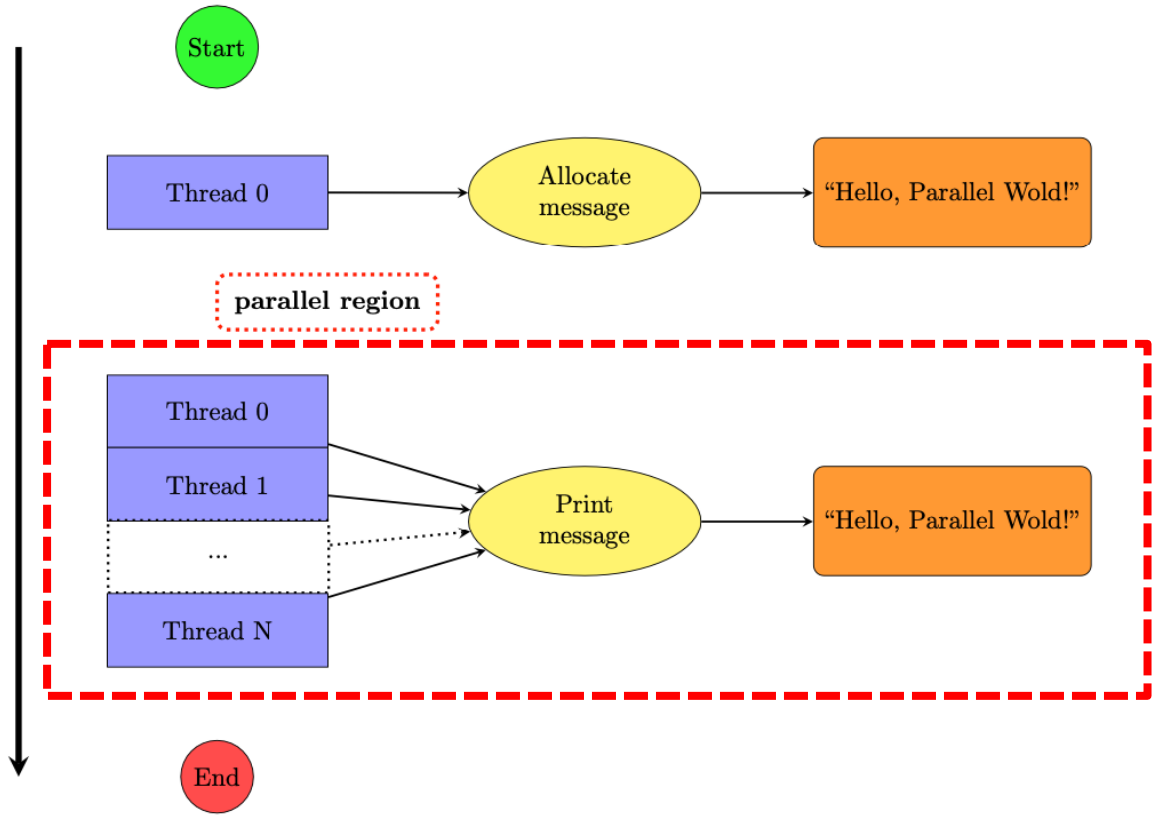
```
26  #include <iostream>
27  #include <string>
28  #include "omp.h"
29
30  using namespace std;
31
32  int main()
33  {
34      string hello;
35      hello = "Hello, Parallel World!";
36      #pragma omp parallel
37      {
38          cout << hello << endl;
39      }
40      return 0;
41  }
```



Hello World: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/hello/omp/hello_omp.cpp

```
26  #include <iostream>
27  #include <string>
28  #include "omp.h"
29
30  using namespace std;
31
32  int main()
33  {
34      string hello;
35      hello = "Hello, Parallel World!";
36      #pragma omp parallel
37      {
38          cout << hello << endl;
39      }
40      return 0;
41  }
```



Compilation of OpenMP code

The general syntax for compilation is:

```
compiler –[OpenMP flag] <SourceCode> -o <TargetBinary>
```

For the current example:

```
g++ -fopenmp hello_omp.cpp –o hello_omp.exe
```

Compiler	OpenMP flag
GCC	-fopenmp
Intel	-qopenmp
AOCC	-fopenmp

Programming Language	OpenMP Header
C/C++	#include "omp.h"
Fortran	use :: omp_lib

Compilation of OpenMP code

Today, you can just do:

- `module load gcc`
- `make`

Execution of OpenMP code

The number of threads that are invoked in a parallel region (the size of the team) is controlled by **OMP_NUM_THREADS**

Before executing your application, use **export** to set it to the desired number:

```
export OMP_NUM_THREADS=N
```

And then finally run the application:

```
./<TargetBinary>
```

Execution of OpenMP code

Today, we will be using a job script that is sent to ICC's batch scheduler:

```
sbatch hello.jobscript
```

To check on the status of your job:

```
squeue -u $USER
```

Hello World: making sense of output

Use `cat` to check on your output, copy it and paste in the chat!

The output file should be named like:

hello-YourJobID.out

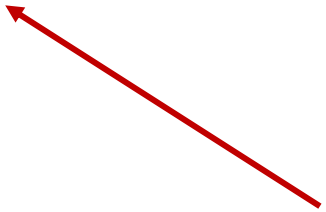
The tab button will help you find it!

Hello World: making sense of output

```
1  Hello, Parallel World!Hello, Parallel World!
2  Hello, Parallel World!
3  Hello, Parallel World!
4  Hello, Parallel World!Hello, Parallel World!
5  Hello, Parallel World!
6
7  Hello, Parallel World!
8
```

Hello World: making sense of output

```
1  Hello, Parallel World!Hello, Parallel World!
2  Hello, Parallel World!
3  Hello, Parallel World!
4  Hello, Parallel World!Hello, Parallel World!
5  Hello, Parallel World!
6
7  Hello, Parallel World!
8
```



Okay, 8 lines were printed, but…

Hello World: making sense of output

```
1  Hello, Parallel World!Hello, Parallel World!
2  Hello, Parallel World!
3  Hello, Parallel World!
4  Hello, Parallel World!Hello, Parallel World!
5  Hello, Parallel World!
6
7  Hello, Parallel World!
8
```

Doubled lines, and…

Okay, 8 lines were printed, but…

Hello World: making sense of output

```
1  Hello, Parallel World!Hello, Parallel World!
2  Hello, Parallel World!
3  Hello, Parallel World!
4  Hello, Parallel World!Hello, Parallel World!
5  Hello, Parallel World!
6
7  Hello, Parallel World!
8
```

Doubled lines, and…

Empty lines!

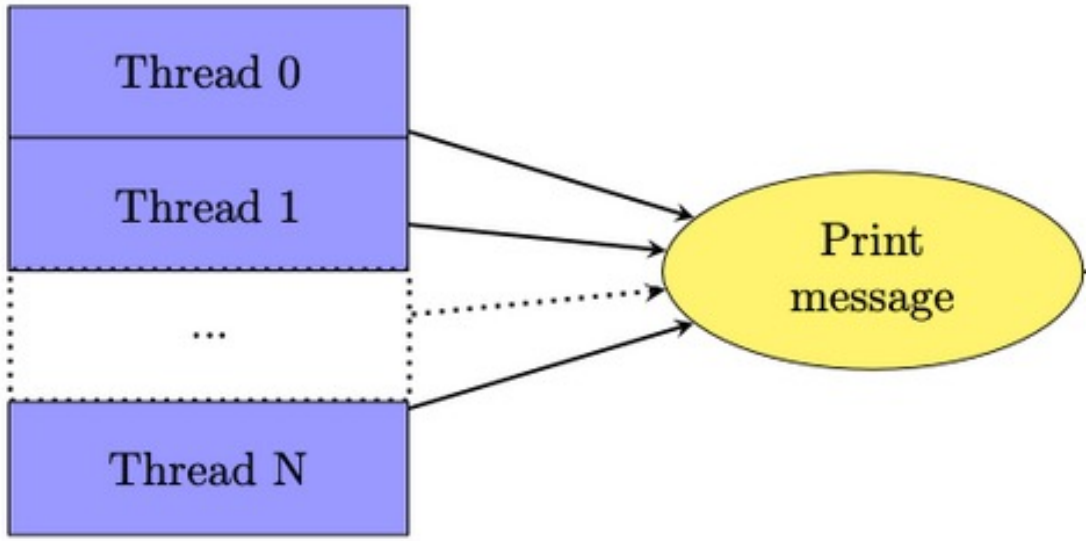
Okay, 8 lines were printed, but…

Hello World: making sense of output

```
1   .
2   .
3   .
4   hello = "Hello, Parallel World!"
5   #pragma omp parallel
6   {
7       cout << hello << endl;
8   }
9   .
10  .
11  .
```

Hello World: making sense of output

```
1   .
2   .
3   .
4   hello = "Hello, Parallel World!"
5   #pragma omp parallel
6   {
7       cout << hello << endl;
8   }
9   .
10  .
11  .
```

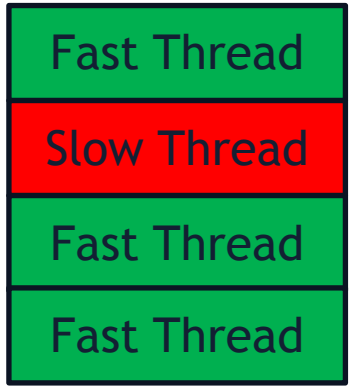


Hello World: making sense of output

```
1   .
2   .
3   .
4   hello = "Hello, Parallel World!"
5   #pragma omp parallel
6   {
7       cout << hello << endl;
8   }
9   .
10  .
11  .
```

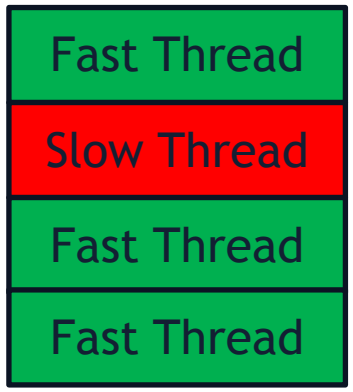
Hello World: making sense of output

```
1   .
2   .
3   .
4   hello = "Hello, Parallel World!"
5   #pragma omp parallel
6   {
7      cout << hello << endl;
8   }
9   .
10  .
11  .
```



Hello World: making sense of output

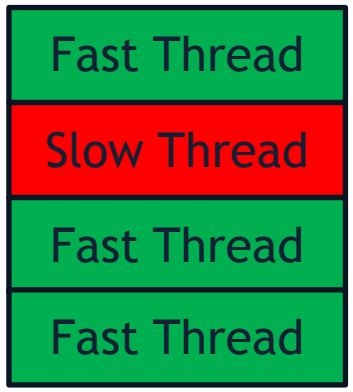
```
1   .
2   .
3   .
4   hello = "Hello, Parallel World!"
5   #pragma omp parallel
6   {
7      cout << hello << endl;
8   }
9   .
10  .
11  .
```




```
1  Hello, Parallel World!Hello, Parallel World!
2  Hello, Parallel World!
3  Hello, Parallel World!
4  Hello, Parallel World!Hello, Parallel World!
5  Hello, Parallel World!
6
7  Hello, Parallel World!
8
```

Hello World: making sense of output

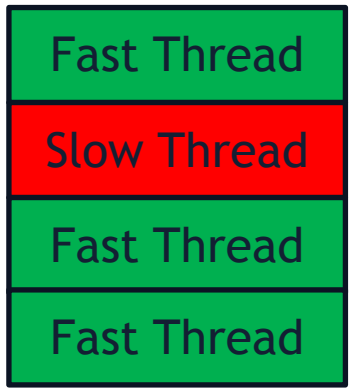
```
1   .
2   .
3   .
4   hello = "Hello, Parallel World!"
5   #pragma omp parallel
6   {
7      cout << hello << endl;
8   }
9   .
10  .
11  .
```




```
1  Hello, Parallel World!Hello, Parallel World!
2  Hello, Parallel World!
3  Hello, Parallel World!
4  Hello, Parallel World!Hello, Parallel World!
5  Hello, Parallel World!
6
7  Hello, Parallel World!
8
```

Hello World: making sense of output

```
1   .
2   .
3   .
4   hello = "Hello, Parallel World!"
5   #pragma omp parallel
6   {
7      cout << hello << endl;
8   }
9   .
10  .
11  .
```



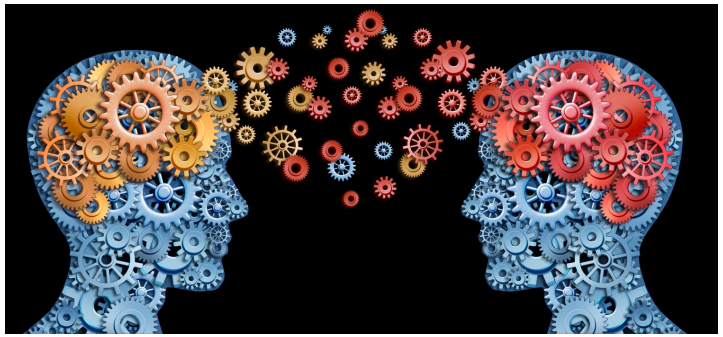
By default, threads execute their instructions **independently** and **asynchronously!**

```
1  Hello, Parallel World!Hello, Parallel World!
2  Hello, Parallel World!
3  Hello, Parallel World!
4  Hello, Parallel World!Hello, Parallel World!
5  Hello, Parallel World!
6
7  Hello, Parallel World!
8
```

Outline

Part A: **Parallel Computing Overview** (45 min)

- Why parallelism?
- Task and Data parallelism
- Parallel computers
- Threads, processes and more



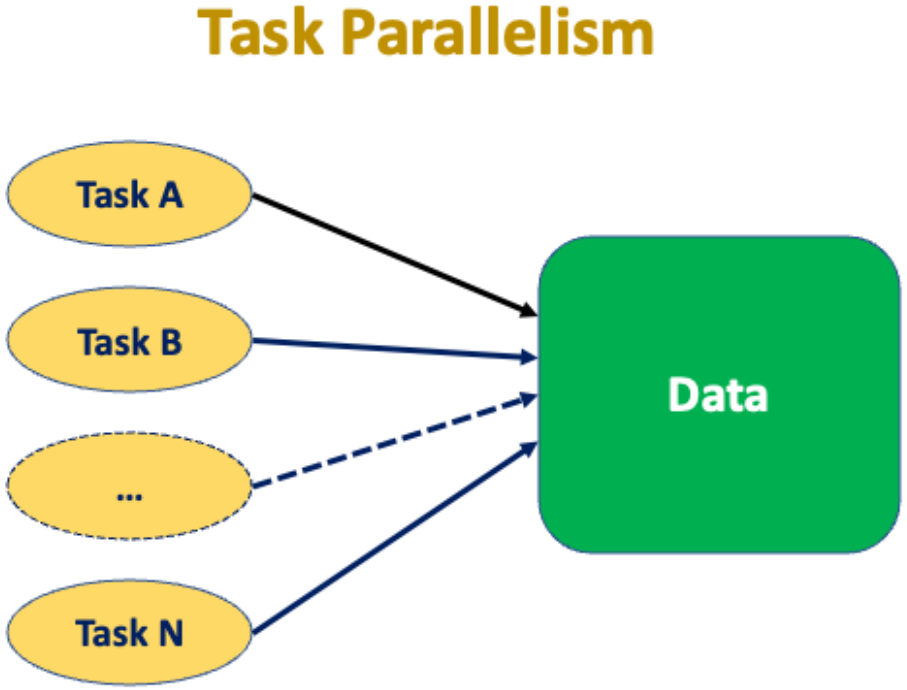
Part B: **How to Parallelize a Code** (75 min)

Hello, Parallel World!
- Parallel regions
- Compilation and execution

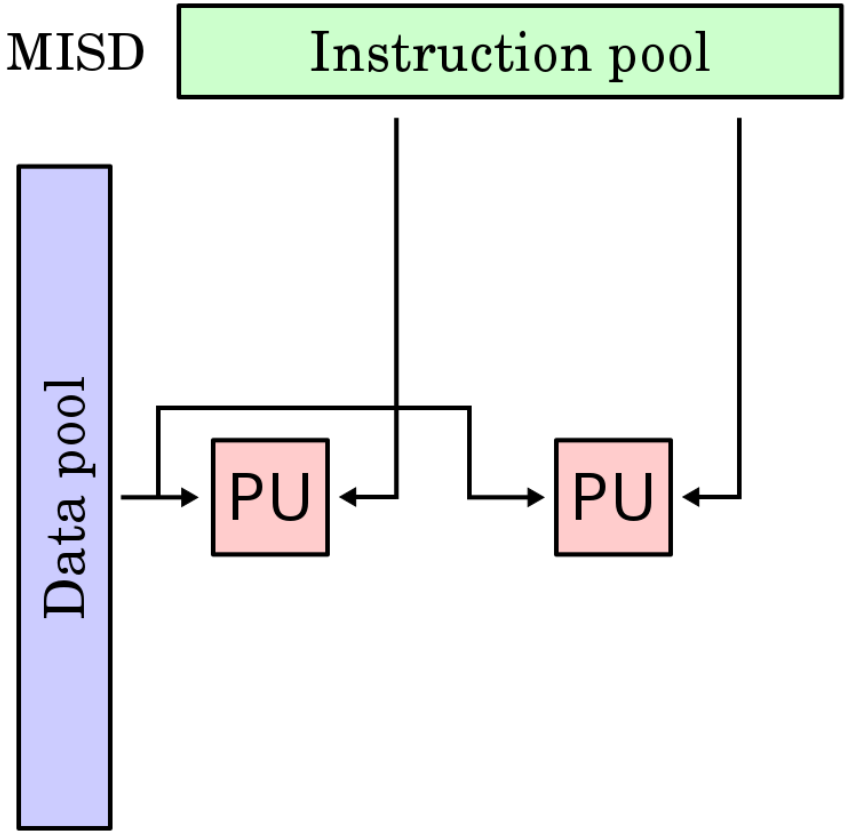
Task Parallelism
- Parallel sections
- Measuring performance

Data Parallelism
- Parallel loops
- Performance and scaling

Task parallelism



Execute different tasks over the same data



OpenMP parallel sections: syntax

C/C++

```
#include "omp.h"
.
.
... serial code ...
.
.
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            ... task A ...
        }
        #pragma omp section
        {
            ... task B ...
        }
    }
}
.
.
... serial code ...
.
.
```

Fortran

```
use :: omp_lib
.
.
... serial code ...
.
.
!$omp parallel
    !$omp sections
        !$omp section
            .
            ... task A ...
            .
        !$omp section
            .
            ... task B ...
            .
  !$omp end sections
!$omp end parallel
.
.
... serial code ...
.
.
```

OpenMP parallel sections: syntax

OpenMP library header

C/C++

```c
#include "omp.h"
.
.
... serial code ...
.
.
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            ... task A ...
        }
        #pragma omp section
        {
            ... task B ...
        }
    }
}
.
.
... serial code ...
.
.
```

Fortran

```fortran
use :: omp_lib
.
.
... serial code ...
.
.
!$omp parallel
    !$omp sections
        !$omp section
            .
            ... task A ...
            .
        !$omp section
            .
            ... task B ...
            .
  !$omp end sections
!$omp end parallel
.
.
... serial code ...
.
.
```

OpenMP parallel sections: syntax

OpenMP library header

OpenMP parallel directives

C/C++

```
#include "omp.h"
.
.
... serial code ...
.
.
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            ... task A ...
        }
        #pragma omp section
        {
            ... task B ...
        }
    }
}
.
.
... serial code ...
.
.
```

Fortran

```
use :: omp_lib
.
.
... serial code ...
.
.
!$omp parallel
    !$omp sections
        !$omp section
            .
            ... task A ...
            .
        !$omp section
            .
            ... task B ...
            .
  !$omp end sections
!$omp end parallel
.
.
... serial code ...
.
.
```

OpenMP parallel sections: syntax

C/C++

```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp sections
10     {
11         #pragma omp section
12         {
13             ... task A ...
14         }
15         #pragma omp section
16         {
17             ... task B ...
18         }
19     }
20 }
21 .
22 .
23 ... serial code ...
24 .
25 .
```

OpenMP library header

OpenMP parallel directives

OpenMP **sections**

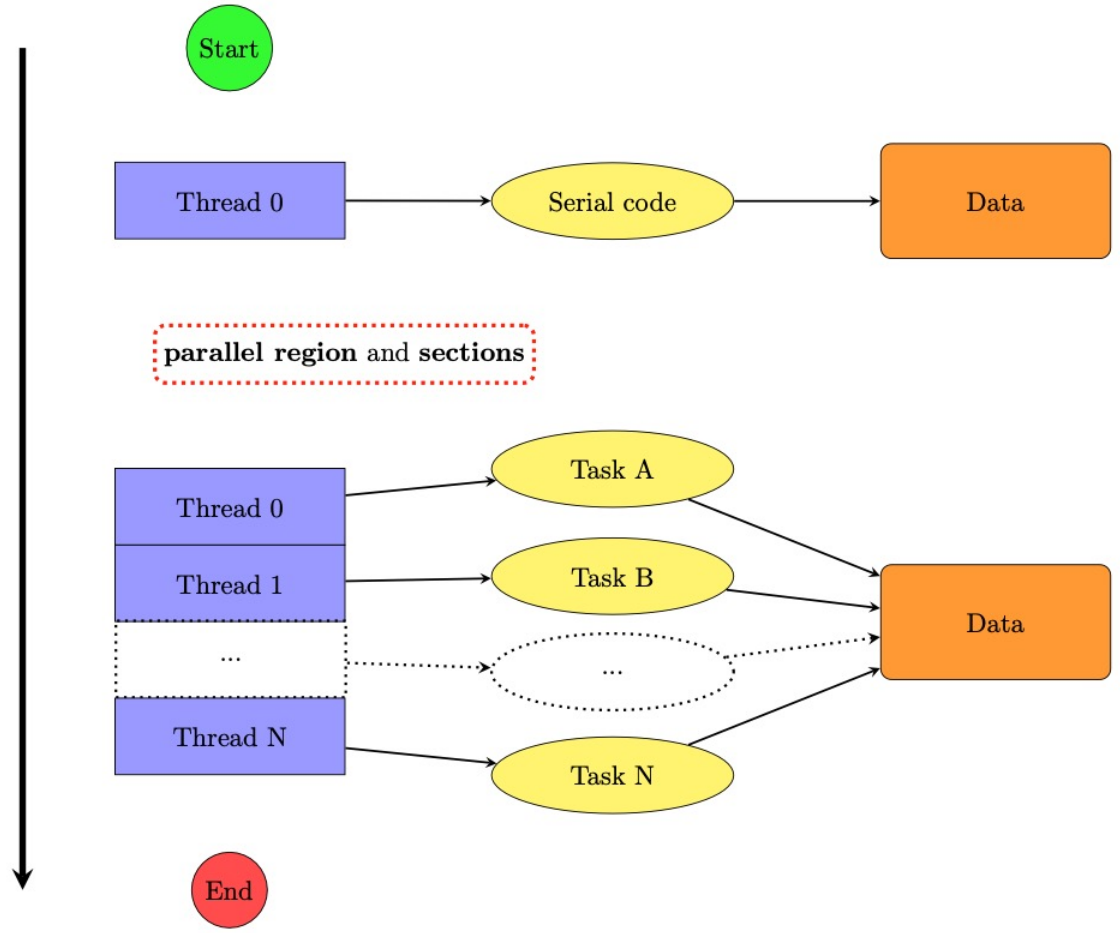
Fortran

```
1  use :: omp_lib
2  .
3  .
4  ... serial code ...
5  .
6  .
7  !$omp parallel
8      !$omp sections
9          !$omp section
10             .
11             ... task A ...
12             .
13         !$omp section
14             .
15             ... task B ...
16             .
17     !$omp end sections
18 !$omp end parallel
19 .
20 .
21 ... serial code ...
22 .
23 .
```

OpenMP parallel sections: workflow

C/C++

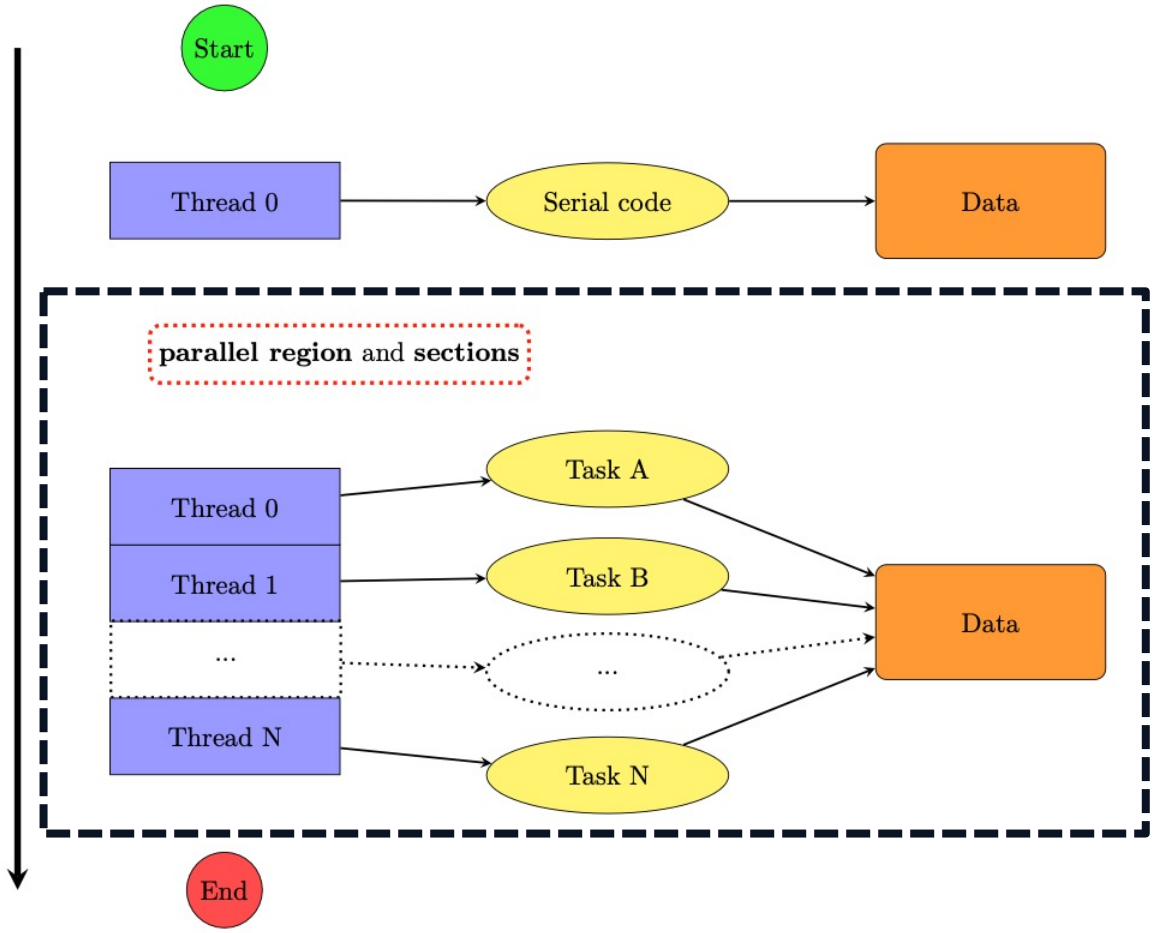
```
#include "omp.h"
.
.
... serial code ...
.
.
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            ... task A ...
        }
        #pragma omp section
        {
            ... task B ...
        }
    }
}
.
.
... serial code ...
.
.
```



OpenMP parallel sections: workflow

C/C++

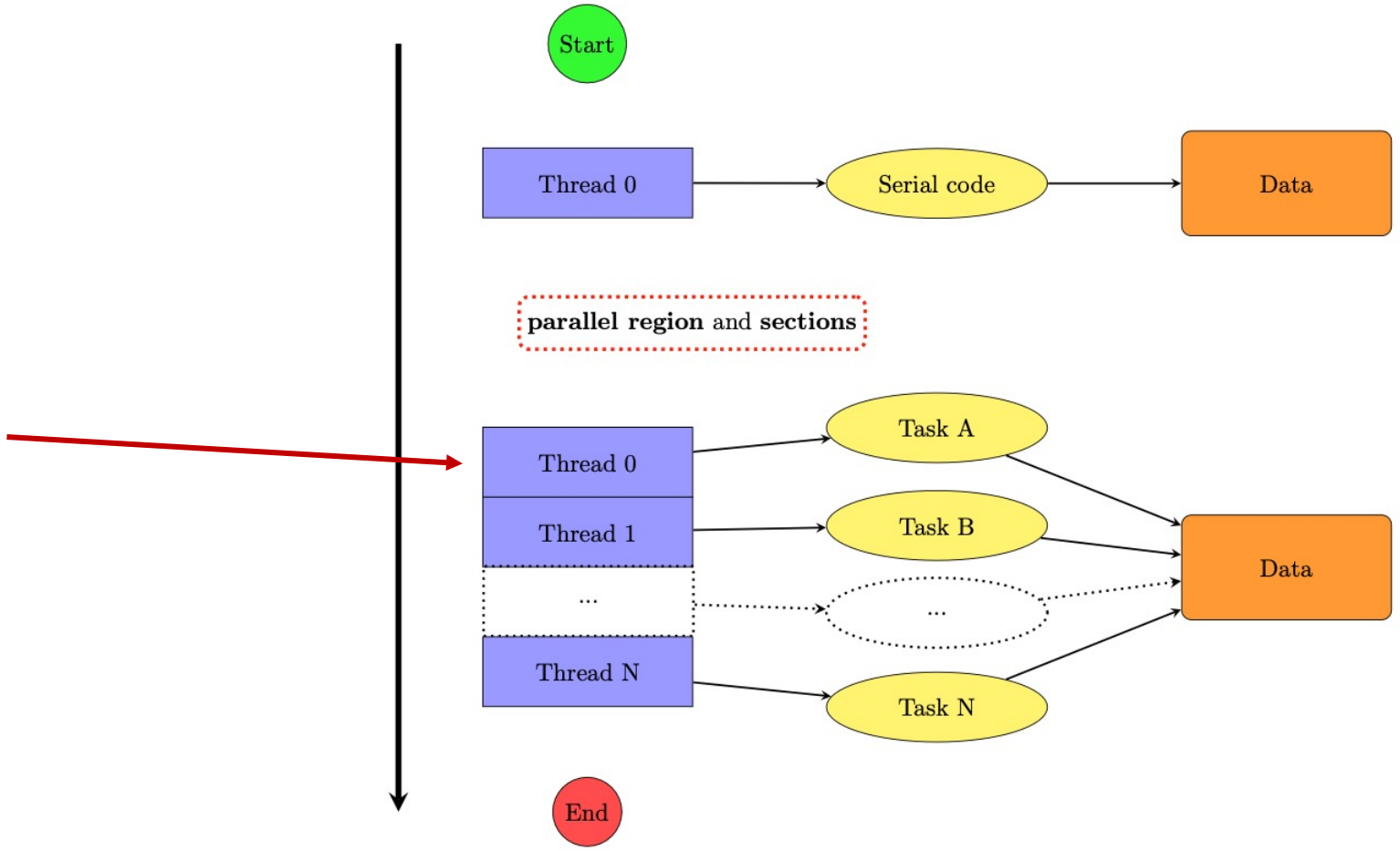
```
#include "omp.h"
.
.
... serial code ...
.

#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            ... task A ...
        }
        #pragma omp section
        {
            ... task B ...
        }
    }
}

.
.
... serial code ...
.
.
```



OpenMP parallel sections: workflow

C/C++

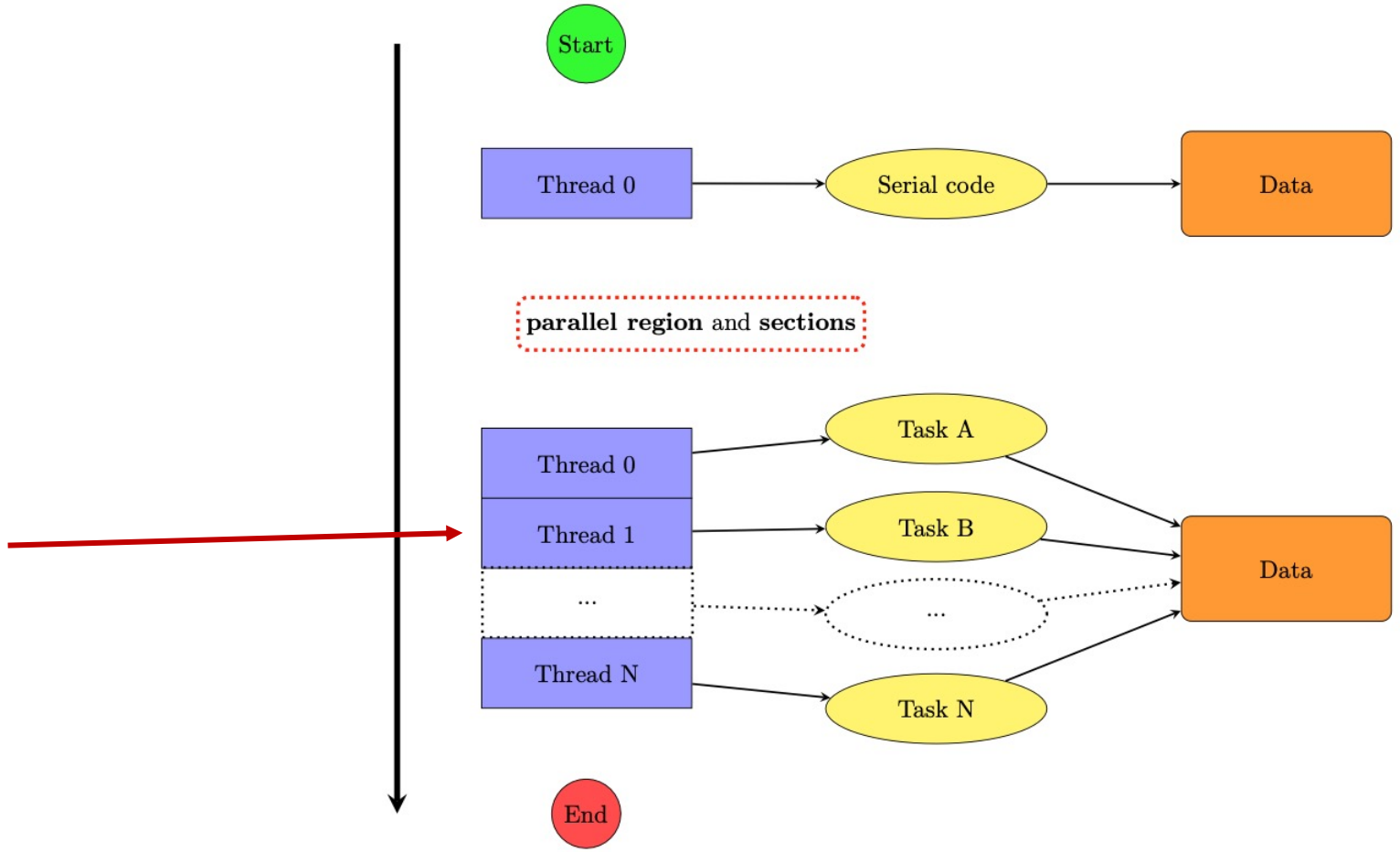
```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp sections
10     {
11        #pragma omp section
12        {
13            ... task A ...
14        }
15        #pragma omp section
16        {
17            ... task B ...
18        }
19     }
20 }
21 .
22 .
23 ... serial code ...
24 .
25 .
```



OpenMP parallel sections: workflow

C/C++

```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp sections
10     {
11         #pragma omp section
12         {
13             ... task A ...
14         }
15         #pragma omp section
16         {
17             ... task B ...
18         }
19     }
20 }
21 .
22 .
23 ... serial code ...
24 .
25 .
```



STATS: serial code

Find the code here:

https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

Calculates simple statistical properties of real numbers in a vector

```cpp
#include <iostream>
#include <vector>
#include <time.h>

#define ORD 1 << 27 // array size

using namespace std;
void get_vec_min(const vector<double> &v);
void get_vec_max(const vector<double> &v);
void get_vec_avg(const vector<double> &v);

```

```cpp
int main()
{
    // create a vector with random noise
    vector<double> vec;
    double elmnt, r;
    srand(time(NULL));
    for (int i = 0; i < ORD; i++)
    {
        r = rand() % RAND_MAX;
        elmnt = double(r) / (RAND_MAX);
        vec.push_back(elmnt);
    }

    // find vector min
    get_vec_min(vec);
    // find vec max
    get_vec_max(vec);
    // find vec avg
    get_vec_avg(vec);

    return 0;
}
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

Calculates simple statistical properties of real numbers in a vector

Libraries

```
26  #include <iostream>
27  #include <vector>
28  #include <time.h>
29
30  #define ORD 1 << 27 // array size
31
32  using namespace std;
33  void get_vec_min(const vector<double> &v);
34  void get_vec_max(const vector<double> &v);
35  void get_vec_avg(const vector<double> &v);
36
```

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

Calculates simple statistical properties of real numbers in a vector

Libraries

Vector size: 2^{27}

```
26  #include <iostream>
27  #include <vector>
28  #include <time.h>
29
30  #define ORD 1 << 27 // array size
31
32  using namespace std;
33  void get_vec_min(const vector<double> &v);
34  void get_vec_max(const vector<double> &v);
35  void get_vec_avg(const vector<double> &v);
36
```

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

Calculates simple statistical properties of real numbers in a vector

```
26  #include <iostream>
27  #include <vector>
28  #include <time.h>
29
30  #define ORD 1 << 27 // array size
31
32  using namespace std;
33  void get_vec_min(const vector<double> &v);
34  void get_vec_max(const vector<double> &v);
35  void get_vec_avg(const vector<double> &v);
36
```

Libraries

Vector size: 2^{27}

Functions declarations

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

Calculates simple statistical properties of real numbers in a vector

Libraries

Vector size: 2^{27}

```
26  #include <iostream>
27  #include <vector>
28  #include <time.h>
29
30  #define ORD 1 << 27 // array size
31
32  using namespace std;
33  void get_vec_min(const vector<double> &v);
34  void get_vec_max(const vector<double> &v);
35  void get_vec_avg(const vector<double> &v);
36
```

Functions declarations

Fill vector with random numbers

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

Calculates simple statistical properties of real numbers in a vector

Libraries

Vector size: 2^{27}

```
26  #include <iostream>
27  #include <vector>
28  #include <time.h>
29
30  #define ORD 1 << 27 // array size
31
32  using namespace std;
33  void get_vec_min(const vector<double> &v);
34  void get_vec_max(const vector<double> &v);
35  void get_vec_avg(const vector<double> &v);
36
```

Functions declarations

Fill vector with random numbers

Calculate properties

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

Function implementation

```
void get_vec_min(const vector<double> &v)
{
    double min;
    min = v[0];
    for (int i = 1; i < v.size(); i++)
    {
        if (v[i] < min)
            min = v[i];
    }
    cout << scientific;
    cout << "Min of vector: " << min << endl;
}
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

Function implementation

```
void get_vec_max(const vector<double> &v)
{
    double max;
    max = v[0];
    for (int i = 1; i < v.size(); i++)
    {
        if (v[i] > max)
            max = v[i];
    }
    cout << scientific;
    cout << "Max of vector: " << max << endl;
}
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

Function implementation

```
void get_vec_avg(const vector<double> &v)
{
    double sum;
    sum = v[0];
    for (int i = 1; i < v.size(); i++)
    {
        sum += v[i];
    }
    cout << scientific;
    cout << "Avg of vector: " << sum / v.size() << endl;
}
```

STATS: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/serial/stats_serial.cpp

```
37  int main()
38  {
39      // create a vector with random noise
40      vector<double> vec;
41      double elmnt, r;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = double(r) / (RAND_MAX);
47          vec.push_back(elmnt);
48      }
49
50      // find vector min
51      get_vec_min(vec);
52      // find vec max
53      get_vec_max(vec);
54      // find vec avg
55      get_vec_avg(vec);
56
57      return 0;
58  }
```

POLL

Function implementation

```
void get_vec_avg(const vector<double> &v)
{
    double sum;
    sum = v[0];
    for (int i = 1; i < v.size(); i++)
    {
        sum += v[i];
    }
    cout << scientific;
    cout << "Avg of vector: " << sum / v.size() << endl;
}
```

STATS: parallel code

Find the code here:

https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

STATS: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29  #include "omp.h"
30
31  #define ORD 1 << 27 // array size
32
33  using namespace std;
34  void get_vec_min(const vector<double> &v);
35  void get_vec_max(const vector<double> &v);
36  void get_vec_avg(const vector<double> &v);
37
```

OpenMP Library

STATS: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

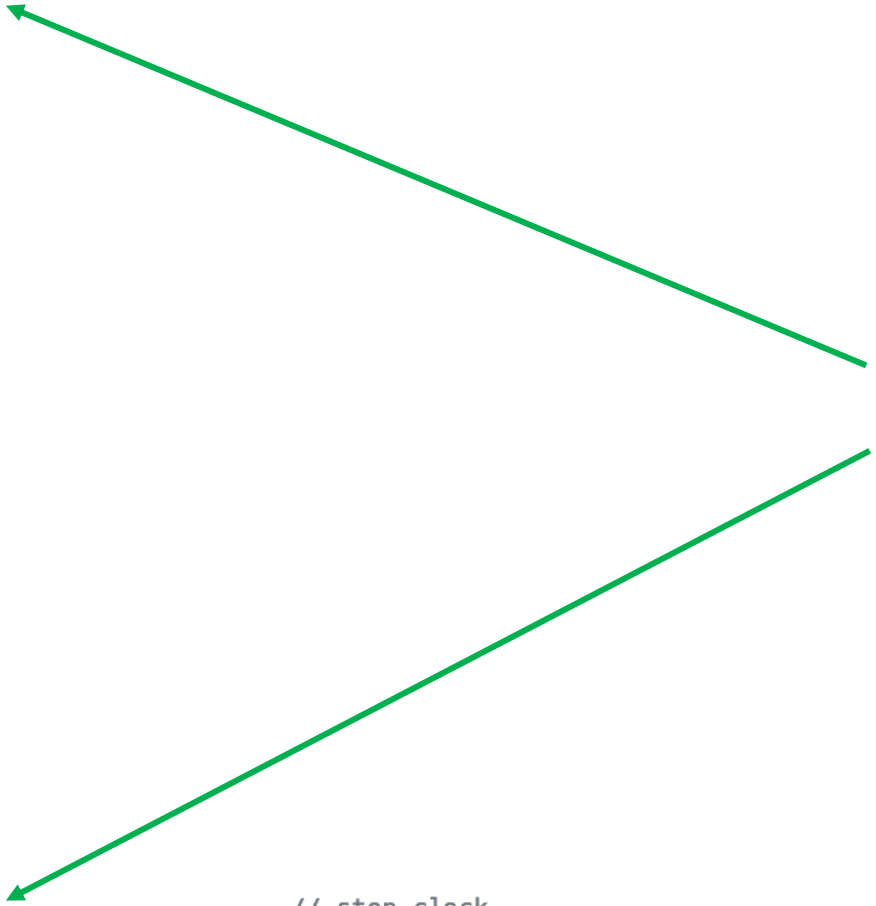
```
38  int main()
39  {
40      // create a vector with random noise
41      vector<double> vec;
42      double elmnt, r;
43      srand(time(NULL));
44      for (int i = 0; i < ORD; i++)
45      {
46          r = rand() % RAND_MAX;
47          elmnt = double(r) / (RAND_MAX);
48          vec.push_back(elmnt);
49      }
50
51      // variables for timing
52      struct timeval start_time, stop_time, elapsed_time;
53
```

Variables to measure elapsed time

STATS: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

```
54      gettimeofday(&start_time, NULL); // start clock
55      #pragma omp parallel
56      {
57          #pragma omp sections
58          {
59              #pragma omp section
60              {
61                  // find vector min
62                  get_vec_min(vec);
63              }
64              #pragma omp section
65              {
66                  // find vec max
67                  get_vec_max(vec);
68              }
69              #pragma omp section
70              {
71                  // find vec avg
72                  get_vec_avg(vec);
73              }
74          }
75      }
76      gettimeofday(&stop_time, NULL);                 // stop clock
```



Start/stop clock

STATS: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

```
54      gettimeofday(&start_time, NULL); // start clock
55      #pragma omp parallel
56      {
57          #pragma omp sections
58          {
59              #pragma omp section
60              {
61                  // find vector min
62                  get_vec_min(vec);
63              }
64              #pragma omp section
65              {
66                  // find vec max
67                  get_vec_max(vec);
68              }
69              #pragma omp section
70              {
71                  // find vec avg
72                  get_vec_avg(vec);
73              }
74          }
75      }
76      gettimeofday(&stop_time, NULL);                // stop clock
```

Parallel Block

STATS: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

```
54      gettimeofday(&start_time, NULL); // start clock
55      #pragma omp parallel
56      {
57          #pragma omp sections
58          {
59              #pragma omp section
60              {
61                  // find vector min
62                  get_vec_min(vec);
63              }
64              #pragma omp section
65              {
66                  // find vec max
67                  get_vec_max(vec);
68              }
69              #pragma omp section
70              {
71                  // find vec avg
72                  get_vec_avg(vec);
73              }
74          }
75      }
76      gettimeofday(&stop_time, NULL);                 // stop clock
```

OpenMP **sections**

Anticipates that instructions to individual threads are coming…

STATS: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

```
54      gettimeofday(&start_time, NULL); // start clock
55      #pragma omp parallel
56      {
57          #pragma omp sections
58          {
59              #pragma omp section
60              {
61                  // find vector min
62                  get_vec_min(vec);
63              }
64              #pragma omp section
65              {
66                  // find vec max
67                  get_vec_max(vec);
68              }
69              #pragma omp section
70              {
71                  // find vec avg
72                  get_vec_avg(vec);
73              }
74          }
75      }
76      gettimeofday(&stop_time, NULL);                 // stop clock
```

OpenMP **section**

Explicitly tells each thread what to do

STATS: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp

What about the implementation of each function?

```
83  void get_vec_min(const vector<double> &v)
84  {
85      double min;
86      min = v[0];
87      for (int i = 1; i < v.size(); i++)
88      {
89          if (v[i] < min)
90              min = v[i];
91      }
92      cout << scientific;
93      cout << "Min of vector: " << min << endl;
94  }
```

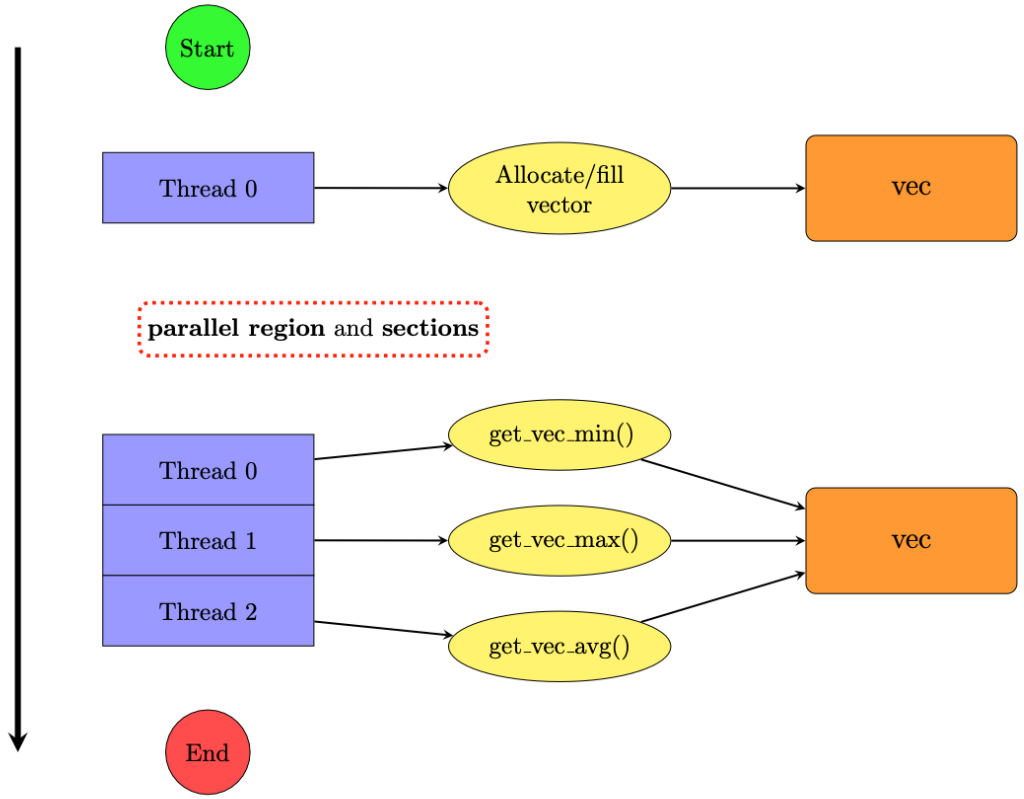
```
96  void get_vec_max(const vector<double> &v)
97  {
98      double max;
99      max = v[0];
100     for (int i = 1; i < v.size(); i++)
101     {
102         if (v[i] > max)
103             max = v[i];
104     }
105     cout << scientific;
106     cout << "Max of vector: " << max << endl;
107 }
```

```
109 void get_vec_avg(const vector<double> &v)
110 {
111     double sum;
112     sum = v[0];
113     for (int i = 1; i < v.size(); i++)
114     {
115         sum += v[i];
116     }
117     cout << scientific;
118     cout << "Avg of vector: " << sum / v.size() << endl;
119 }
```

They remain the same!

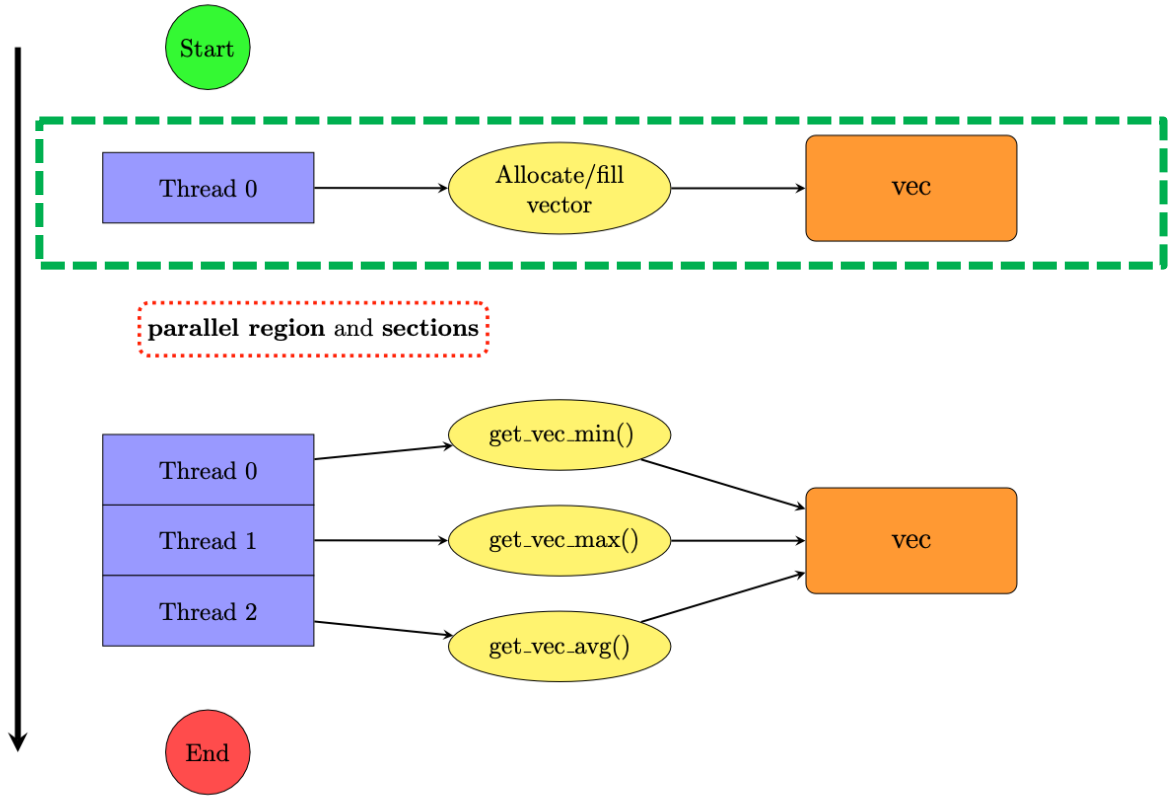
STATS: workflow diagram

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp



STATS: workflow diagram

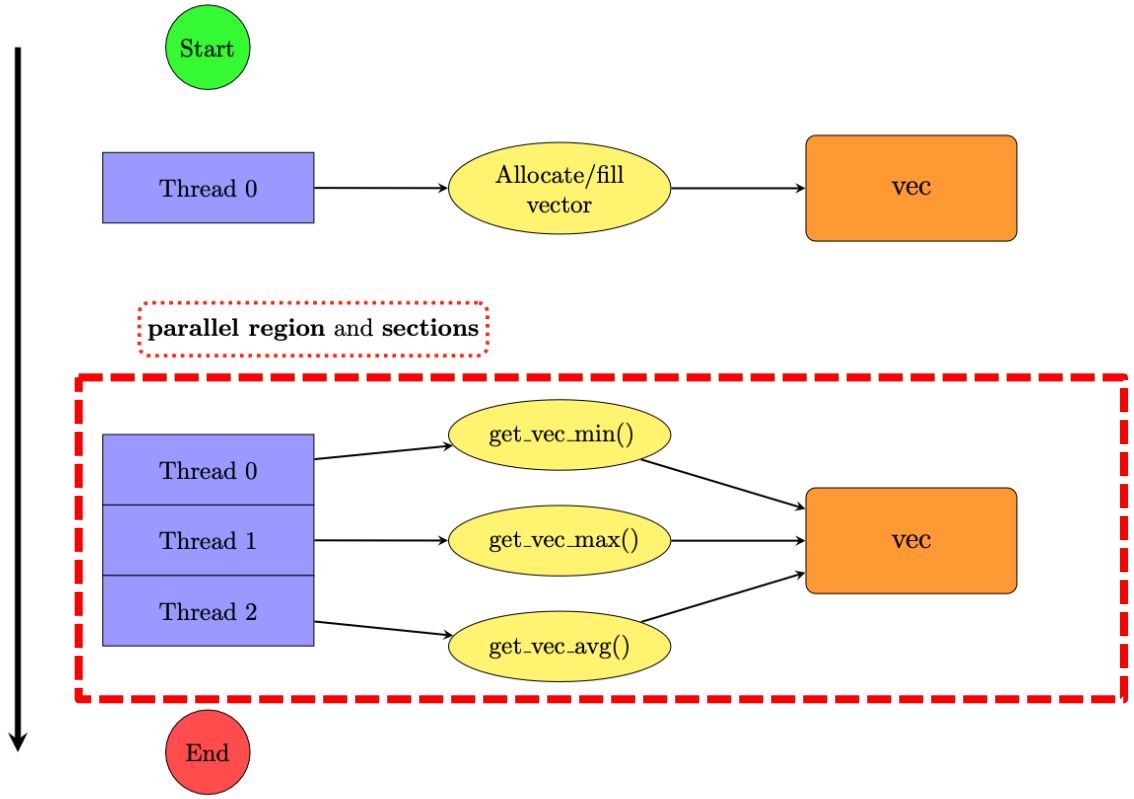
Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp




```
38  int main()
39  {
40      // create a vector with random noise
41      vector<double> vec;
42      double elmnt, r;
43      srand(time(NULL));
44      for (int i = 0; i < ORD; i++)
45      {
46          r = rand() % RAND_MAX;
47          elmnt = double(r) / (RAND_MAX);
48          vec.push_back(elmnt);
49      }
50
51      // variables for timing
52      struct timeval start_time, stop_time, elapsed_time;
```

STATS: workflow diagram

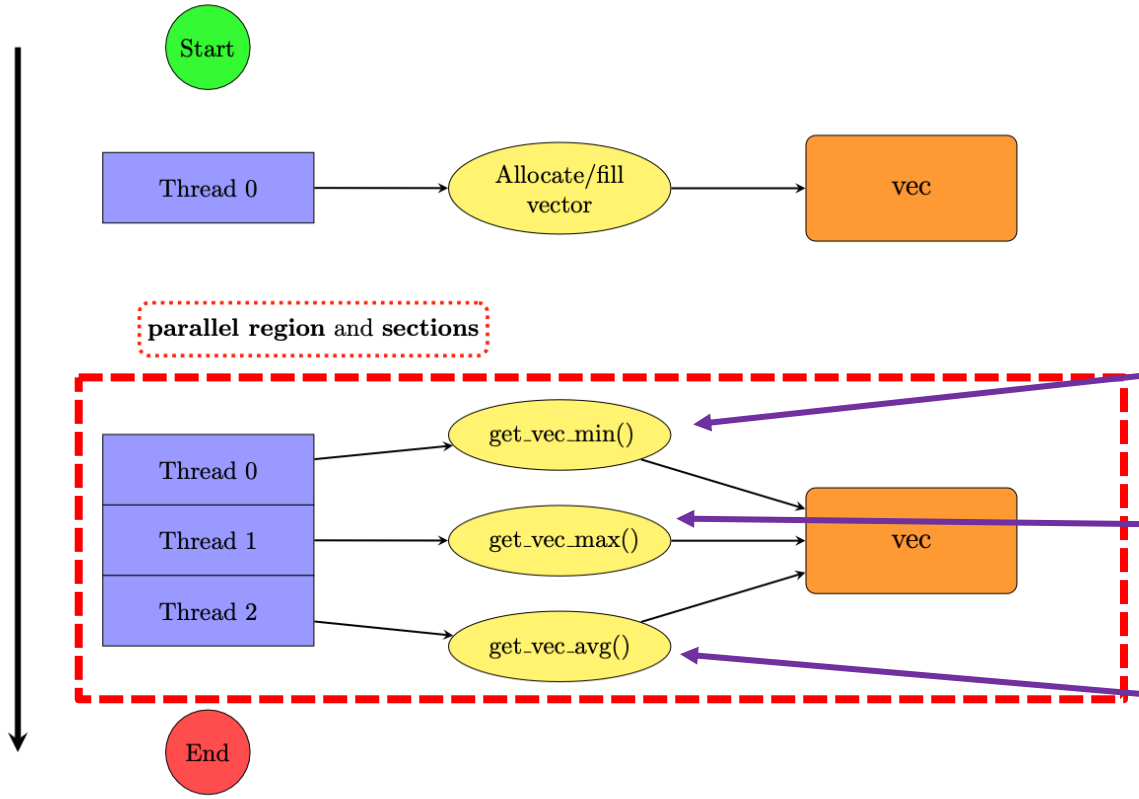
Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp




```
55      #pragma omp parallel
56      {
57          #pragma omp sections
58          {
59              #pragma omp section
60              {
61                  // find vector min
62                  get_vec_min(vec);
63              }
64              #pragma omp section
65              {
66                  // find vec max
67                  get_vec_max(vec);
68              }
69              #pragma omp section
70              {
71                  // find vec avg
72                  get_vec_avg(vec);
73              }
74          }
75      }
```

STATS: workflow diagram

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/stats/omp/stats_omp.cpp




```
55      #pragma omp parallel
56      {
57          #pragma omp sections
58          {
59              #pragma omp section
60              {
61                  // find vector min
62                  get_vec_min(vec);
63              }
64              #pragma omp section
65              {
66                  // find vec max
67                  get_vec_max(vec);
68              }
69              #pragma omp section
70              {
71                  // find vec avg
72                  get_vec_avg(vec);
73              }
74          }
75      }
```

STATS: compilation and execution

- **To compile:**

```
make
```

- **To run:**

```
sbatch stats.jobscript
```

- **To check status:**

```
squeue –u $USER
```

Use `cat` to check on your output, copy it and paste in the chat!

The output file should be named like:

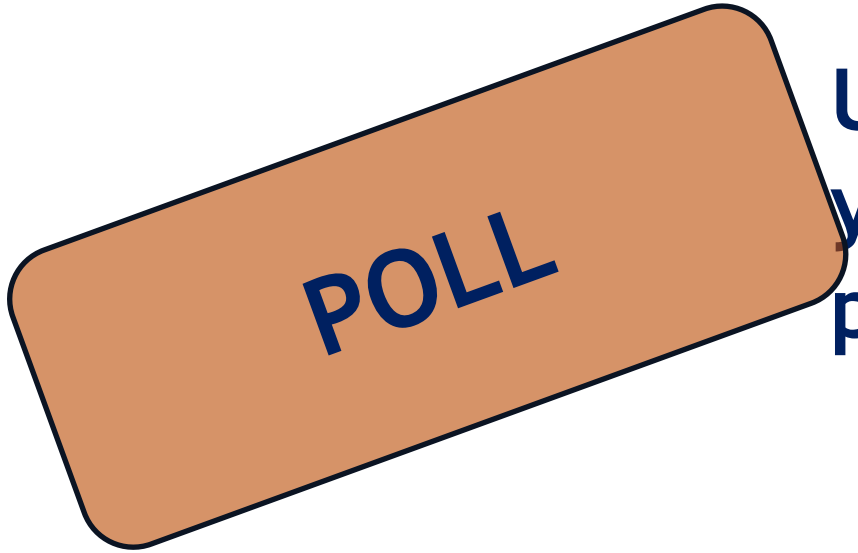
stats-YourJobID.out

The tab button will help you find it!

STATS: compilation and execution

- **To compile:**

```
make
```



- **To run:**

```
sbatch stats.jobscript
```

- **To check status:**

```
squeue –u $USER
```

Use `cat` to check on your output, copy it and paste in the chat!

The output file should be named like:

stats-YourJobID.out

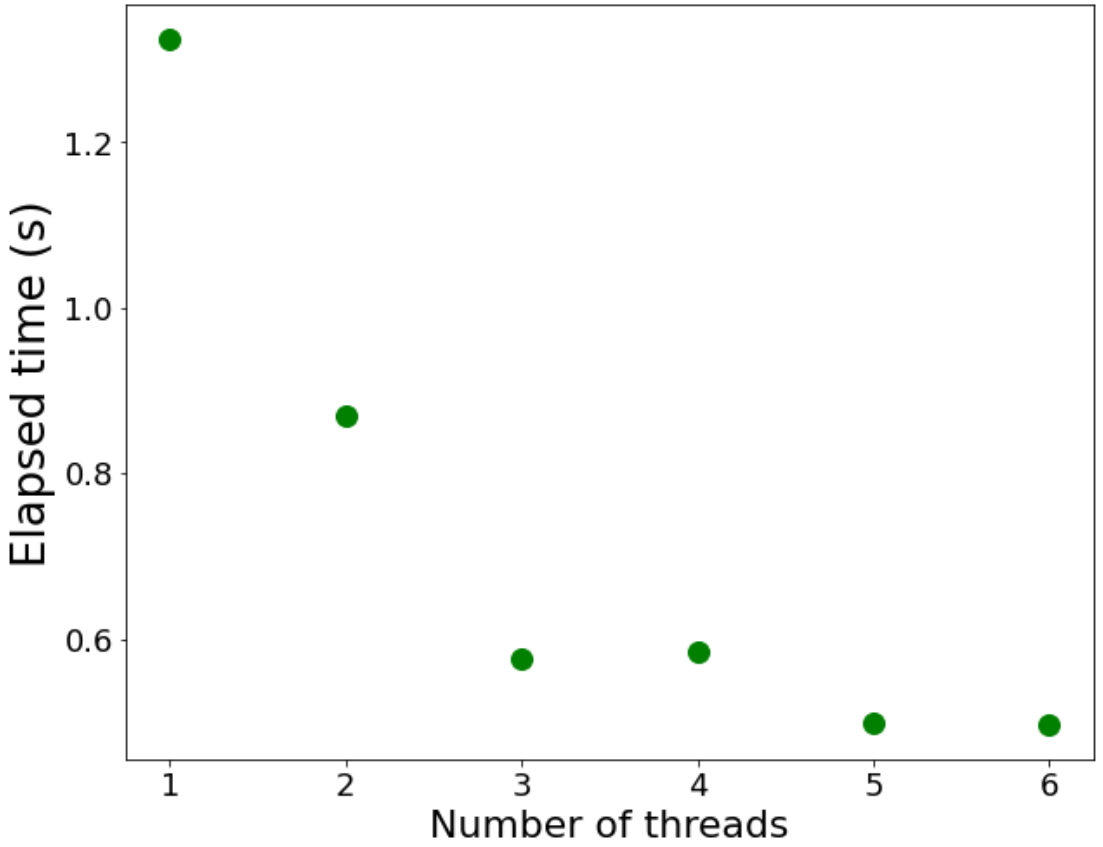
The tab button will help you find it!

STATS: measuring performance

Here's a typical result:

Number of threads	Elapsed time (s)
1	1.32492
2	0.86940
3	0.57525
4	0.58531
5	0.49785
6	0.49656

Raw time measurements are not very helpful…



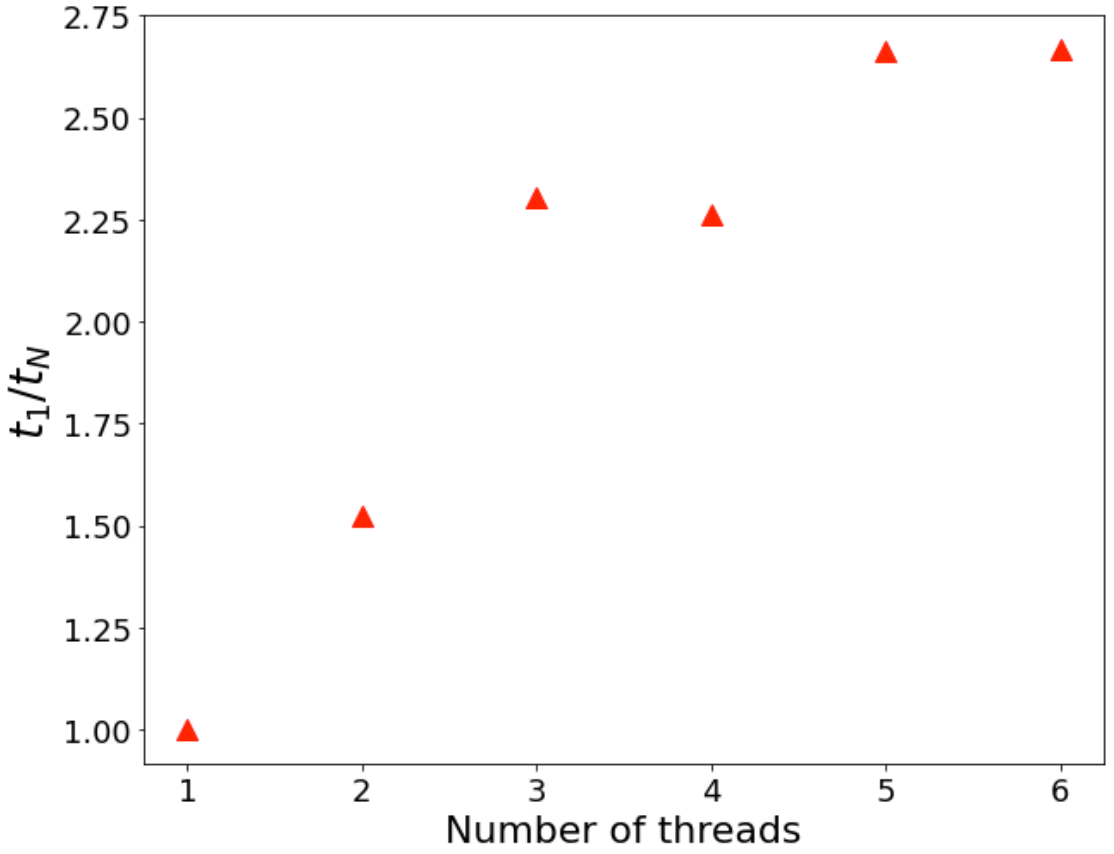
STATS: measuring performance

A better measure is how much faster the code runs in parallel when compared to when it runs serially:

$\frac{t_1}{t_N}$ ← one thread

t_N ← multiple threads

Number of threads	Elapsed time (s)	t_1 / t_N
1	1.32492	1.000
2	0.86940	1.524
3	0.57525	2.303
4	0.58531	2.264
5	0.49785	2.661
6	0.49656	2.668



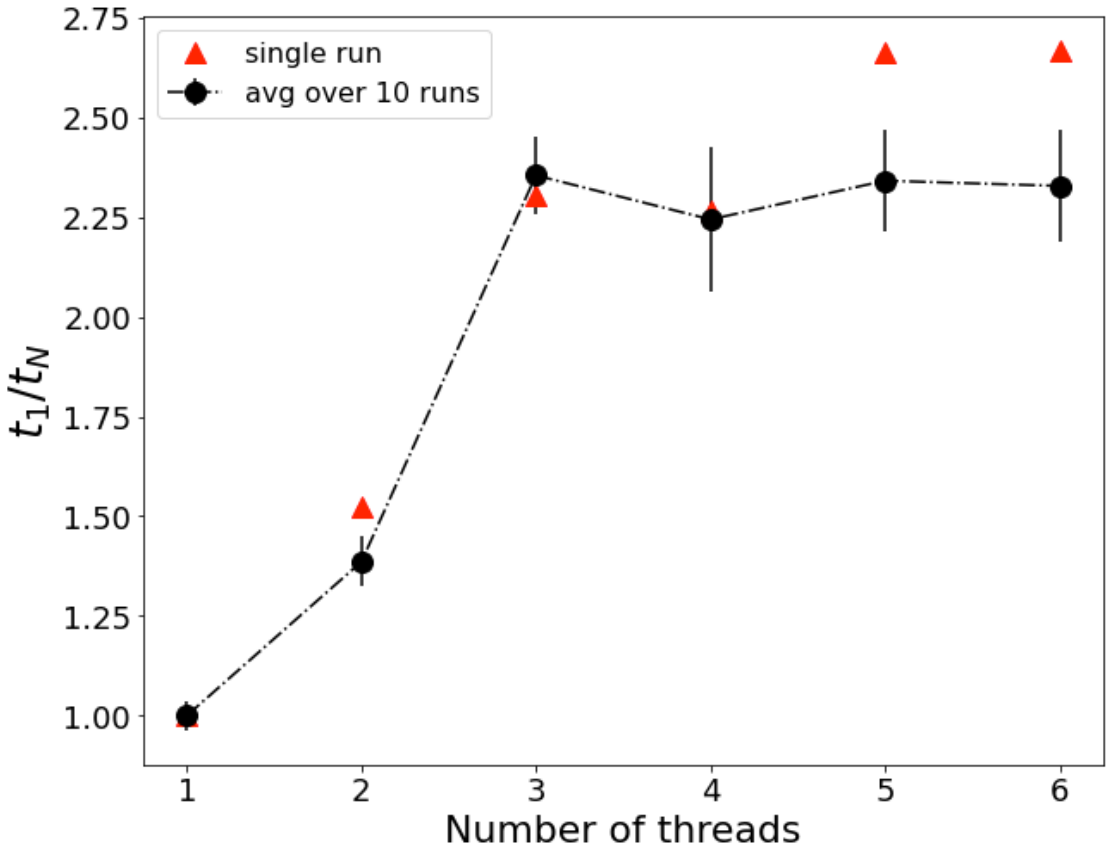
STATS: measuring performance

A better measure is how much faster the code runs in parallel when compared to when it runs serially:

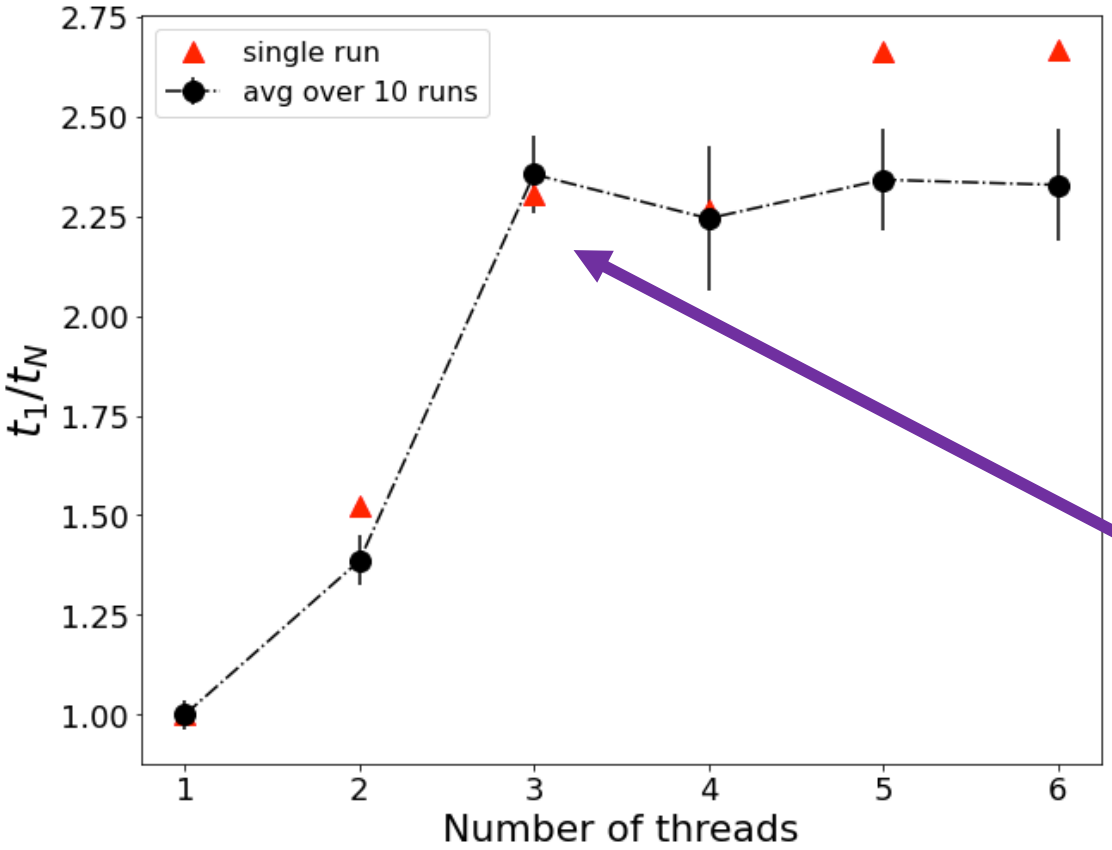
$\frac{t_1}{t_N}$ ← one thread

t_N ← multiple threads

Number of threads	Elapsed time (s)	t_1 / t_N
1	1.32492	1.000
2	0.86940	1.524
3	0.57525	2.303
4	0.58531	2.264
5	0.49785	2.661
6	0.49656	2.668

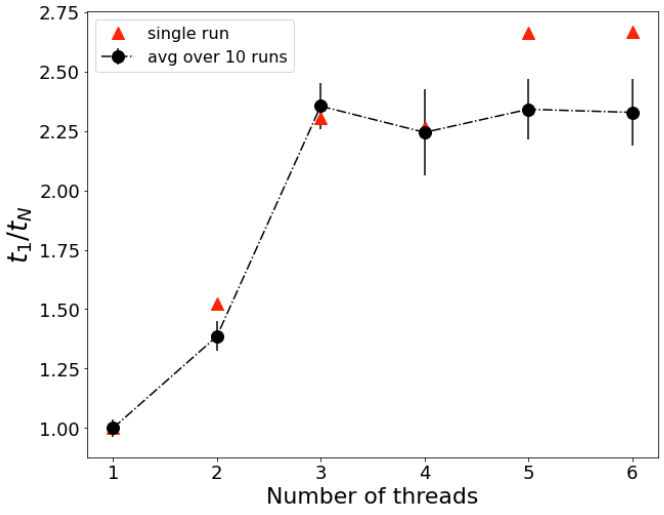


STATS: performance analysis

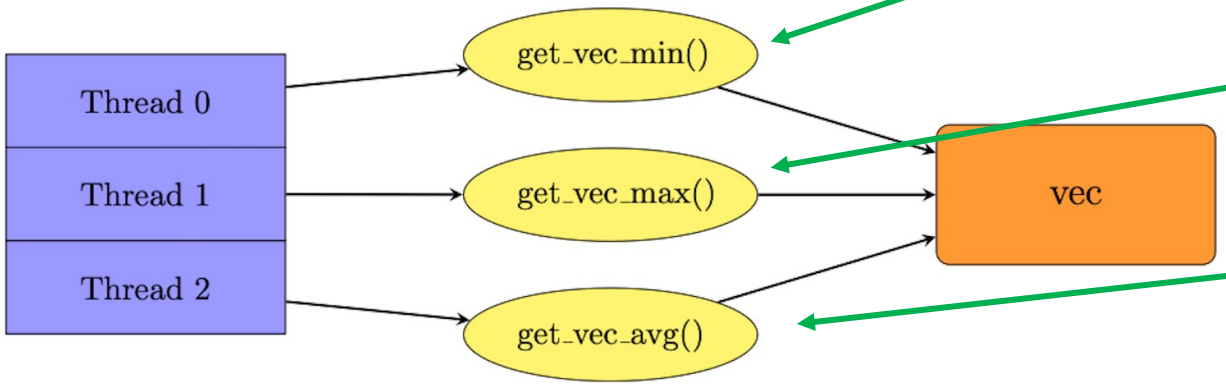


Performance gain does not change for N > 3

STATS: performance analysis

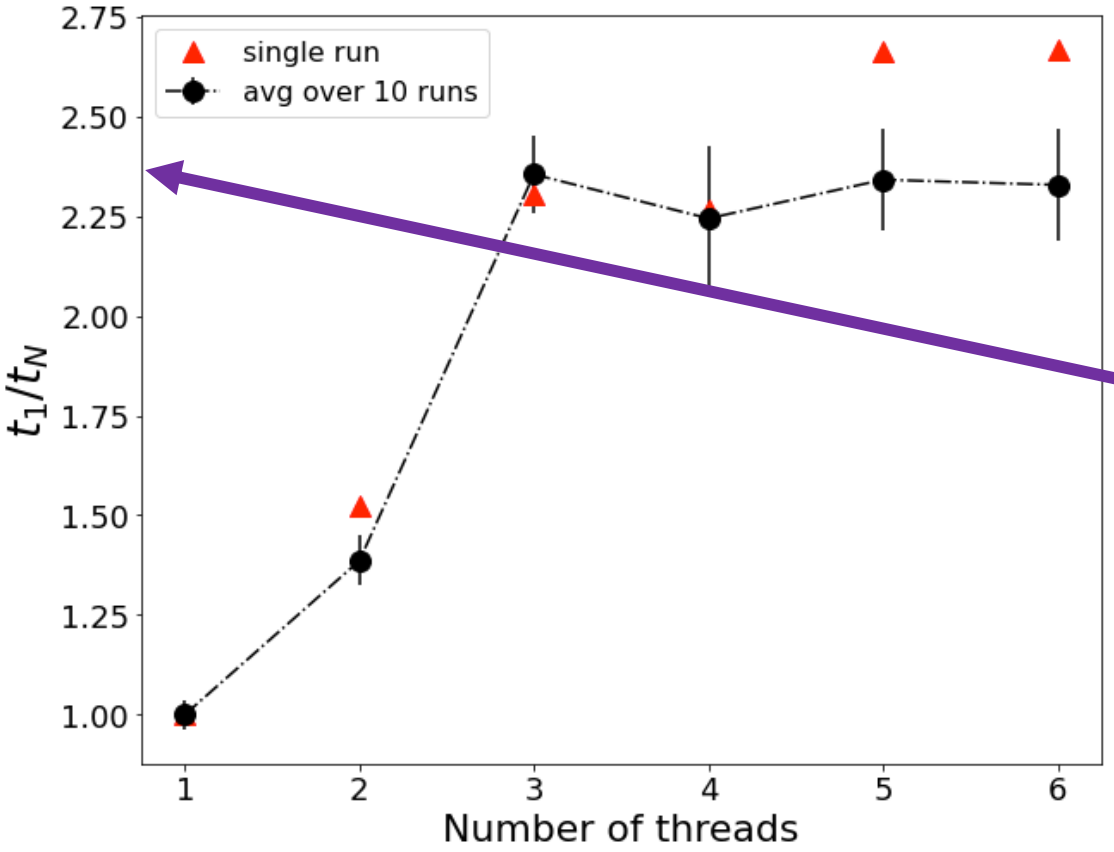


Additional threads are not assigned to any task and just sit idle!




```
55      #pragma omp parallel
56      {
57          #pragma omp sections
58          {
59              #pragma omp section
60              {
61                  // find vector min
62                  get_vec_min(vec);
63              }
64              #pragma omp section
65              {
66                  // find vec max
67                  get_vec_max(vec);
68              }
69              #pragma omp section
70              {
71                  // find vec avg
72                  get_vec_avg(vec);
73              }
74          }
75      }
```

STATS: performance analysis



Performance gain is "only" about 2.4x

STATS: performance analysis

```
83  void get_vec_min(const vector<double> &v)
84  {
85      double min;
86      min = v[0];
87      for (int i = 1; i < v.size(); i++)
88      {
89          if (v[i] < min)
90              min = v[i];
91      }
92      cout << scientific;
93      cout << "Min of vector: " << min << endl;
94  }
```

```
96  void get_vec_max(const vector<double> &v)
97  {
98      double max;
99      max = v[0];
100     for (int i = 1; i < v.size(); i++)
101     {
102         if (v[i] > max)
103             max = v[i];
104     }
105     cout << scientific;
106     cout << "Max of vector: " << max << endl;
107 }
```

```
109 void get_vec_avg(const vector<double> &v)
110 {
111     double sum;
112     sum = v[0];
113     for (int i = 1; i < v.size(); i++)
114     {
115         sum += v[i];
116     }
117     cout << scientific;
118     cout << "Avg of vector: " << sum / v.size() << endl;
119 }
```

STATS: performance analysis

```
83  void get_vec_min(const vector<double> &v)
84  {
85      double min;
86      min = v[0];
87      for (int i = 1; i < v.size(); i++)
88      {
89          if (v[i] < min)
90              min = v[i];
91      }
92      cout << scientific;
93      cout << "Min of vector: " << min << endl;
94  }
```

0.423 s to run

```
96  void get_vec_max(const vector<double> &v)
97  {
98      double max;
99      max = v[0];
100     for (int i = 1; i < v.size(); i++)
101     {
102         if (v[i] > max)
103             max = v[i];
104     }
105     cout << scientific;
106     cout << "Max of vector: " << max << endl;
107 }
```

0.434 s to run

```
109 void get_vec_avg(const vector<double> &v)
110 {
111     double sum;
112     sum = v[0];
113     for (int i = 1; i < v.size(); i++)
114     {
115         sum += v[i];
116     }
117     cout << scientific;
118     cout << "Avg of vector: " << sum / v.size() << endl;
119 }
```

0.588 s to run

STATS: performance analysis

```
83  void get_vec_min(const vector<double> &v)
84  {
85      double min;
86      min = v[0];
87      for (int i = 1; i < v.size(); i++)
88      {
89          if (v[i] < min)
90              min = v[i];
91      }
92      cout << scientific;
93      cout << "Min of vector: " << min << endl;
94  }
```

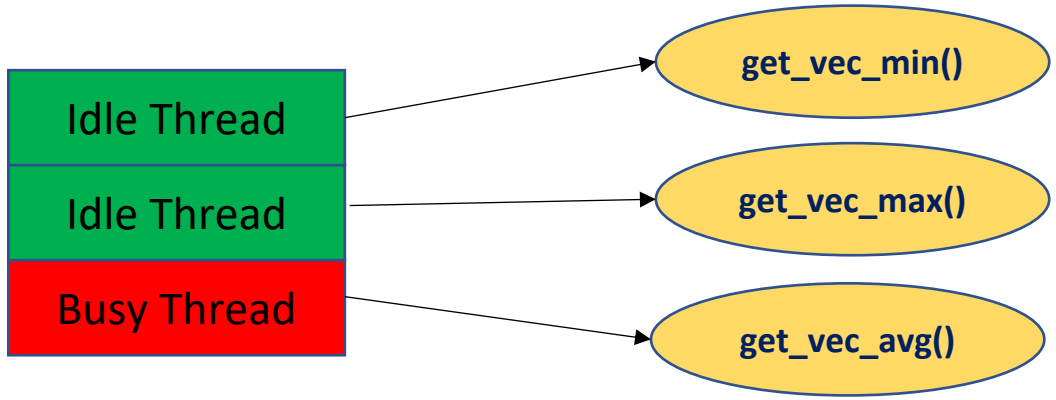
0.423 s to run

```
96  void get_vec_max(const vector<double> &v)
97  {
98      double max;
99      max = v[0];
100     for (int i = 1; i < v.size(); i++)
101     {
102         if (v[i] > max)
103             max = v[i];
104     }
105     cout << scientific;
106     cout << "Max of vector: " << max << endl;
107 }
```

0.434 s to run

```
109 void get_vec_avg(const vector<double> &v)
110 {
111     double sum;
112     sum = v[0];
113     for (int i = 1; i < v.size(); i++)
114     {
115         sum += v[i];
116     }
117     cout << scientific;
118     cout << "Avg of vector: " << sum / v.size() << endl;
119 }
```

0.588 s to run

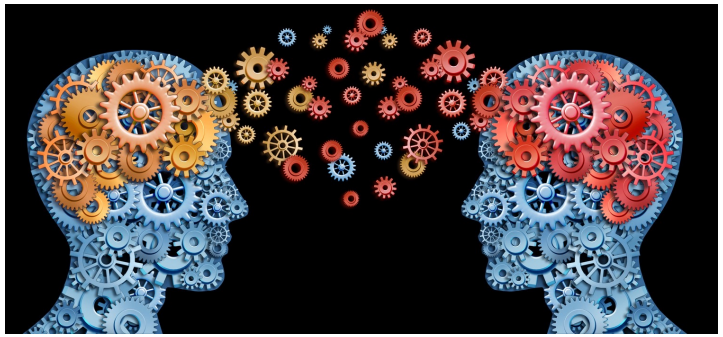


Total execution time is dominated by the thread with the largest workload!

Outline

Part A: **Parallel Computing Overview** (45 min)

- Why parallelism?
- Task and Data parallelism
- Parallel computers
- Threads, processes and more



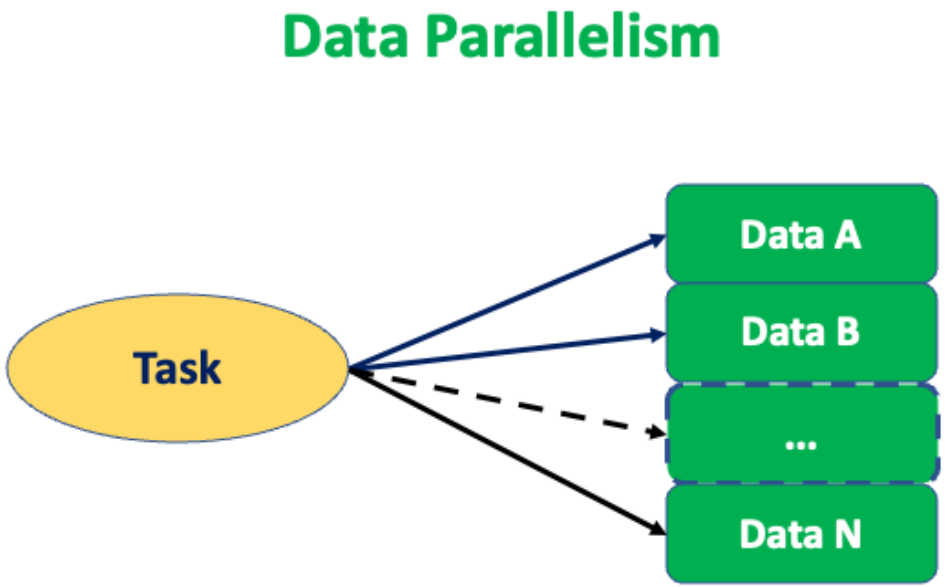
Part B: **How to Parallelize a Code** (75 min)

Hello, Parallel World!
- Parallel regions
- Compilation and execution

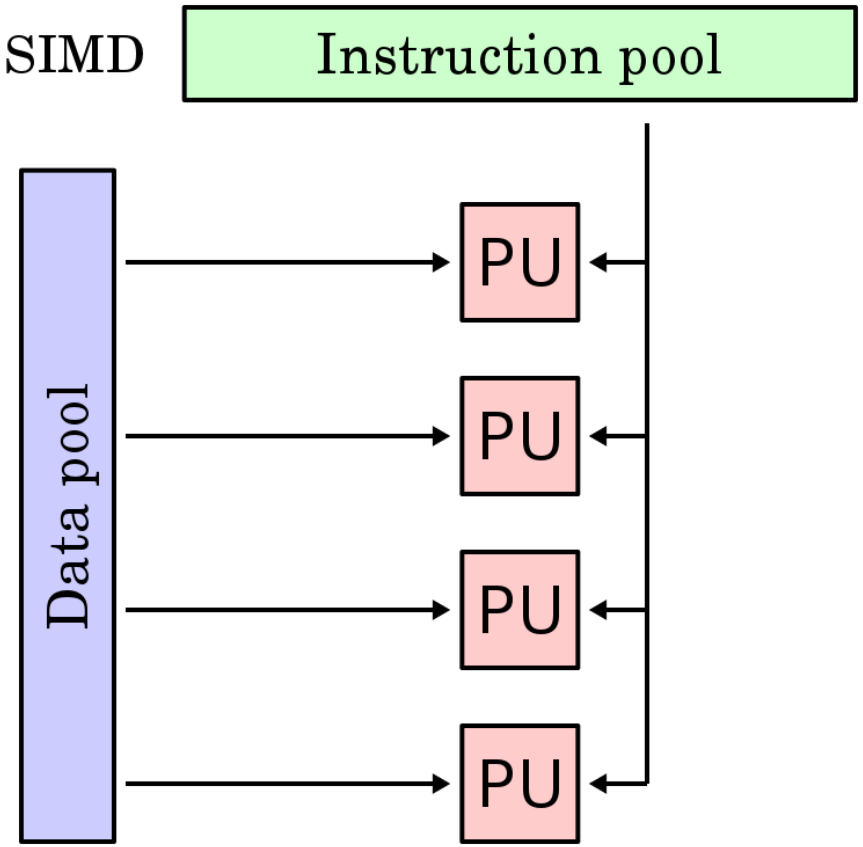
Task Parallelism
- Parallel sections
- Measuring performance

Data Parallelism
- Parallel loops
- Performance and scaling

Data parallelism



Execute the same task over different chunks of data



OpenMP parallel loops: syntax

C/C++

```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp for
10     for(int i = 0; i < N; i++)
11     {
12         ... task ...
13     }
14 }
15 .
16 .
17 ... serial code ...
18 .
19 .
```

Fortran

```
1  use :: omp_lib
2  .
3  .
4  ... serial code ...
5  .
6  .
7  !$omp parallel
8      !$omp do
9      do i = 1, N
10         .
11         ... task ...
12         .
13     enddo
14     !$omp end do
15 !$omp end parallel
16 .
17 .
18 ... serial code ...
19 .
20 .
```

OpenMP parallel loops: syntax

OpenMP library header

C/C++

```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp for
10     for(int i = 0; i < N; i++)
11     {
12         ... task ...
13     }
14 }
15 .
16 .
17 ... serial code ...
18 .
19 .
```

Fortran

```
1  use :: omp_lib
2  .
3  .
4  ... serial code ...
5  .
6  .
7  !$omp parallel
8      !$omp do
9      do i = 1, N
10         .
11         ... task ...
12         .
13     enddo
14     !$omp end do
15 !$omp end parallel
16 .
17 .
18 ... serial code ...
19 .
20 .
```

OpenMP parallel loops: syntax

C/C++

OpenMP library header

OpenMP parallel directives

```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp for
10     for(int i = 0; i < N; i++)
11     {
12         ... task ...
13     }
14 }
15 .
16 .
17 ... serial code ...
18 .
19 .
```

Fortran

```
1  use :: omp_lib
2  .
3  .
4  ... serial code ...
5  .
6  .
7  !$omp parallel
8      !$omp do
9      do i = 1, N
10         .
11         ... task ...
12         .
13     enddo
14     !$omp end do
15 !$omp end parallel
16 .
17 .
18 ... serial code ...
19 .
20 .
```

OpenMP parallel loops: syntax

C/C++

```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9     #pragma omp for
10    for(int i = 0; i < N; i++)
11    {
12        ... task ...
13    }
14 }
15 .
16 .
17 ... serial code ...
18 .
19 .
```

OpenMP library header

OpenMP parallel directives

Fortran

```
1  use :: omp_lib
2  .
3  .
4  ... serial code ...
5  .
6  .
7  !$omp parallel
8     !$omp do
9     do i = 1, N
10        .
11        ... task ...
12        .
13    enddo
14    !$omp end do
15 !$omp end parallel
16 .
17 .
18 ... serial code ...
19 .
20 .
```

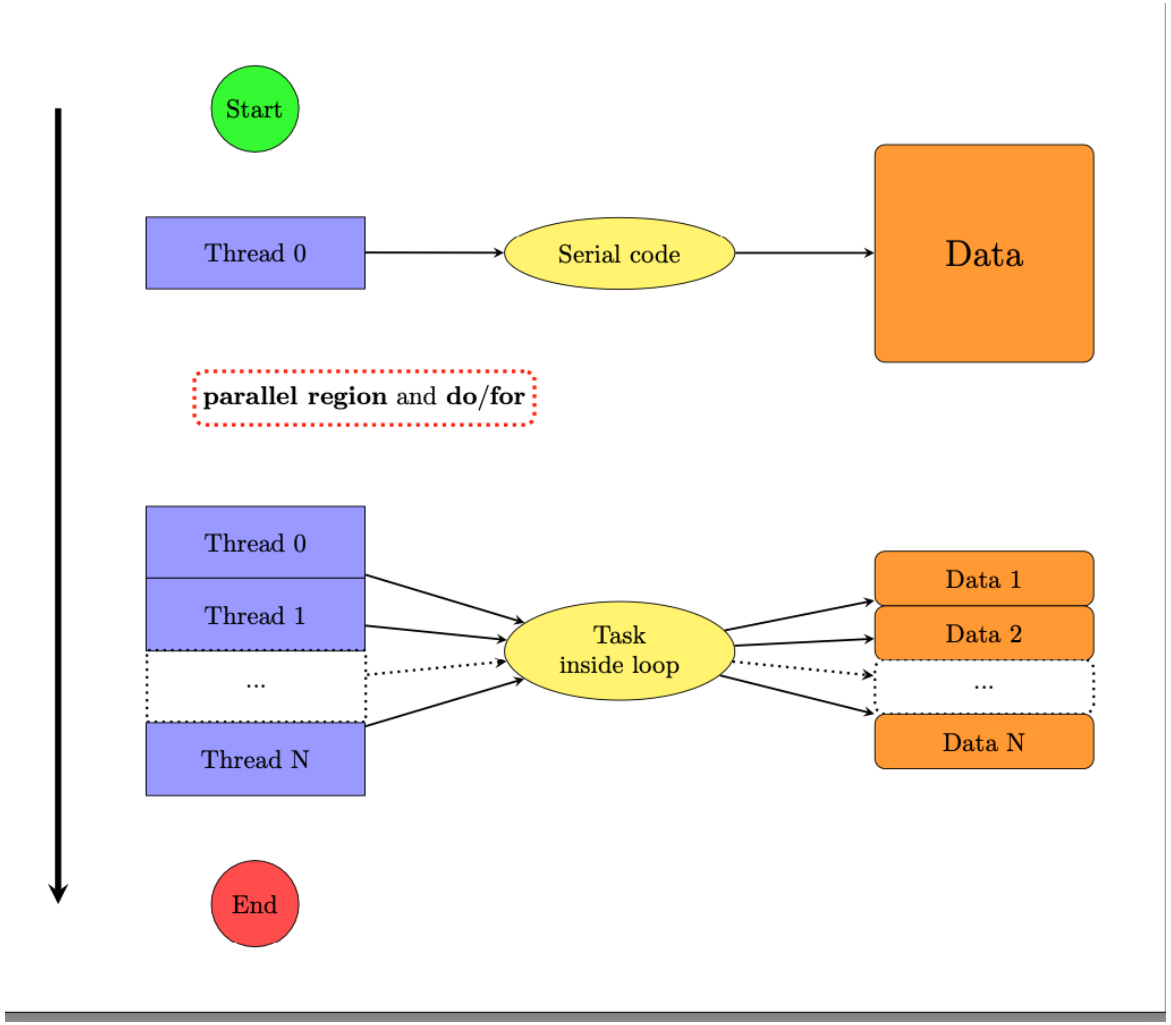
OpenMP do/for

Loop implementation must follow!

OpenMP parallel loops: workflow

C/C++

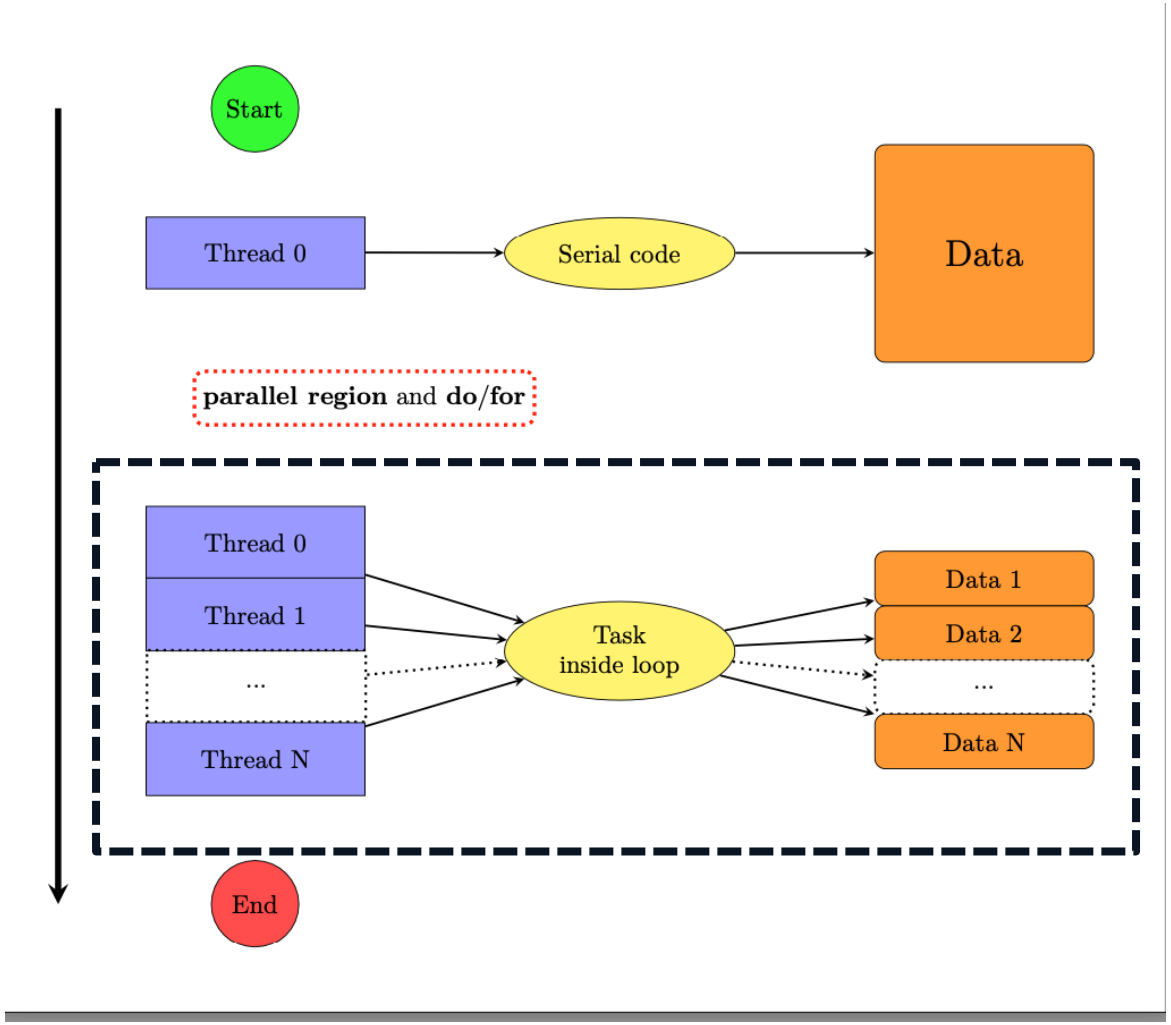
```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp for
10     for(int i = 0; i < N; i++)
11     {
12         ... task ...
13     }
14 }
15 .
16 .
17 ... serial code ...
18 .
19 .
```



OpenMP parallel loops: workflow

C/C++

```
1  #include "omp.h"
2  .
3  .
4  ... serial code ...
5  .
6  .
7  #pragma omp parallel
8  {
9      #pragma omp for
10     for(int i = 0; i < N; i++)
11     {
12         ... task ...
13     }
14 }
15 .
16 .
17 ... serial code ...
18 .
19 .
```

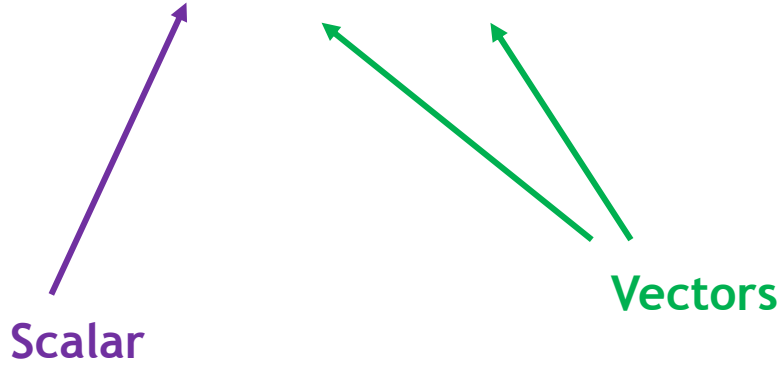


SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

SAXPY: **S**ingle-precision **A*X** **P**lus **Y**

$$y[i] \rightarrow a \cdot x[i] + y[i]$$



SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29
30  #define ORD 1 << 27 // size of array
31
32  using namespace std;
33
34  void saxpy(float a, const vector<float> &x, vector<float> &y);
35
```

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

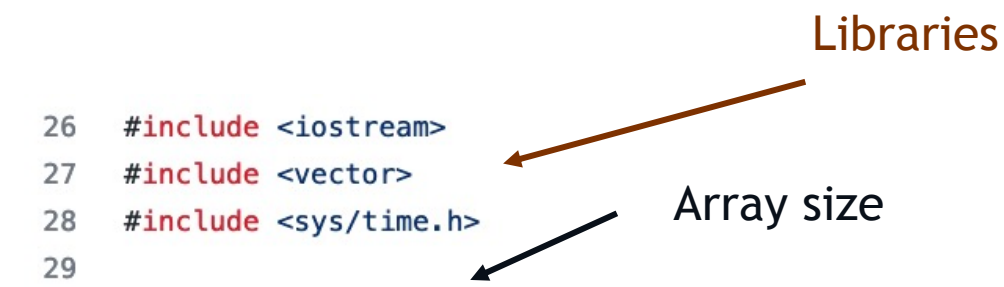
Libraries

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29
30  #define ORD 1 << 27 // size of array
31
32  using namespace std;
33
34  void saxpy(float a, const vector<float> &x, vector<float> &y);
35
```

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp




```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29
30  #define ORD 1 << 27 // size of array
31
32  using namespace std;
33
34  void saxpy(float a, const vector<float> &x, vector<float> &y);
35
```

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

Libraries

Array size

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29
30  #define ORD 1 << 27 // size of array
31
32  using namespace std;
33
34  void saxpy(float a, const vector<float> &x, vector<float> &y);
35
```

Function declaration

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

Libraries

Array size

Instantiate vectors and scalar

Function declaration

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29
30  #define ORD 1 << 27 // size of array
31
32  using namespace std;
33
34  void saxpy(float a, const vector<float> &x, vector<float> &y);
35
```

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

Libraries

Array size

Instantiate vectors and scalar

Function declaration

Perform SAXPY

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29
30  #define ORD 1 << 27 // size of array
31
32  using namespace std;
33
34  void saxpy(float a, const vector<float> &x, vector<float> &y);
35
```

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

Function implementation

```
57
58  void saxpy(float a, const vector<float> &x, vector<float> &y)
59  {
60      for (int i = 0; i < x.size(); i++)
61      {
62          y[i] = a * x[i] + y[i];
63      }
64  }
```

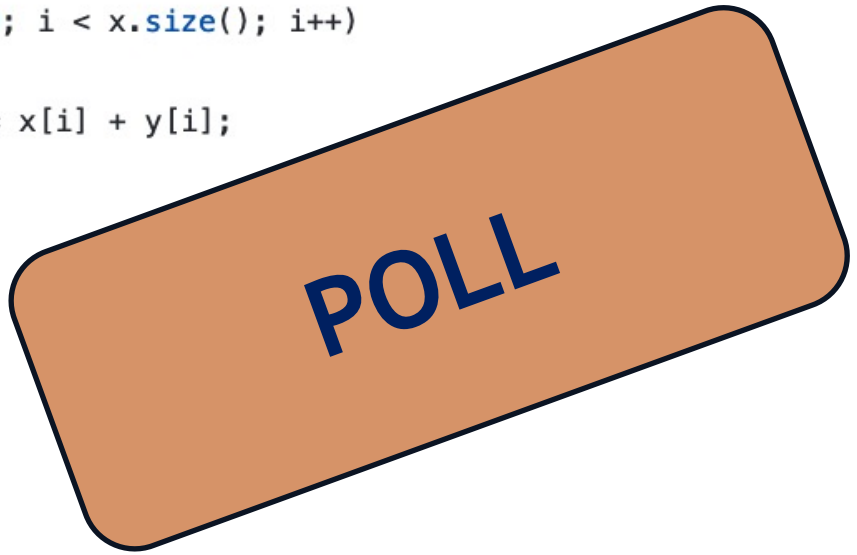
SAXPY: serial code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/serial/saxpy_serial.cpp

```
36  int main()
37  {
38      // instantiate vectors and attribute values
39      vector<float> x, y;
40      float r, elmnt;
41      srand(time(NULL));
42      for (int i = 0; i < ORD; i++)
43      {
44          r = rand() % RAND_MAX;
45          elmnt = float(r) / (RAND_MAX);
46          x.push_back(elmnt); // x is noise
47          y.push_back(1.0);   // y is just ones
48      }
49
50      float a = 2.0; // scalar multiplier (y -> a*x + y)
51
52      // now call saxpy
53      saxpy(a, x, y);
54
55      return 0;
56  }
```

Function implementation

```
57
58  void saxpy(float a, const vector<float> &x, vector<float> &y)
59  {
60      for (int i = 0; i < x.size(); i++)
61      {
62          y[i] = a * x[i] + y[i];
63      }
64  }
```



SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29  #include "omp.h"
30
31  #define ORD 1 << 27 // size of array
32
33  using namespace std;
34
35  void saxpy(float a, const vector<float> &x, vector<float> &y);
36
```

SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29  #include "omp.h"
30
31  #define ORD 1 << 27 // size of array
32
33  using namespace std;
34
35  void saxpy(float a, const vector<float> &x, vector<float> &y);
36
```

SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29  #include "omp.h"
30
31  #define ORD 1 << 27 // size of array
32
33  using namespace std;
34
35  void saxpy(float a, const vector<float> &x, vector<float> &y);
36
```

OpenMP Library

```
37  int main()
38  {
39      // instantiate vectors and attribute values
40      vector<float> x, y;
41      float r, elmnt;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = float(r) / (RAND_MAX);
47          x.push_back(elmnt); // x is noise
48          y.push_back(1.0);   // y is just ones
49      }
50
51      float a = 2.0; // scalar multiplier (y -> a*x + y)
```

SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

```
26  #include <iostream>
27  #include <vector>
28  #include <sys/time.h>
29  #include "omp.h"
30
31  #define ORD 1 << 27 // size of array
32
33  using namespace std;
34
35  void saxpy(float a, const vector<float> &x, vector<float> &y);
36
```

OpenMP Library

Instantiate vectors and scalar

```
37  int main()
38  {
39      // instantiate vectors and attribute values
40      vector<float> x, y;
41      float r, elmnt;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = float(r) / (RAND_MAX);
47          x.push_back(elmnt); // x is noise
48          y.push_back(1.0);   // y is just ones
49      }
50
51      float a = 2.0; // scalar multiplier (y -> a*x + y)
```

SAXPY: parallel code

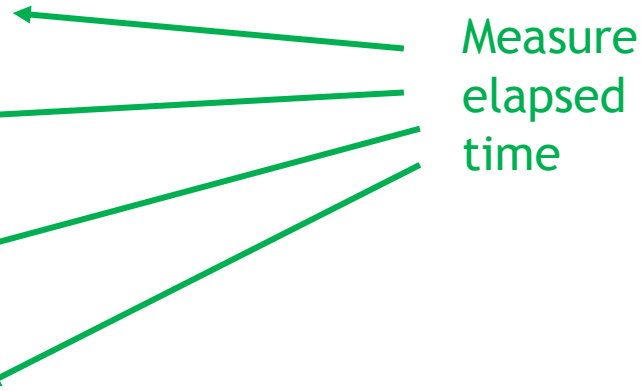
Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

```
53      // stopwatch variables
54      struct timeval start_time, stop_time, elapsed_time;
55
56      // now call saxpy
57      gettimeofday(&start_time, NULL);
58      #pragma omp parallel
59      {
60          saxpy(a, x, y);
61      }
62      gettimeofday(&stop_time, NULL);
63
64      timersub(&stop_time, &start_time, &elapsed_time);
65      cout << "Elapsed time (s): " << elapsed_time.tv_sec + elapsed_time.tv_usec / 1000000.0 << endl;
66
67      return 0;
68  }
```

SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

```
53      // stopwatch variables
54      struct timeval start_time, stop_time, elapsed_time;
55
56      // now call saxpy
57      gettimeofday(&start_time, NULL);
58      #pragma omp parallel
59      {
60          saxpy(a, x, y);
61      }
62      gettimeofday(&stop_time, NULL);
63
64      timersub(&stop_time, &start_time, &elapsed_time);
65      cout << "Elapsed time (s): " << elapsed_time.tv_sec + elapsed_time.tv_usec / 1000000.0 << endl;
66
67      return 0;
68  }
```



Measure elapsed time

SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

```
53      // stopwatch variables
54      struct timeval start_time, stop_time, elapsed_time;
55
56      // now call saxpy
57      gettimeofday(&start_time, NULL);
58      #pragma omp parallel
59      {
60          saxpy(a, x, y);
61      }
62      gettimeofday(&stop_time, NULL);
63
64      timersub(&stop_time, &start_time, &elapsed_time);
65      cout << "Elapsed time (s): " << elapsed_time.tv_sec + elapsed_time.tv_usec / 1000000.0 << endl;
66
67      return 0;
68  }
```

Enclose function call with OpenMP **parallel** directive

SAXPY: parallel code

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp

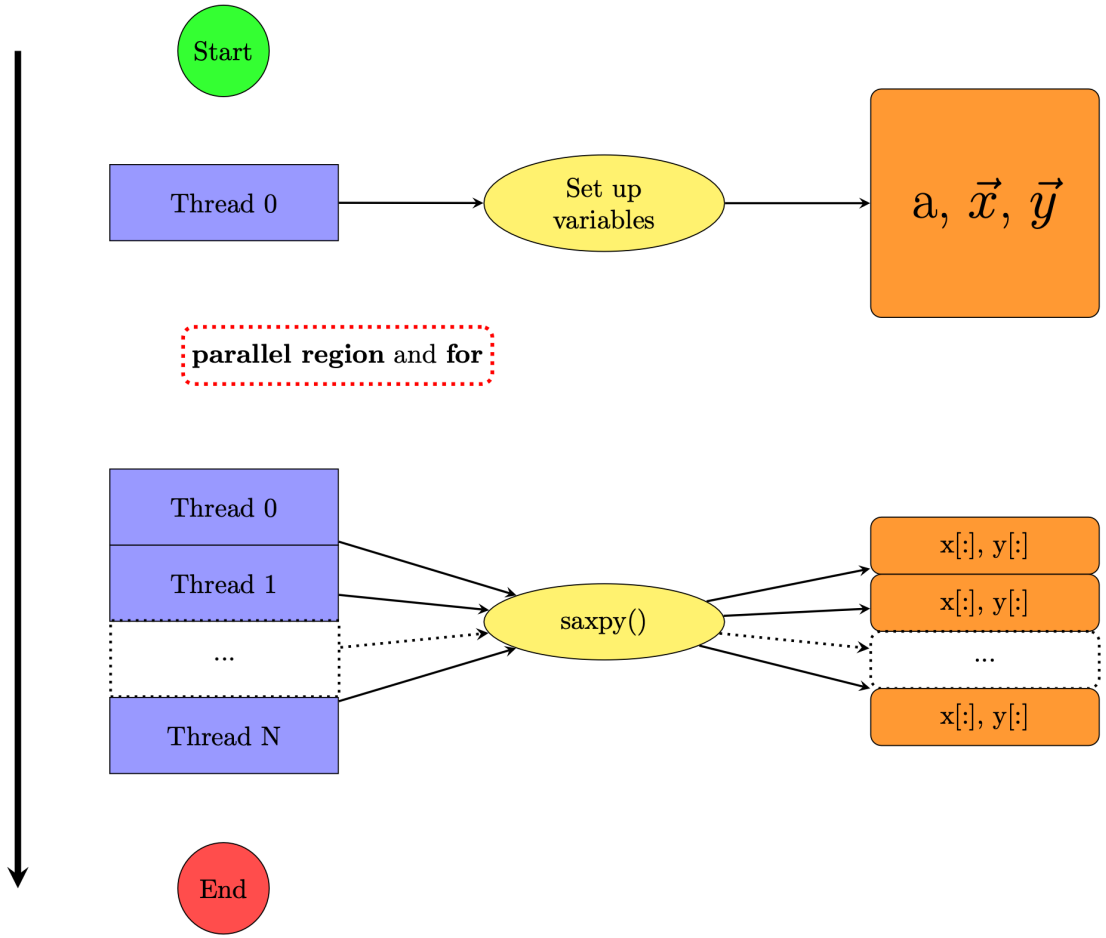
```
53      // stopwatch variables
54      struct timeval start_time, stop_time, elapsed_time;
55
56      // now call saxpy
57      gettimeofday(&start_time, NULL);
58      #pragma omp parallel
59      {
60          saxpy(a, x, y);
61      }
62      gettimeofday(&stop_time, NULL);
63
64      timersub(&stop_time, &start_time, &elapsed_time);
65      cout << "Elapsed time (s): " << elapsed_time.tv_sec + elapsed_time.tv_usec / 1000000.0 << endl;
66
67      return 0;
68  }
```

Insert OpenMP **for** directive at function level, right before loop

```
70  void saxpy(float a, const vector<float> &x, vector<float> &y)
71  {
72      #pragma omp for
73      for (int i = 0; i < x.size(); i++)
74      {
75          y[i] = a * x[i] + y[i];
76      }
77  }
```

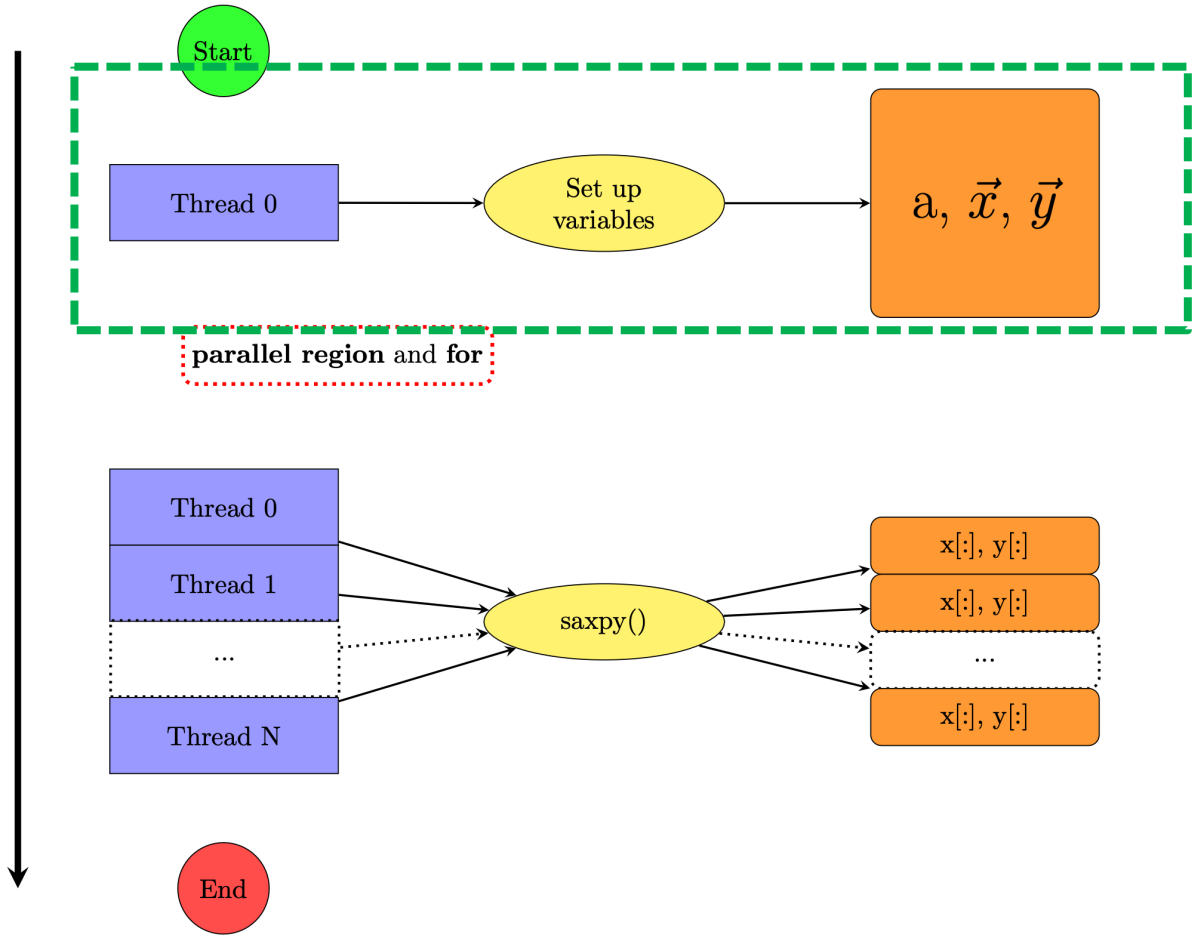
SAXPY: workflow diagram

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp



SAXPY: workflow diagram

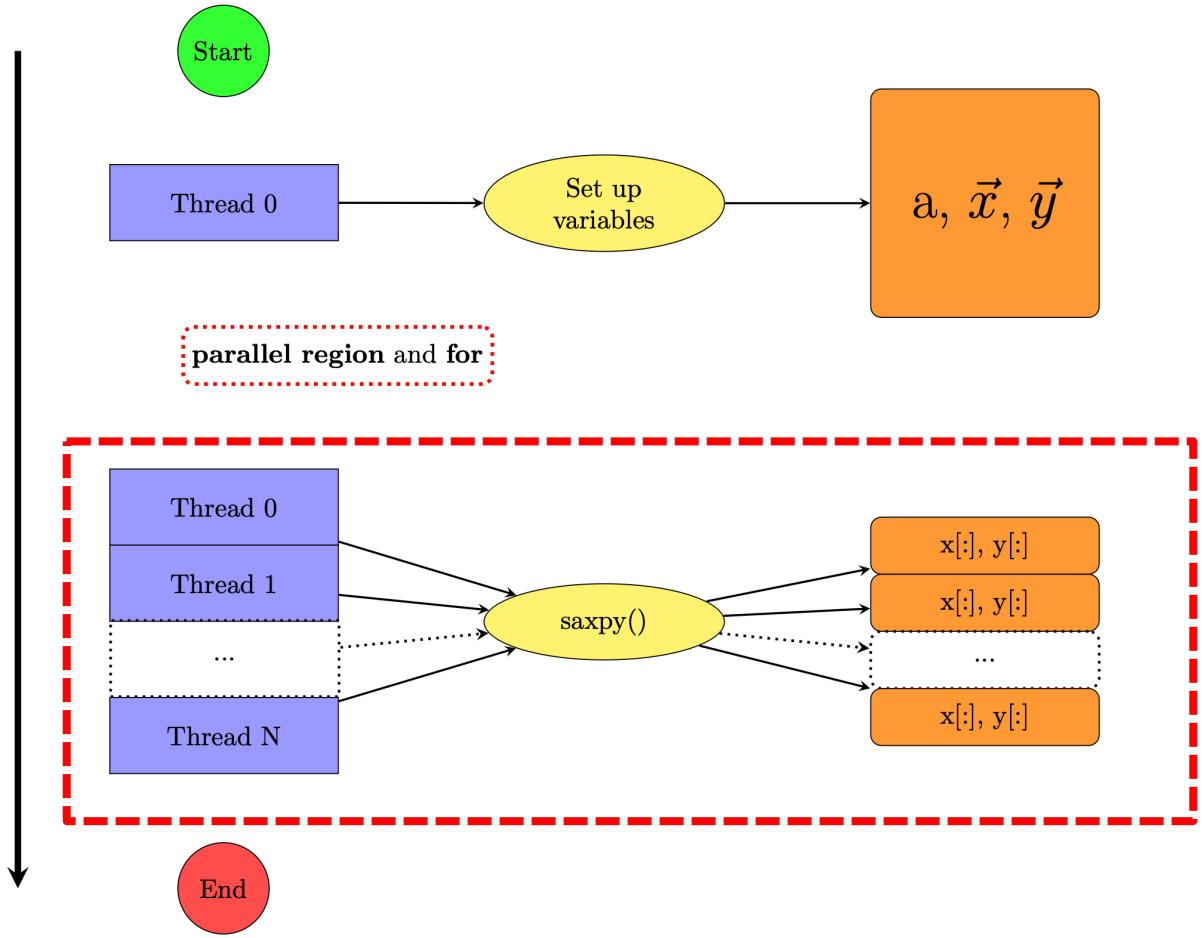
Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp




```
37  int main()
38  {
39      // instantiate vectors and attribute values
40      vector<float> x, y;
41      float r, elmnt;
42      srand(time(NULL));
43      for (int i = 0; i < ORD; i++)
44      {
45          r = rand() % RAND_MAX;
46          elmnt = float(r) / (RAND_MAX);
47          x.push_back(elmnt); // x is noise
48          y.push_back(1.0);   // y is just ones
49      }
50
51      float a = 2.0; // scalar multiplier (y -> a*x + y)
```

SAXPY: workflow diagram

Find the code here: https://github.com/babreu-ncsa/IntroToPC/blob/uiuc-icc/saxpy/omp/saxpy_omp.cpp




```
56        // now call saxpy
57        gettimeofday(&start_time, NULL);
58        #pragma omp parallel
59        {
60            saxpy(a, x, y);
61        }
62        gettimeofday(&stop_time, NULL);
63
```

```
70  void saxpy(float a, const vector<float> &x, vector<float> &y)
71  {
72      #pragma omp for
73      for (int i = 0; i < x.size(); i++)
74      {
75          y[i] = a * x[i] + y[i];
76      }
77  }
```

SAXPY: parallelized loop

```
70  void saxpy(float a, const vector<float> &x, vector<float> &y)
71  {
72      #pragma omp for
73      for (int i = 0; i < x.size(); i++)
74      {
75          y[i] = a * x[i] + y[i];
76      }
77  }
```

SAXPY: parallelized loop

```
70  void saxpy(float a, const vector<float> &x, vector<float> &y)
71  {
72      #pragma omp for
73      for (int i = 0; i < x.size(); i++)
74      {
75          y[i] = a * x[i] + y[i];
76      }
77  }
```

2^{27} = 134,217,728 iterations

SAXPY: parallelized loop

```
70  void saxpy(float a, const vector<float> &x, vector<float> &y)
71  {
72      #pragma omp for
73      for (int i = 0; i < x.size(); i++)
74      {
75          y[i] = a * x[i] + y[i];
76      }
77  }
```

2^{27} = 134,217,728 iterations

`export OMP_NUM_THREADS=N`

SAXPY: parallelized loop

```
70  void saxpy(float a, const vector<float> &x, vector<float> &y)
71  {
72      #pragma omp for
73      for (int i = 0; i < x.size(); i++)
74      {
75          y[i] = a * x[i] + y[i];
76      }
77  }
```

2^{27} = 134,217,728 iterations

```
export OMP_NUM_THREADS=N
```

Each loop gets a chunk of size

$\frac{2^{27}}{N}$ to work on

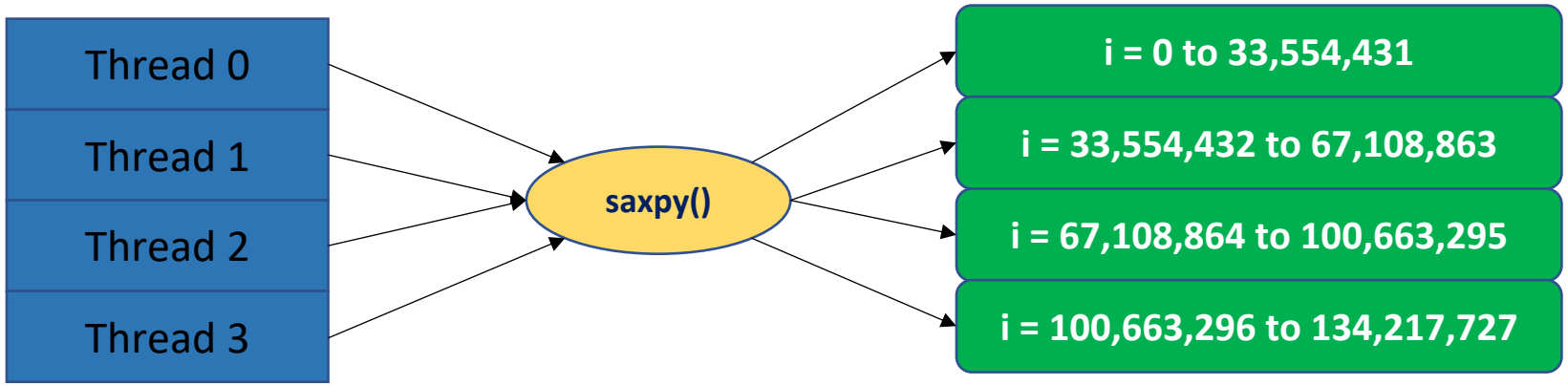
SAXPY: parallelized loop

```
70  void saxpy(float a, const vector<float> &x, vector<float> &y)
71  {
72      #pragma omp for
73      for (int i = 0; i < x.size(); i++)
74      {
75          y[i] = a * x[i] + y[i];
76      }
77  }
```

2^{27} = 134,217,728 iterations

`export OMP_NUM_THREADS=N`

Each loop gets a chunk of size $\frac{2^{27}}{N}$ to work on



OpenMP does **static scheduling** by default

SAXPY: compilation and execution

- **To compile:**

```
make
```

- **To run:**

```
sbatch saxpy.jobscript
```

- **To check status:**

```
squeue –u $USER
```

Use `cat` to check on your output, copy it and paste in the chat!

The output file should be named like:

saxpy-YourJobID.out

The tab button will help you find it!

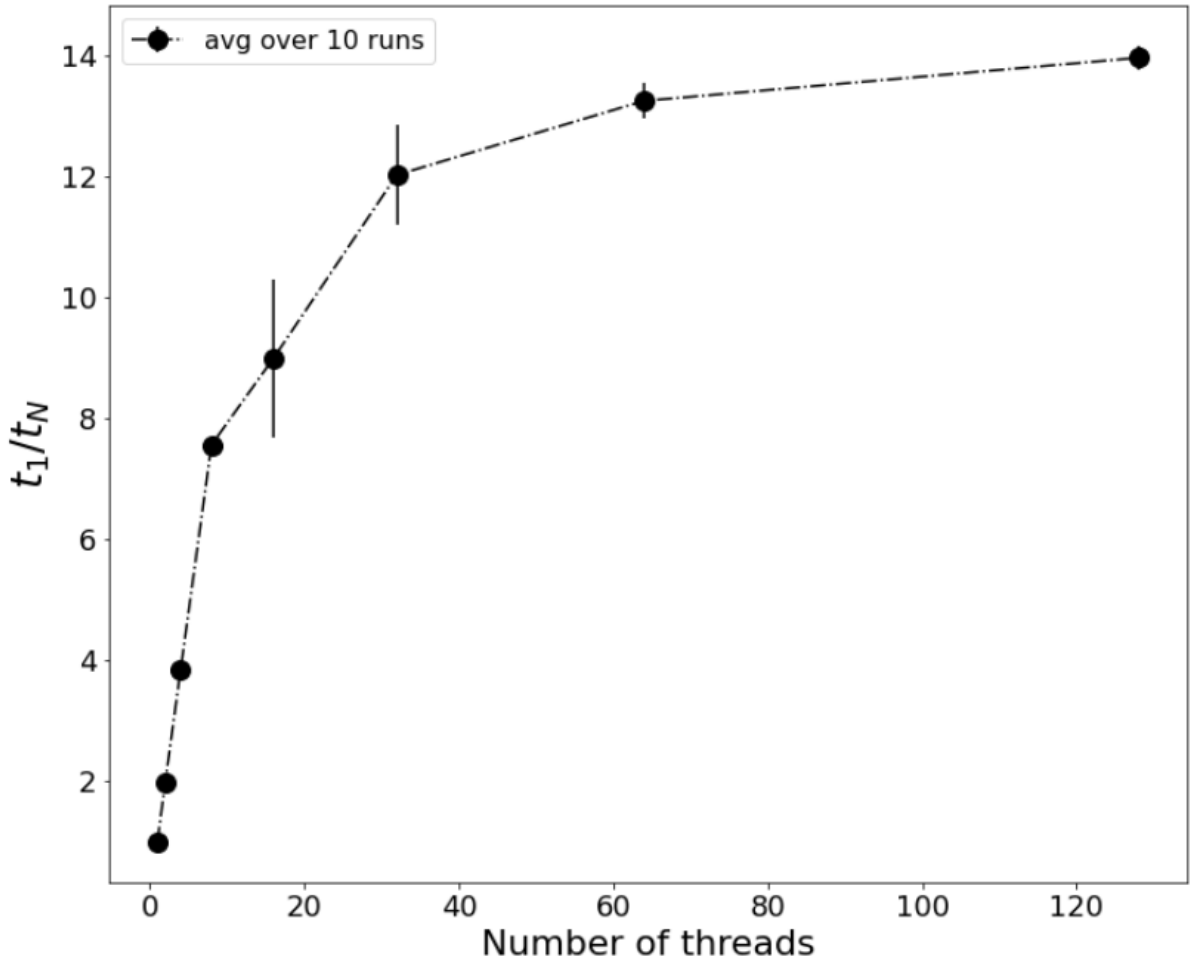
SAXPY: performance and scaling

Here are some results with 10 runs for each N

Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972

SAXPY: performance and scaling

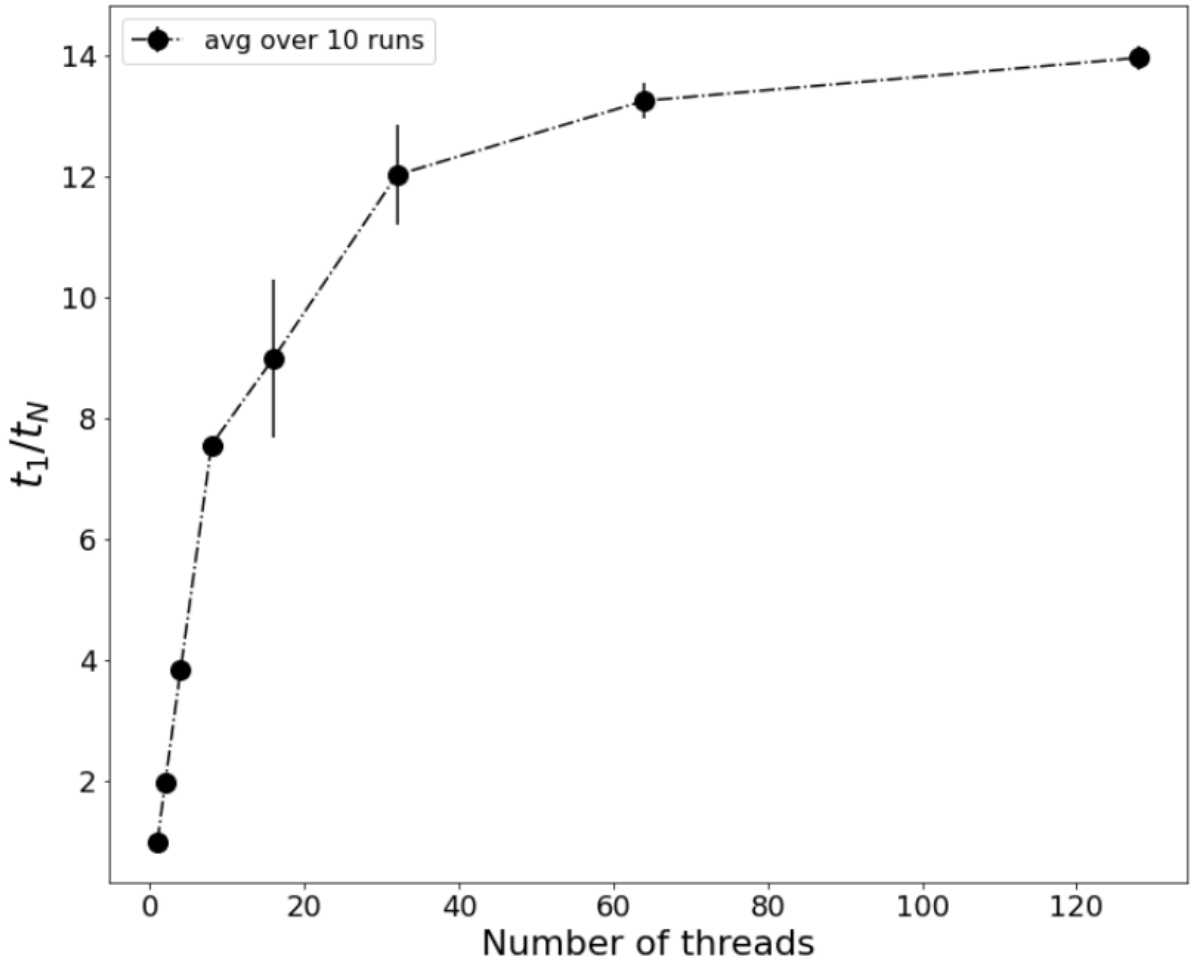
Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972



SAXPY: performance and scaling

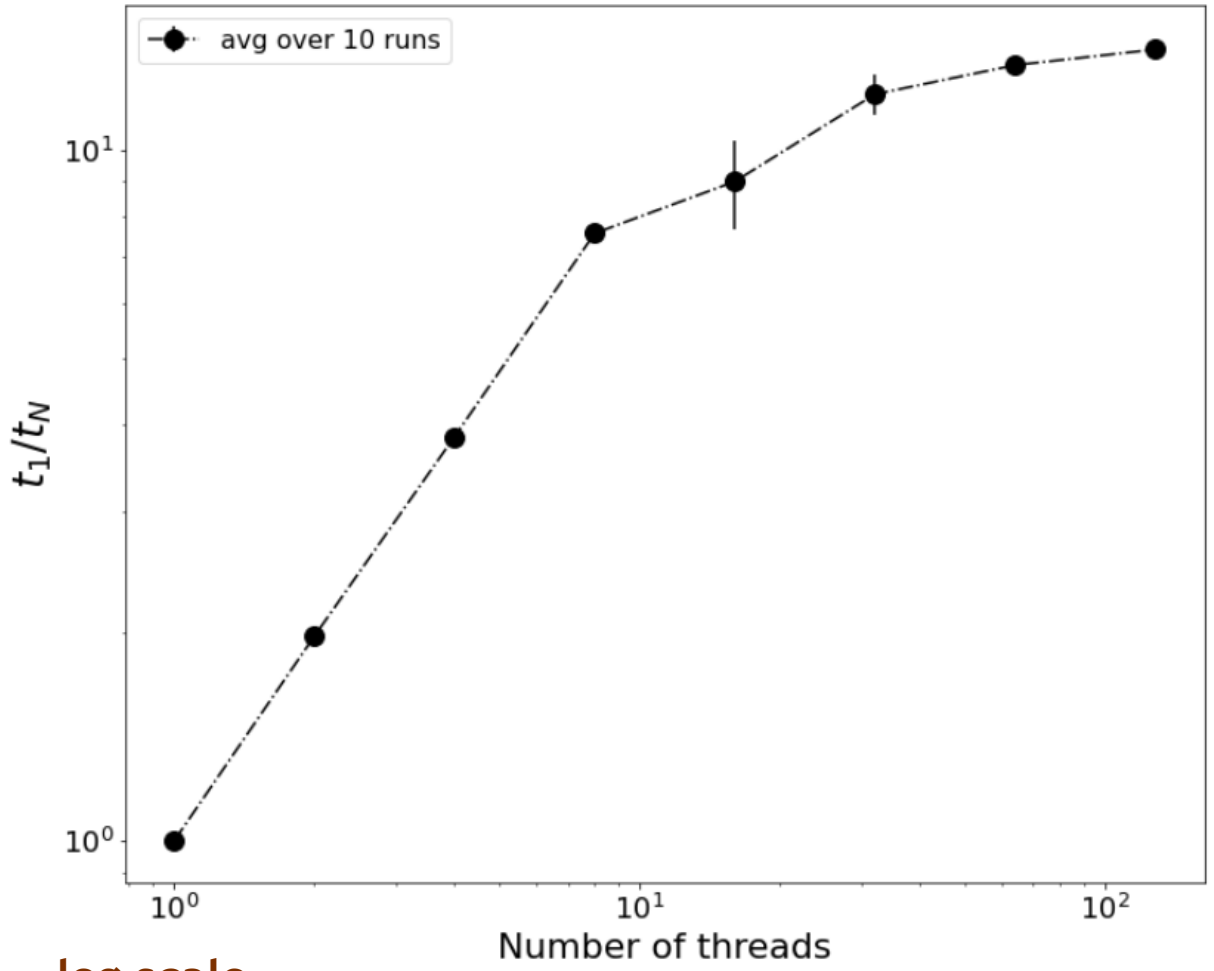
Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972

Exponential scale



SAXPY: performance and scaling

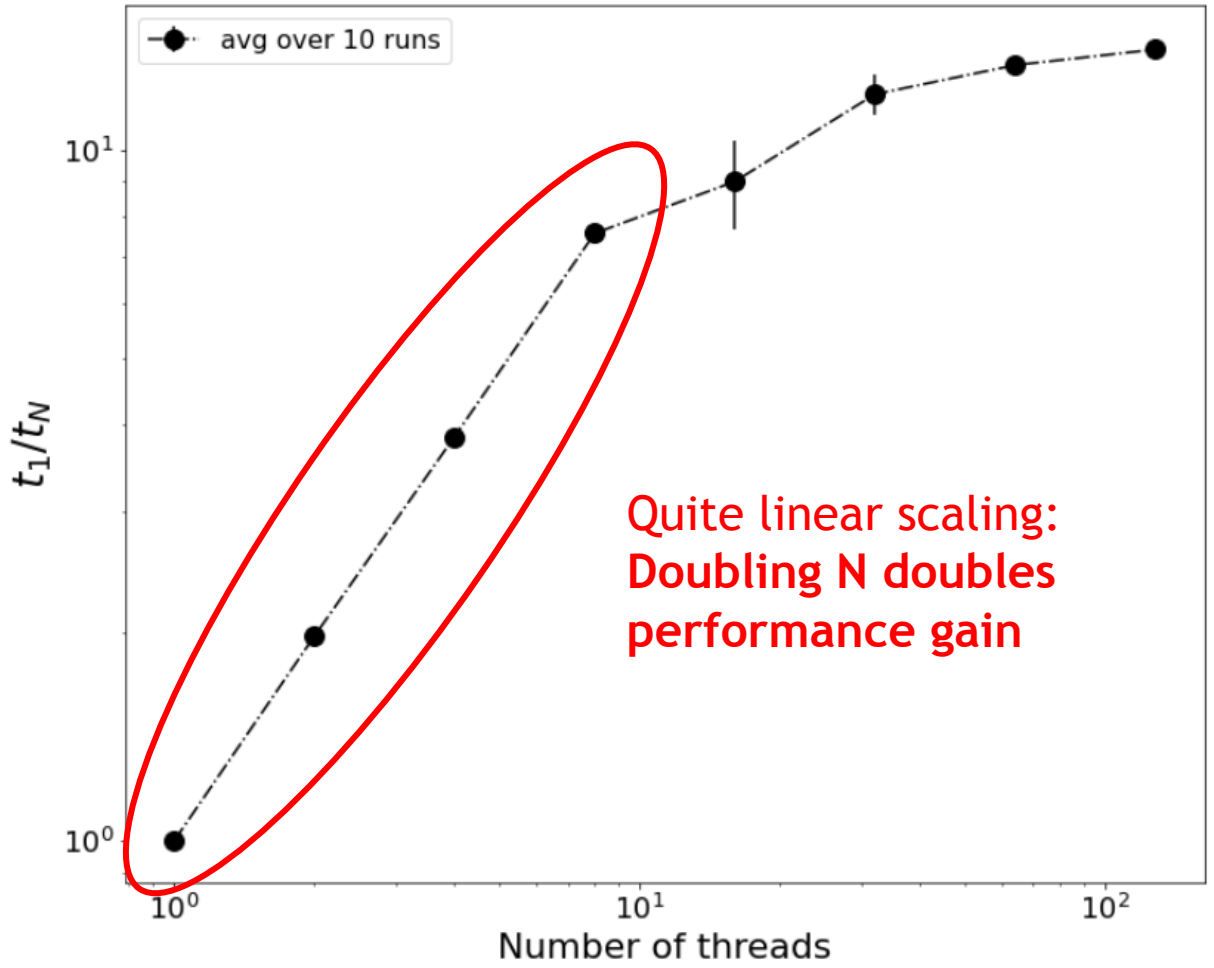
Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972



log scale

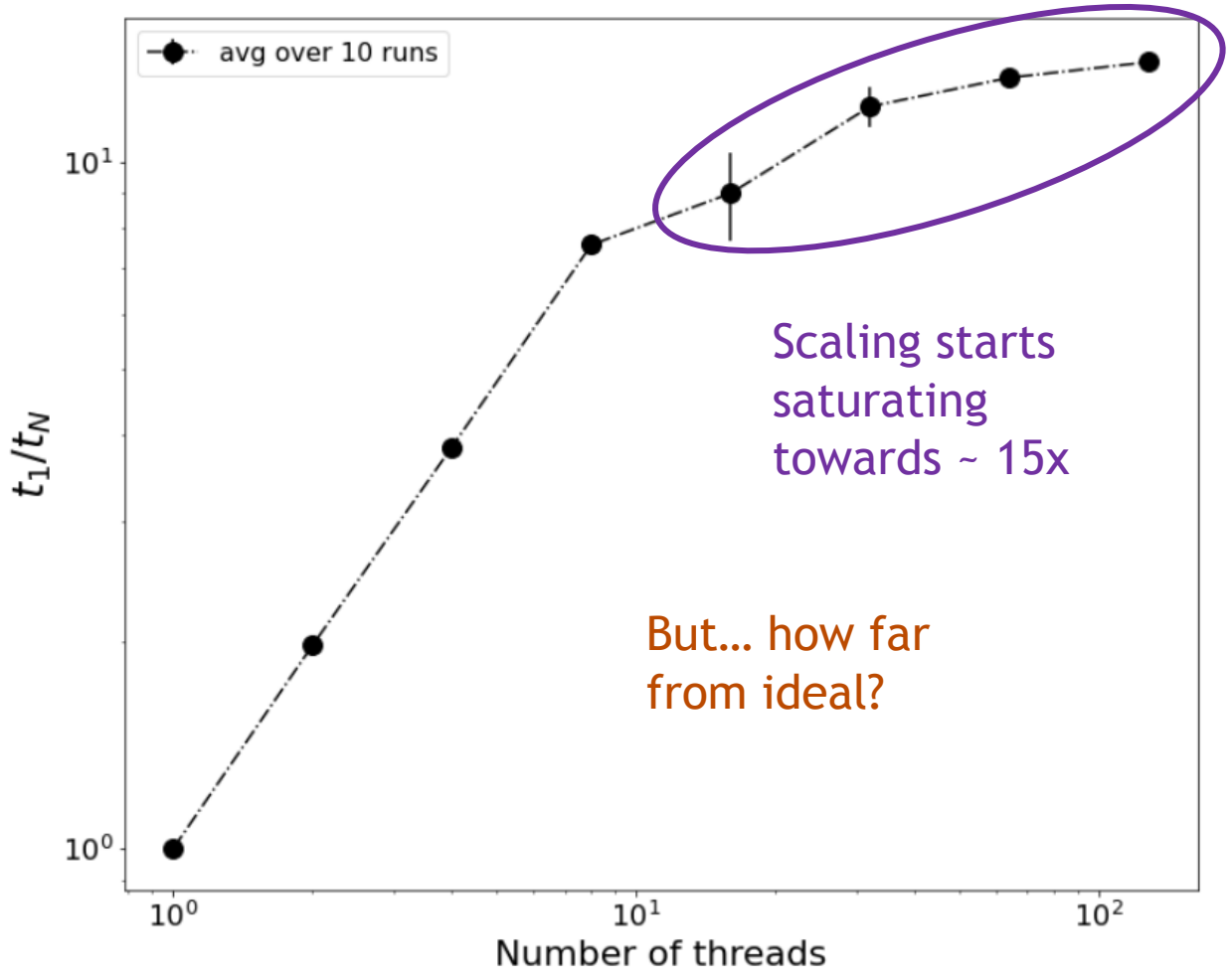
SAXPY: performance and scaling

Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972



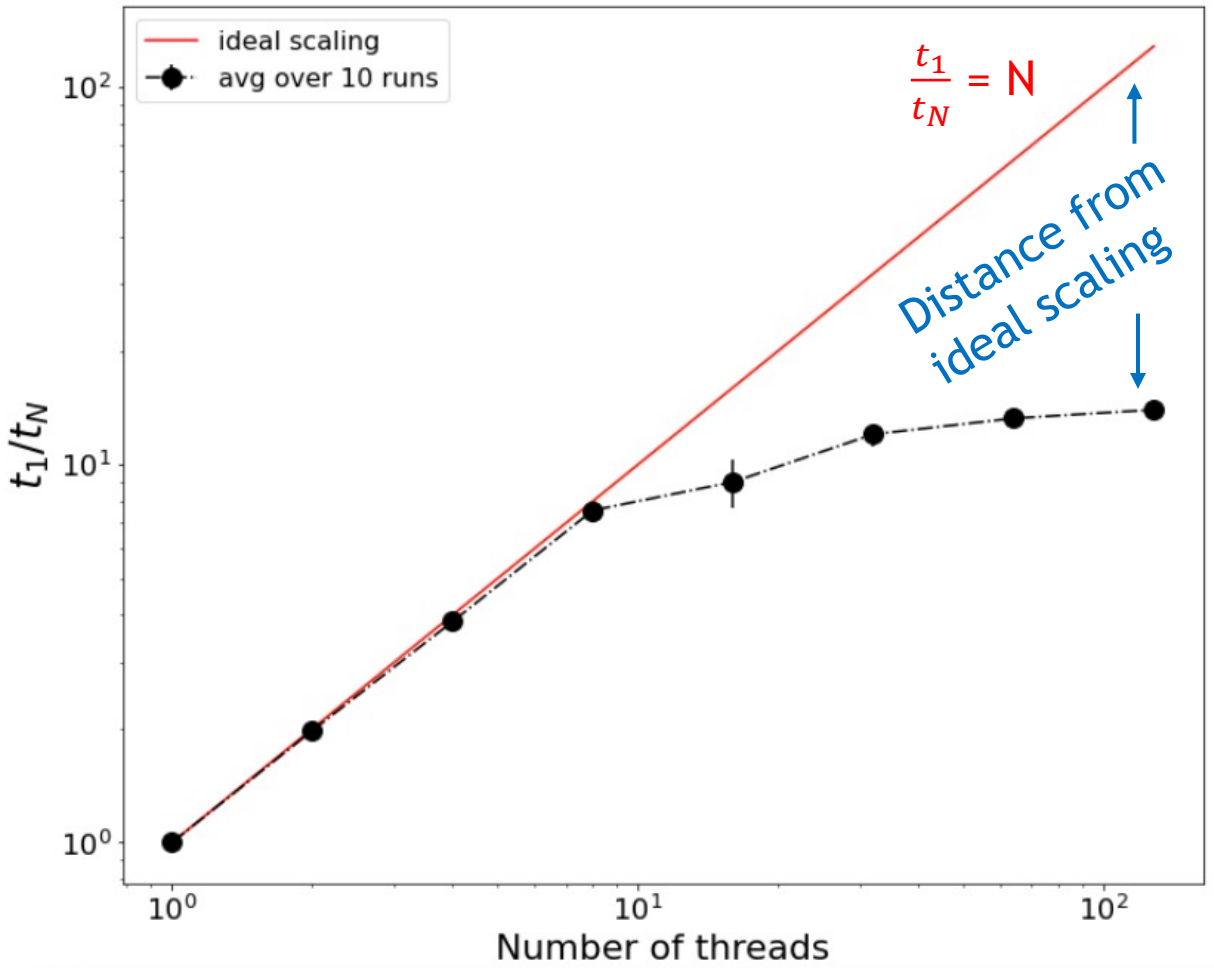
SAXPY: performance and scaling

Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972



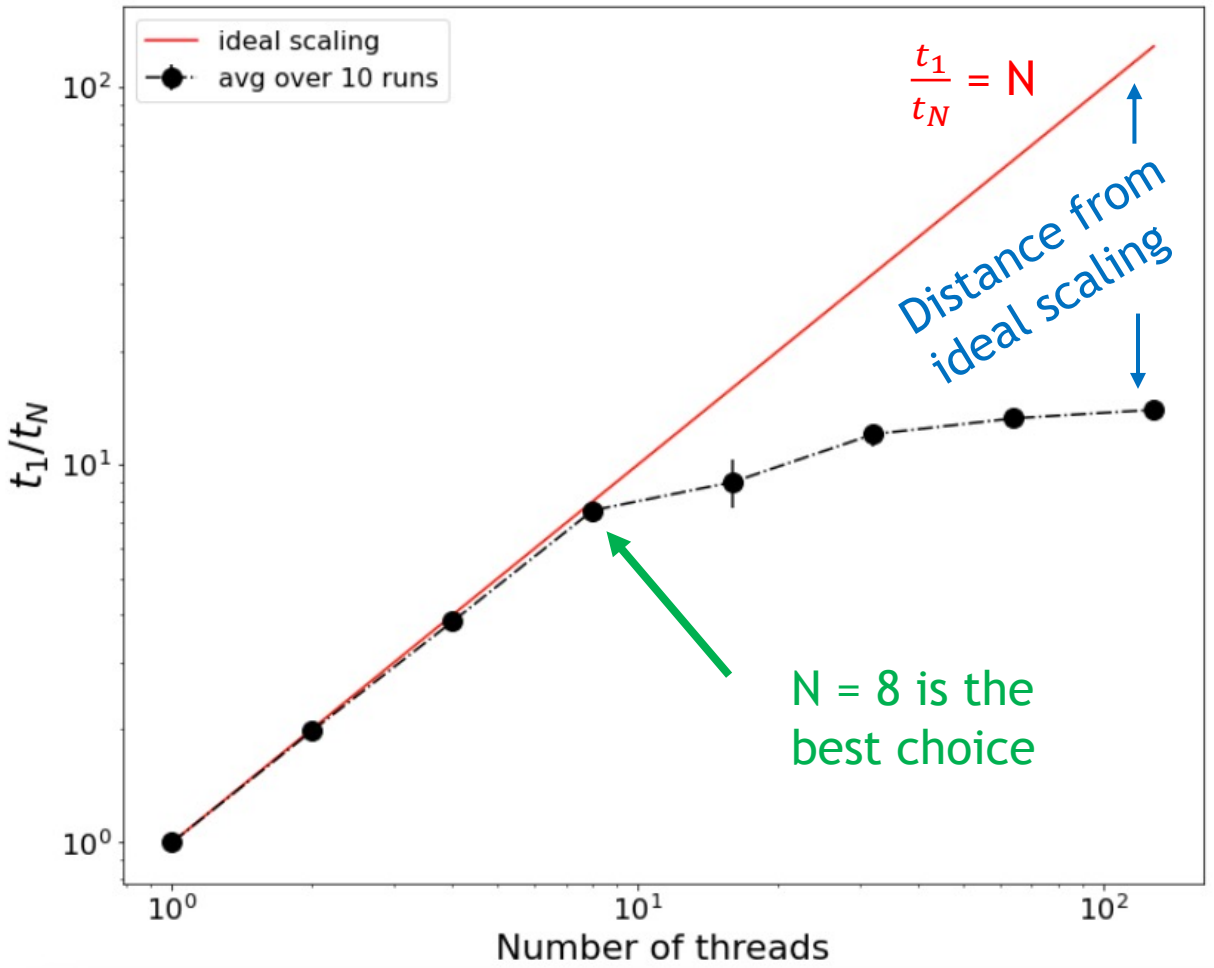
SAXPY: performance and scaling

Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972



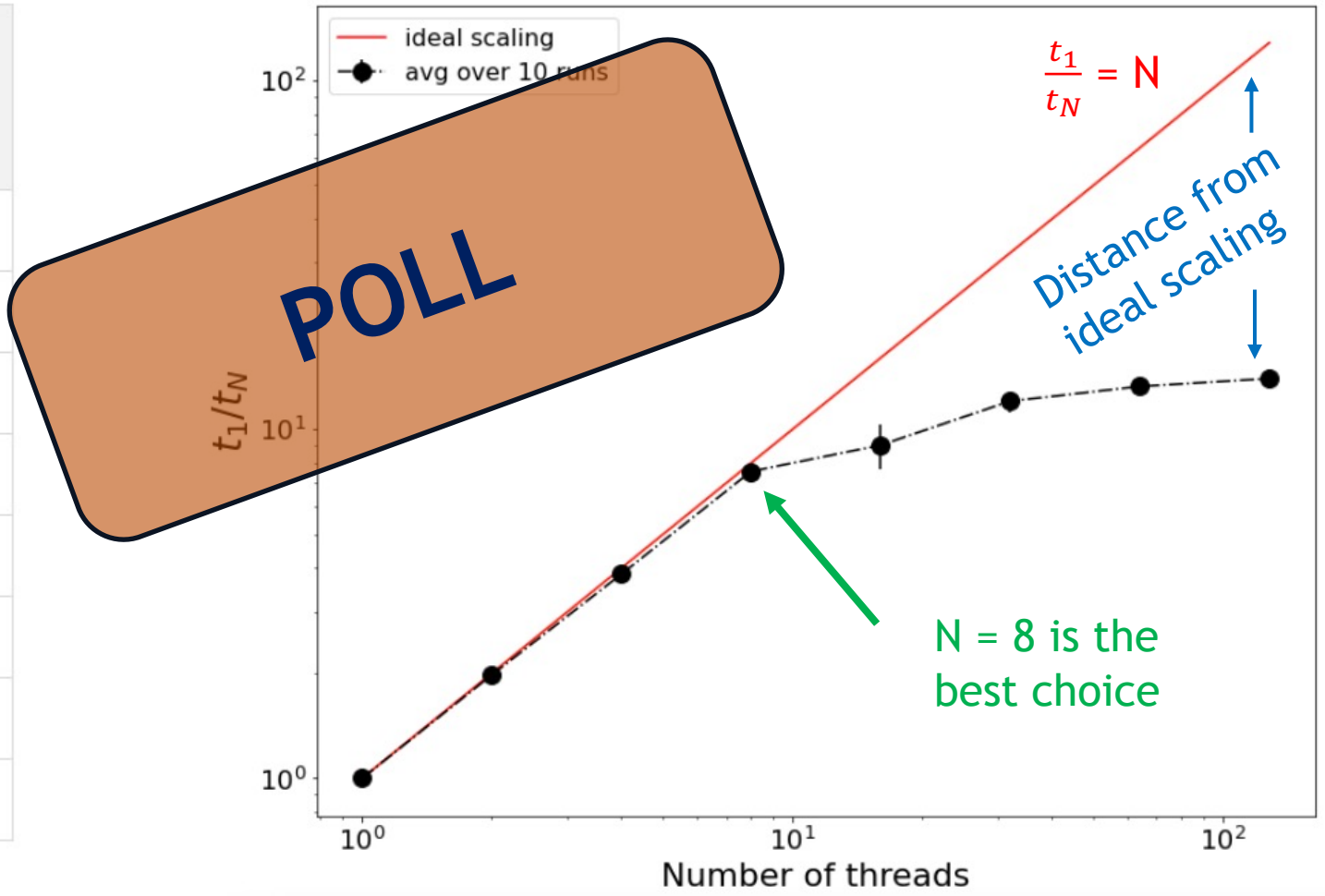
SAXPY: performance and scaling

Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972



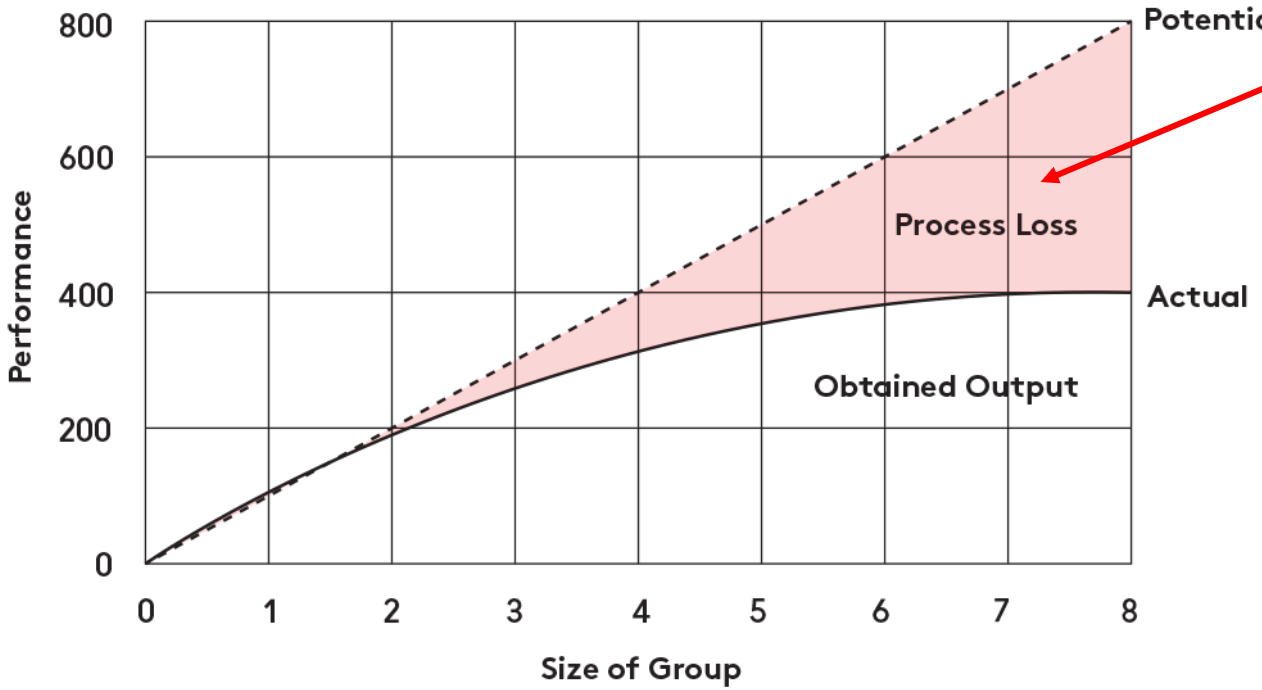
SAXPY: performance and scaling

Number of threads	Average elapsed time (s)	t_1 / t_N
1	0.7533	1.000
2	0.3810	1.977
4	0.1960	3.844
8	0.0995	7.569
16	0.0837	8.997
32	0.0626	12.035
64	0.0568	13.260
128	0.0539	13.972



SAXPY: performance analysis

Threads are not much different from people…



This is called the **Ringelmann effect**

Losses generally come from:

- More complex management requirements
- Increased communication requirements
- Task conflicts

In parallel computing language, this is called **parallel overhead**

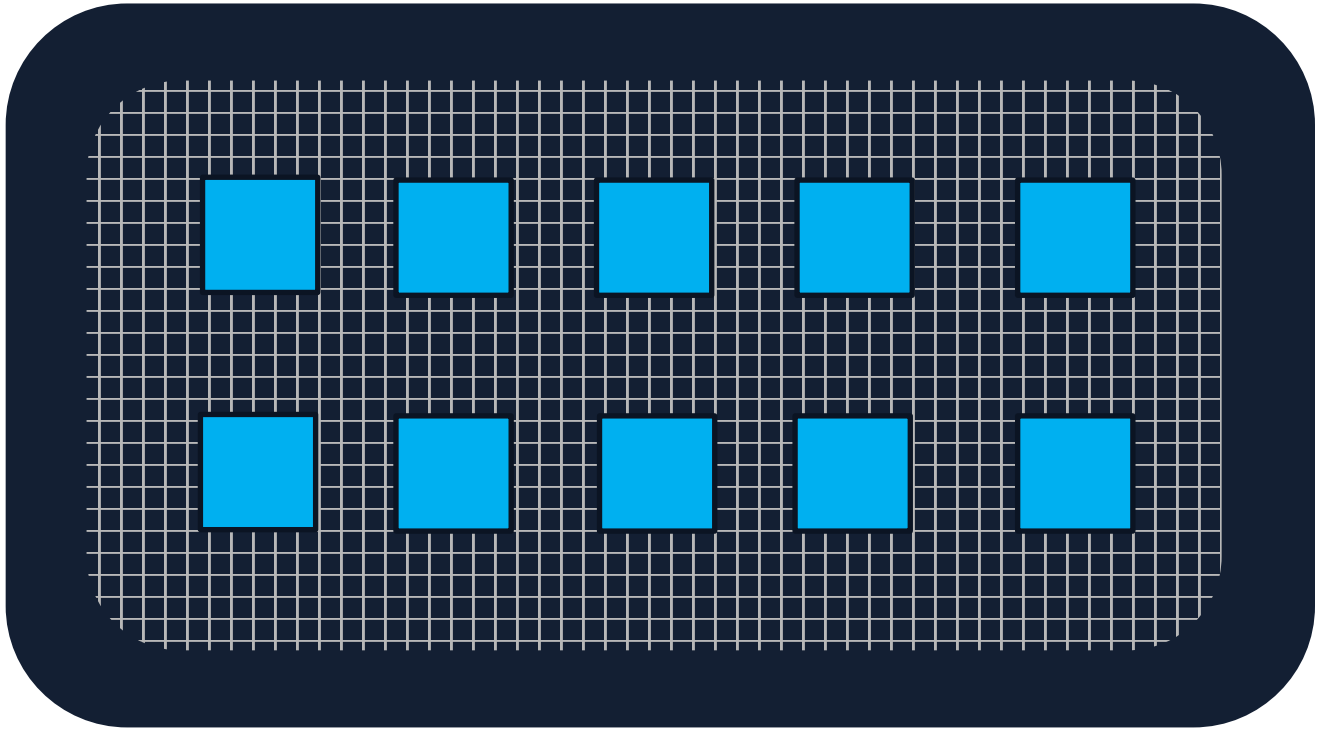
- Creating, distributing workloads and destroying a larger number of threads simply takes longer!

BUT…

SAXPY: performance analysis

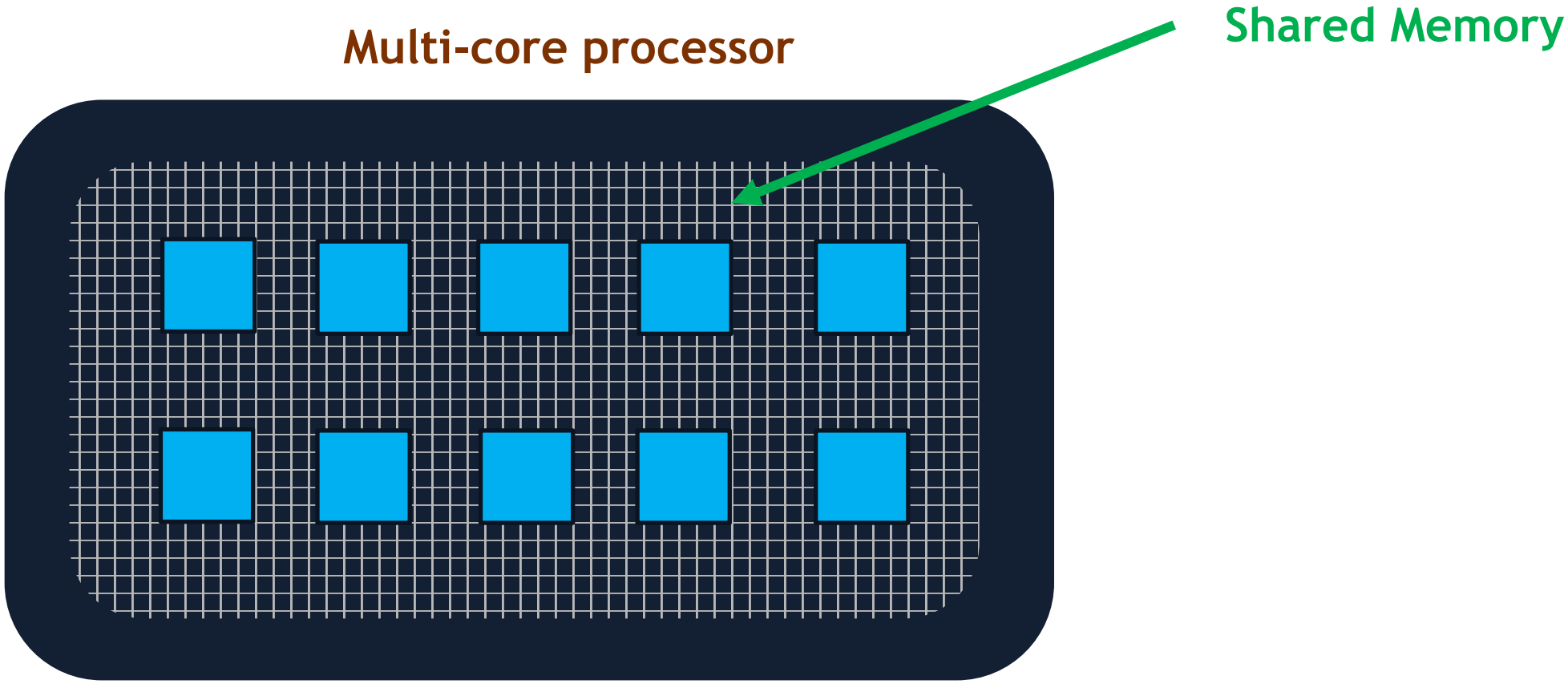
Parallel overhead is not that much of a problem here…

Multi-core processor



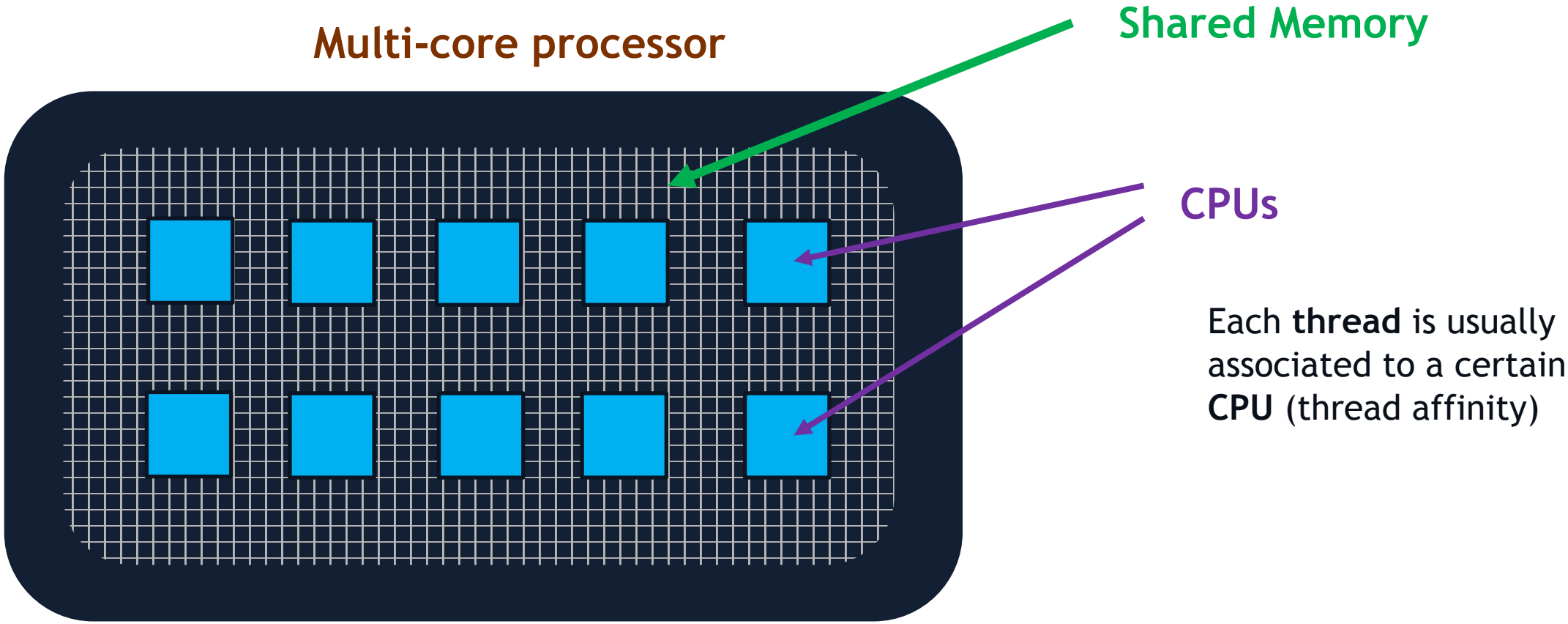
SAXPY: performance analysis

Parallel overhead is not that much of a problem here…



SAXPY: performance analysis

Parallel overhead is not that much of a problem here…

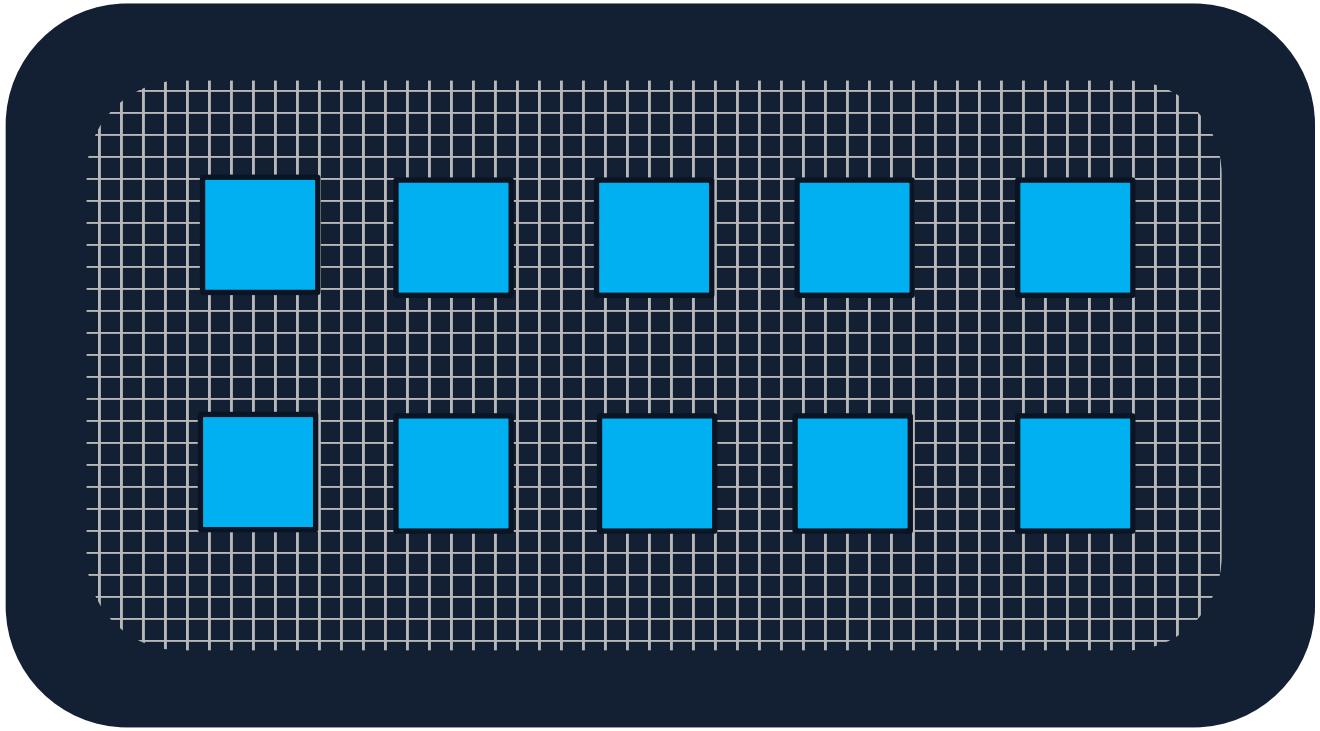


Each **thread** is usually associated to a certain **CPU** (thread affinity)

SAXPY: performance analysis

We start the code by allocating two vectors, **x** and **y**, with 2^{27} entries each…

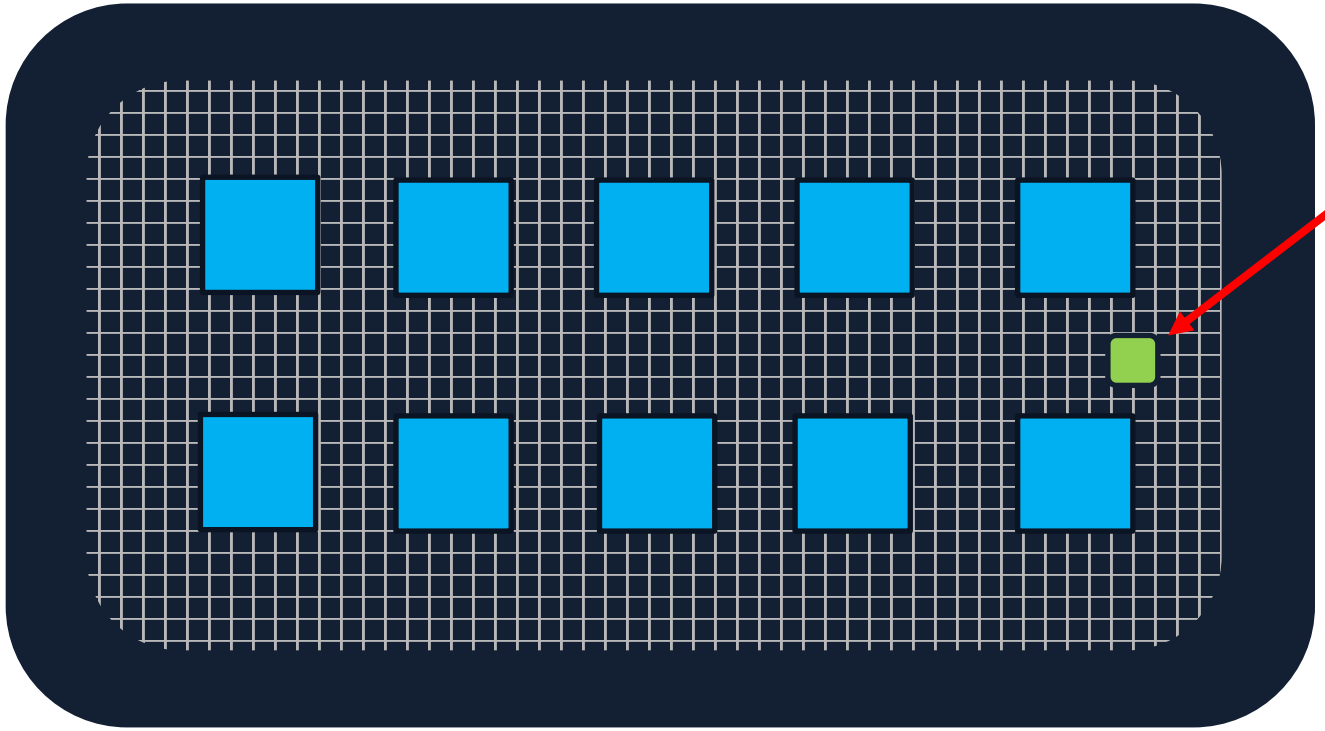
Multi-core processor



SAXPY: performance analysis

We start the code by allocating two vectors, **x** and **y**, with 2^{27} entries each…

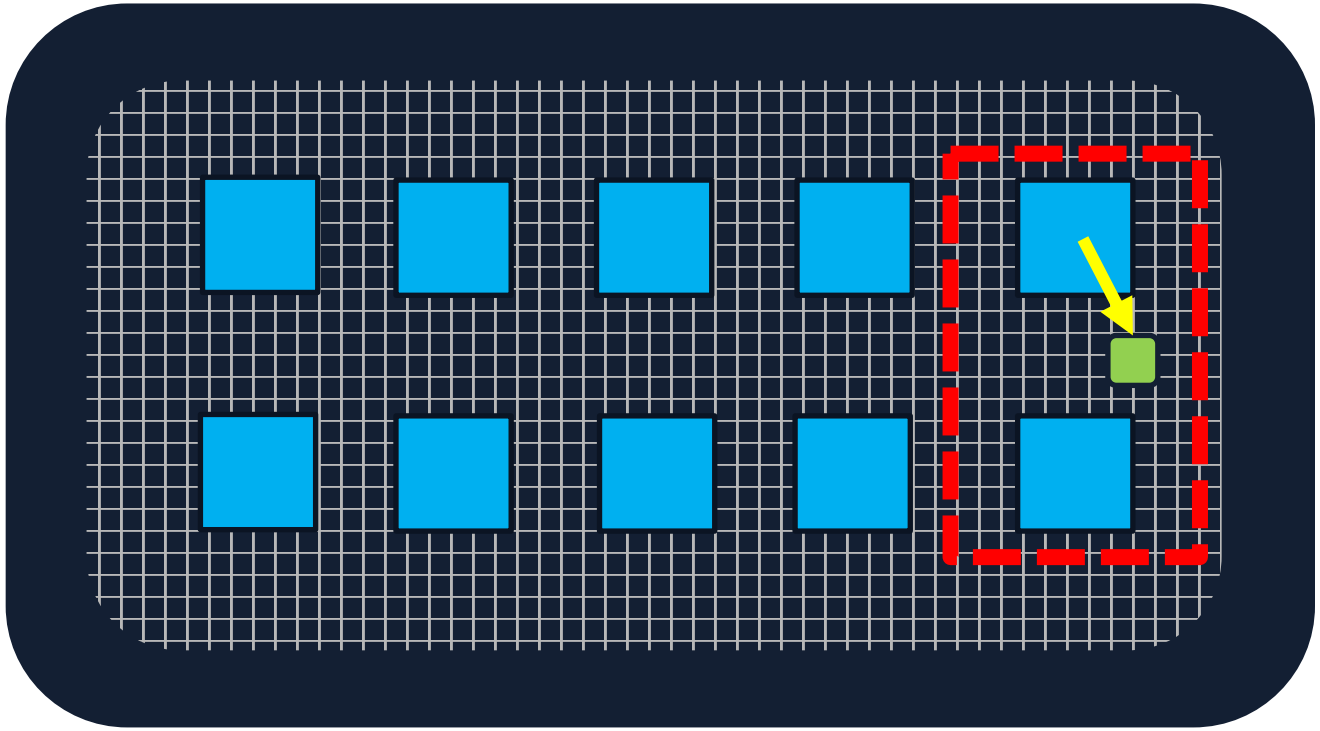
Multi-core processor



Which is only about 1 GB out a total of 256 GB!

SAXPY: performance analysis

Multi-core processor

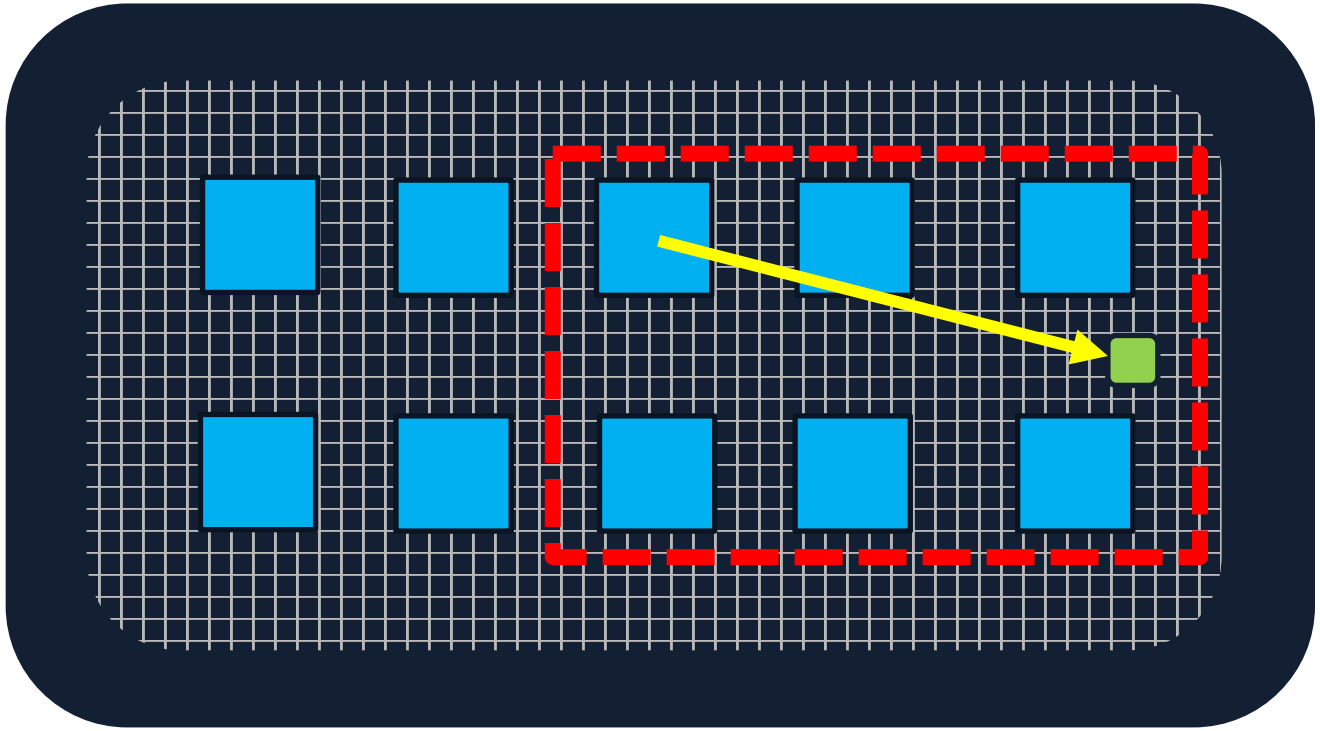


→ Maximum distance that a thread needs to travel to access memory address

N = 2 Threads

SAXPY: performance analysis

Multi-core processor

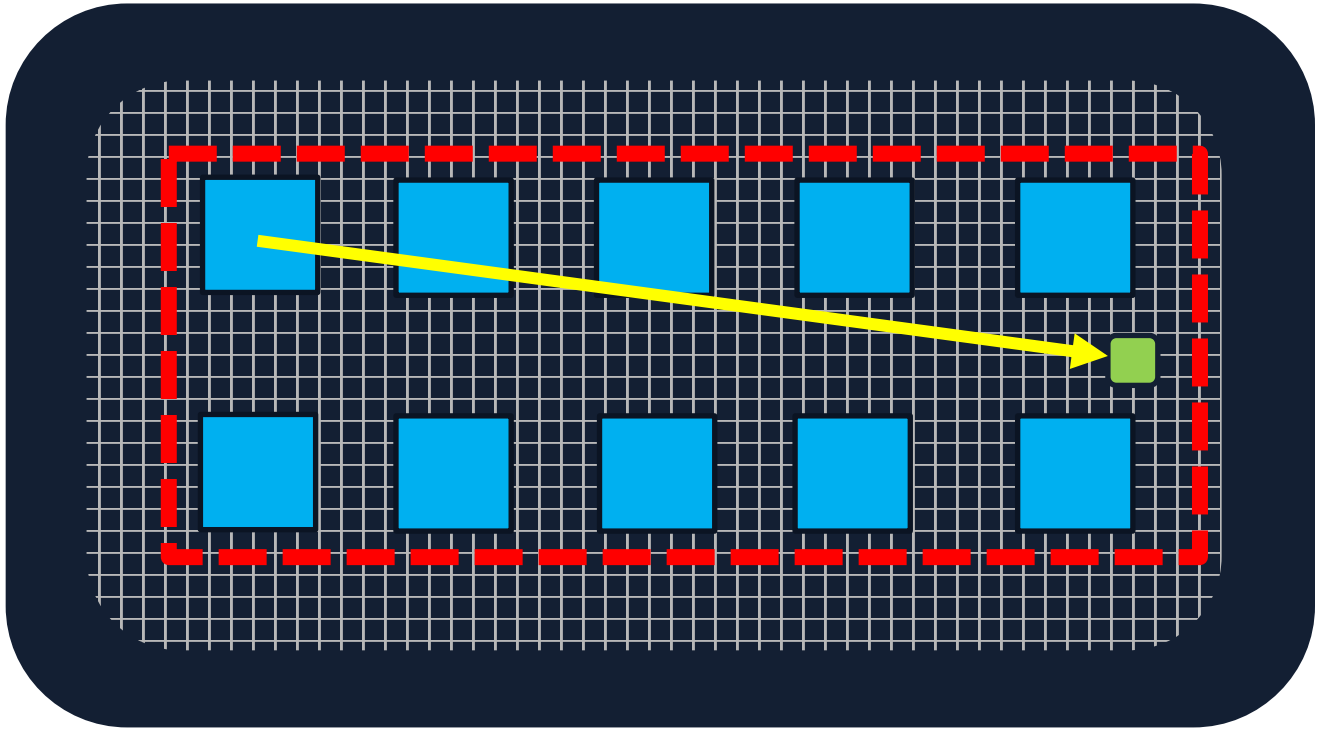


→ Maximum distance that a thread needs to travel to access memory address

N = 6 Threads

SAXPY: performance analysis

Multi-core processor



→ Maximum distance that a thread needs to travel to access memory address

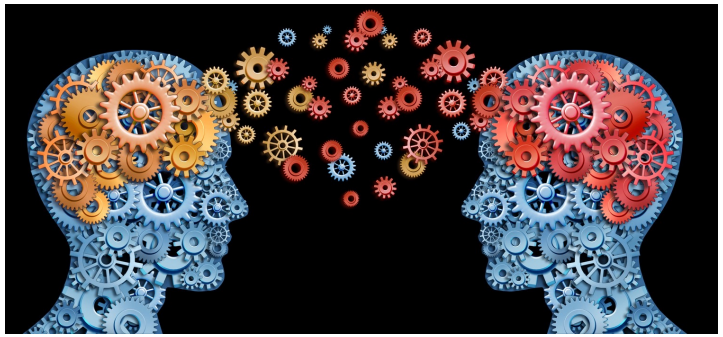
N = 10
Threads

Ideal scaling requires problem scaling as well!

Outline

Part A: **Parallel Computing Overview** (45 min)

- Why parallelism?
- Task and Data parallelism
- Parallel computers
- Threads, processes and more



Part B: **How to Parallelize a Code** (75 min)

Hello, Parallel World!
- Parallel regions
- Compilation and execution

Task Parallelism
- Parallel sections
- Measuring performance

Data Parallelism
- Parallel loops
- Performance and scaling

Summary

Parallel computing is essential for modern scientific research computing

Computers today execute multiple instructions on multiple pieces of data (MIMD)

OpenMP allows for incremental parallelization with very simple directives

Using Task Parallelism, we improved the performance of the STATS application by 2.4x

Using Data Parallelism, we improved the performance of the SAXPY application by 8x

Summary

Parallel computing is essential for modern scientific research computing

Computers today execute multiple instructions on multiple pieces of data (MIMD)

OpenMP allows for incremental parallelization with very simple directives

Using Task Parallelism, we improved the performance of the STATS application by 2.4x

Using Data Parallelism, we improved the performance of the SAXPY application by 8x

In ~100 minutes!

Summary

Parallel computing is essential for modern scientific research computing

Computers today execute multiple instructions on multiple pieces of data (MIMD)

OpenMP allows for incremental parallelization with very simple directives

Using Task Parallelism, we improved the performance of the STATS application by 2.4x

Using Data Parallelism, we improved the performance of the SAXPY application by 8x

In ~100 minutes!

With TWO lines of code!

Take-home message

If your serial code takes **one week** to run… **168h**

- **Take this course** **2h**
- **Work on additional exercises** **2h**
- **Parallelize your code, test it** **4h**
- **Run your code faster (~8x)** **21h**

Take-home message

If your serial code takes **one week** to run… **168h**

- **Take this course** **2h**
- **Work on additional exercises** **2h**
- **Parallelize your code, test it** **4h**
- **Run your code faster (~8x)** **21h**

$$\text{ROI} = \frac{168\text{h} - 21\text{h}}{2\text{h} + 2\text{h} + 4\text{h}} = 18.4$$

For **one single** run!

Take-home message

If your serial code takes **one week** to run… **168h**

- **Take this course** **2h**
- **Work on additional exercises** **2h**
- **Parallelize your code, test it** **4h**
- **Run your code faster (~8x)** **21h**

$$\text{ROI} = \frac{168\text{h} - 21\text{h}}{2\text{h} + 2\text{h} + 4\text{h}} = 18.4$$

For **one single** run!

It is worth giving it a try, and we are here to help you!

Final assignment (optional, encouraged)

Write a simple code that is related to the scientific research problem you are interested in (you may already have it!)

Line up the steps to parallelize it: find the computationally intensive tasks, verify the way it access data, and so on

Draw your application's workflow diagram

Implement it using OpenMP directives

Share your results! Let's add it to the collaborative GitHub folder!

https://github.com/babreu-ncsa/parallelization

Useful links

GitHub Parallelization repo
https://github.com/babreu-ncsa/parallelization

GitHub repo with this session's exercises
https://github.com/babreu-ncsa/IntroToPC

HPC Training Moodle – Parallel Computing in HPC Systems course
https://www.hpc-training.org/xsede/moodle/course/view.php?id=40

OpenMP related contents
https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-1.pdf
https://www.openmp.org/wp-content/uploads/OpenMPRefCard-5.1-web.pdf
https://stackoverflow.com/questions/tagged/openmp

Google Drive folder with this session's materials
https://drive.google.com/drive/folders/1Go7iCyvrSrAARGd4WlxKmQn8lLLgcP27?usp=sharing

Feed us back, please!

Feedback is vital for our continuous improvement as instructors/lecturers, and for our organization.

Please take a couple of minutes to answer this anonymous survey. Any sort of comments are extremely helpful and all of them are going to be carefully taken under consideration.

Thank you!

https://go.ncsa.illinois.edu/IntroToPCFeedback

Q&A

Bruno Abreu, Ph.D.
Research Scientist
babreu@illinois.edu

https://go.ncsa.illinois.edu/IntroToPCFeedback

National Center for Supercomputing Applications
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN