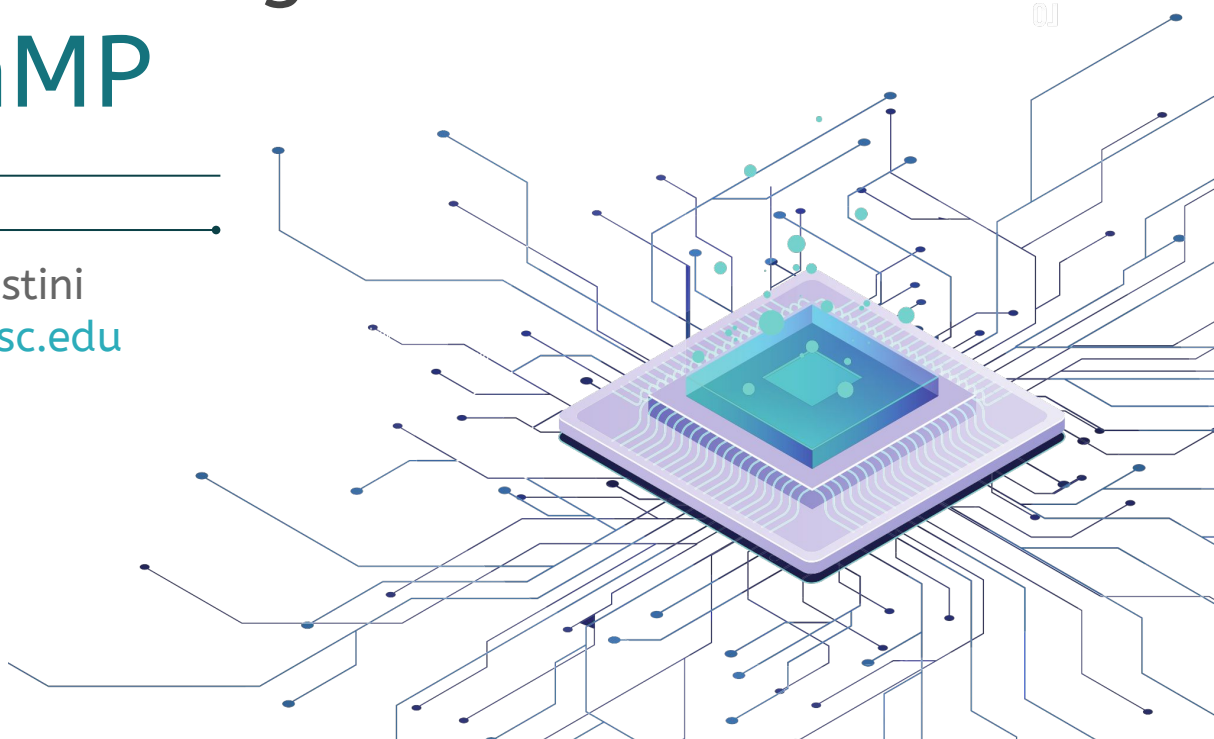
Parallel Programming with OpenMP

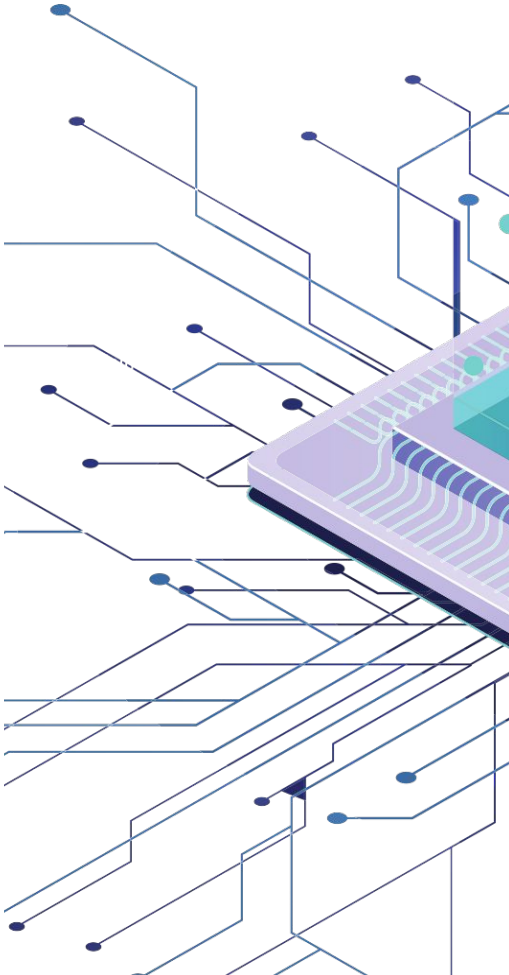
CSE 113 Special Topic Class

Jessica Imlau Dagostini
jessica.dagostini@ucsc.edu



Announcements

- Homework was due yesterday (01/26)
  4 free **late days** on each assignment, so you have until Monday
  No days after that, no exceptions
- Remember: We did not give you all the tests in the autograder!
- We will be grading speedups, which the autograder does not check for right now.
- Part 2: The chunking method we discussed in class will not give a speedup on the servers. You will have to think of other chunking methods. You will need to get a speedup on the grading server to get full points.
- **Homework 2** will be released on **Monday** by midnight



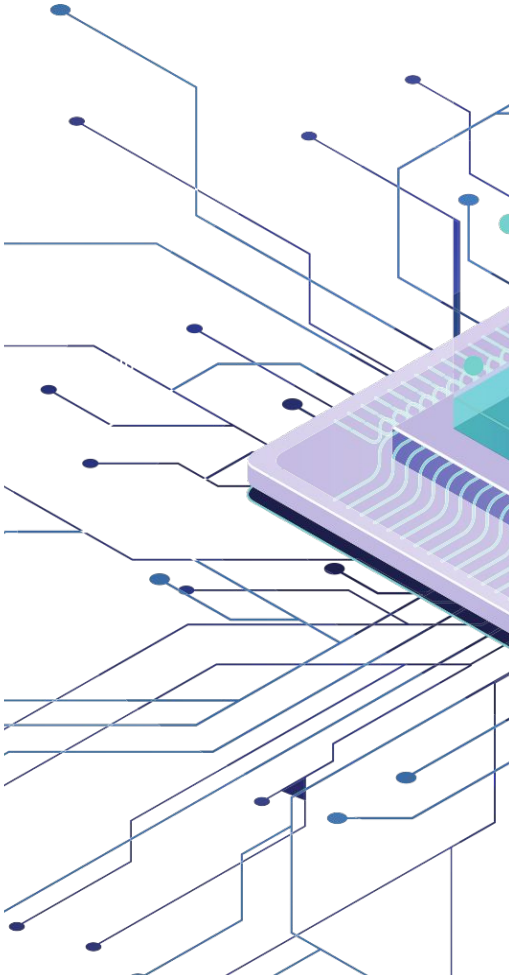
What is OpenMP?

OpenMP is one of the most used parallel programming models nowadays;

It is an model/API for parallelism in machines with multiprocessors (i.e. any computer machine) and shared memory - multithreaded;

OpenMP != OpenMPI

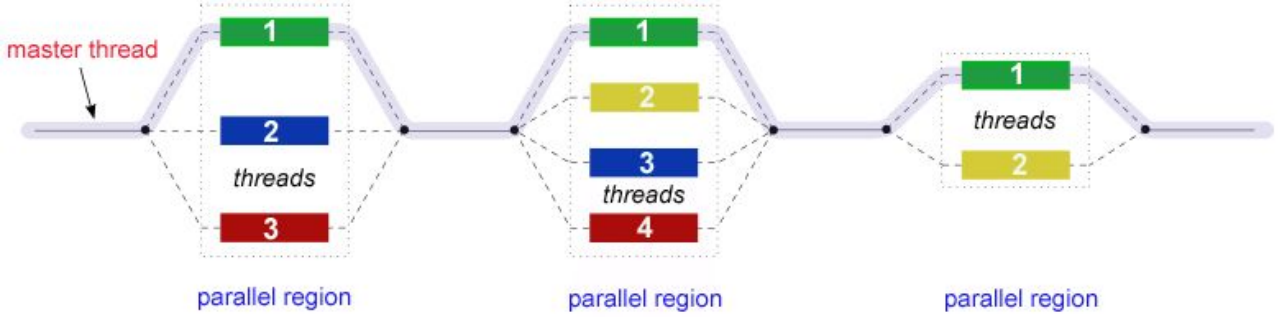
It is a set of compiler directives and libraries to programmers create parallel applications.

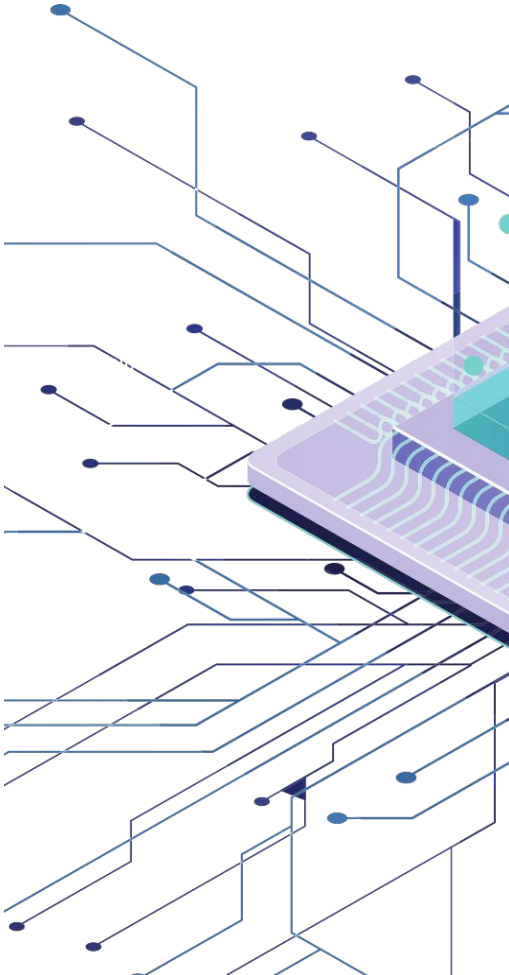


What is OpenMP?

Fork Join Model

OpenMP uses the fork-join model of parallel programming

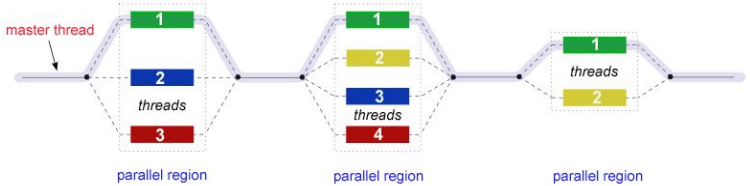




What is OpenMP?

Fork Join Model

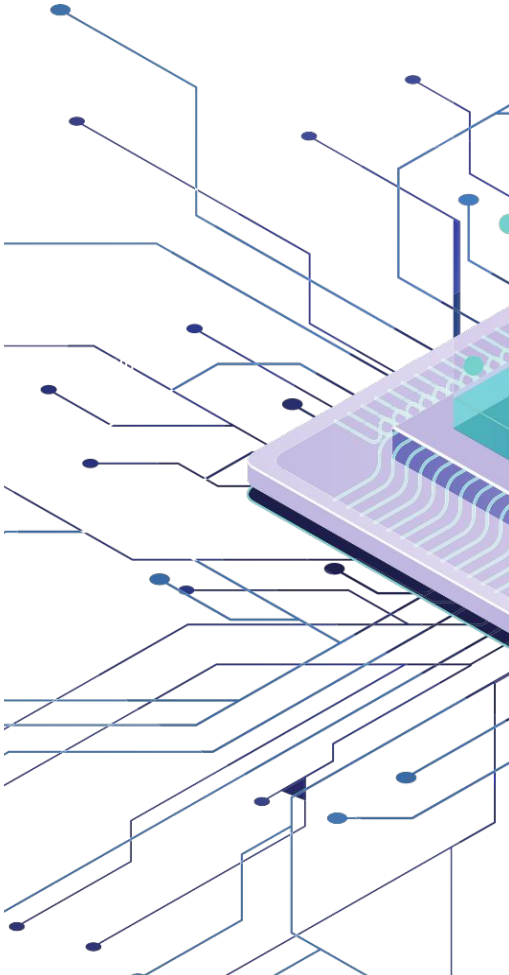
OpenMP uses the fork-join model of parallel programming



All programs start on the master thread, which is sequentially executed until reach a parallel region

Fork - master thread start a group of parallel threads

Join - when the parallel thread finish their jobs, they are synchronized and closed, coming back to the thread master



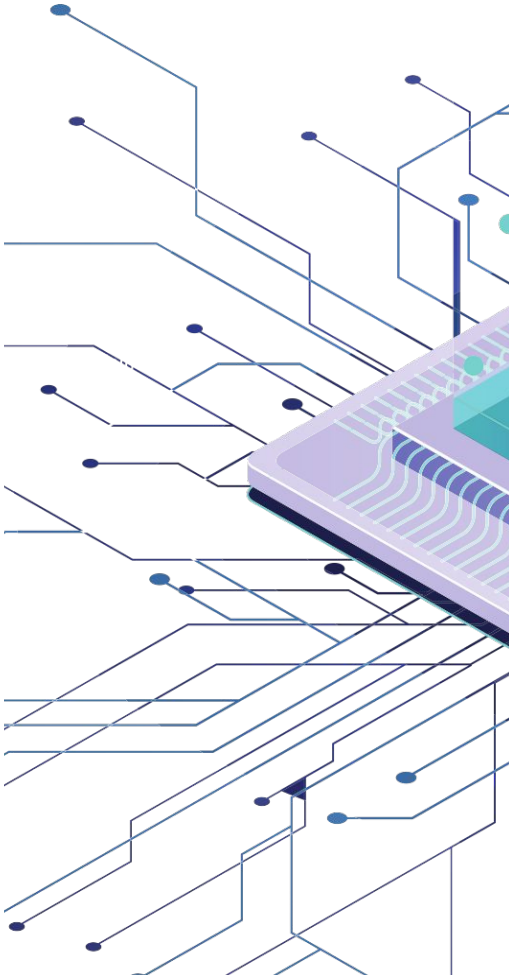
What is OpenMP?

Goals

- Standardize
- Simplify
- Make parallelization easier
- Allow portability

Components

- Compiler Directives
- Runtime Library
- Environment Variables



How to use

OpenMP is available for languages **C**, **C++** e **Fortran**

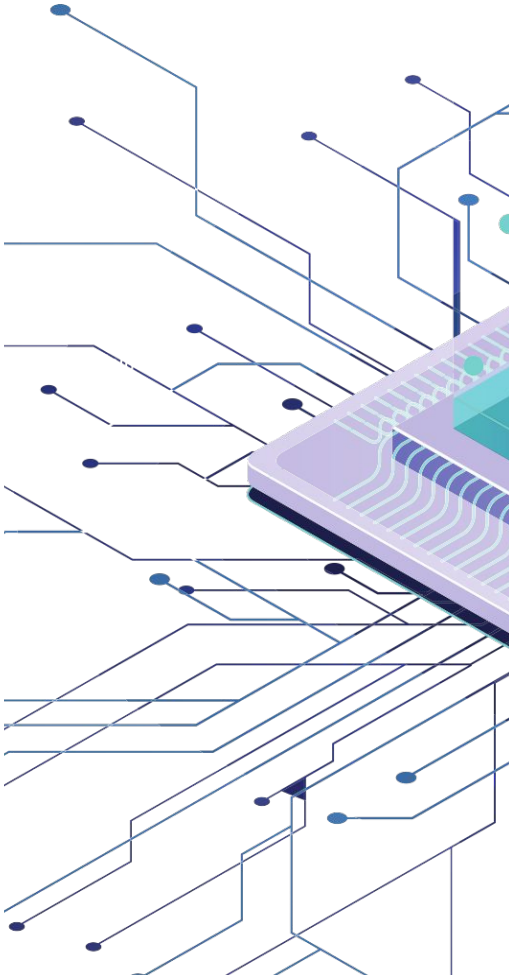
Next examples will be in **C++**

Need to include the **omp.h** library to be able to use specific types and functions from OpemMP

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

To compile, we add the flag **-fopenmp**

```
g++ -o foo -fopenmp foo.cpp
```



Installation

Ubuntu

1. Check if you already have gcc/g++ on your PC

   ```
   gcc -v
   ```

2. If not, install with

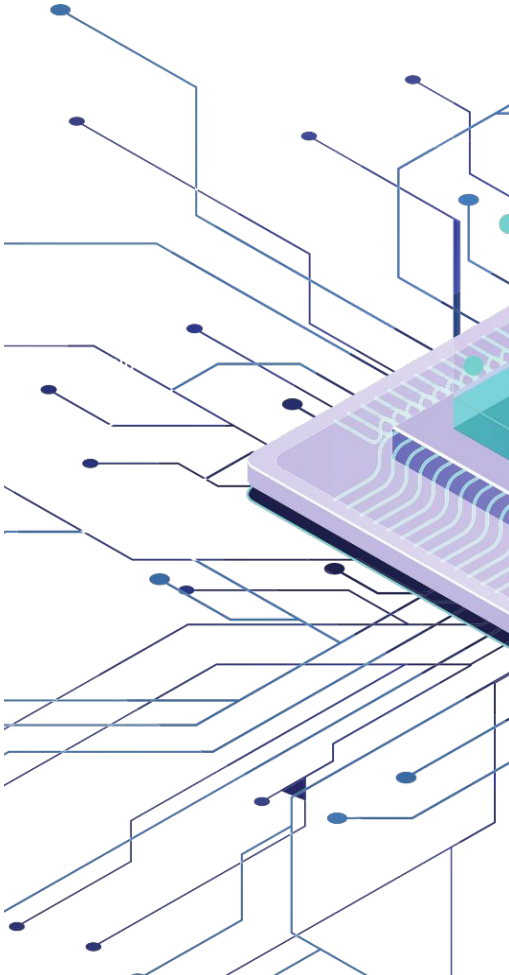
   ```
   apt install g++
   ```

3. To check if the OpenMP is enable, execute

   ```
   echo |cpp -fopenmp -dM |grep -i open
   ```

4. You can also intall the library separated

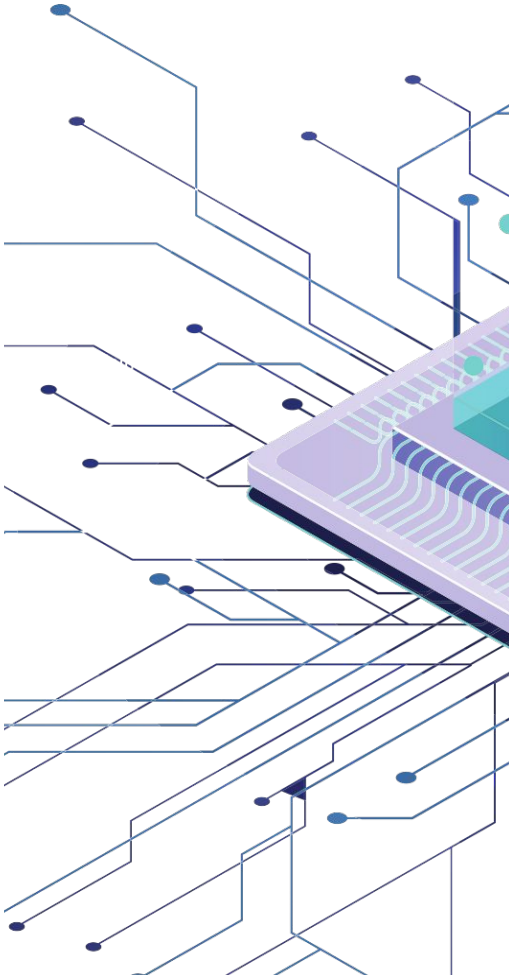
   ```
   apt install libomp-dev
   ```



Installation

Windows

1. Install the minGW compiler if you don't have a gcc/g++ compiler installed already https://osdn.net/frs/redir.php?m=xtom_us&f=mingw%2F68260%2Fmingw-get-setup.exe
2. After downloading, open the executable and click on the following buttons, in order
    a. Install
    b. Continue
    c. Continue
    d. At the MinGW Installation Manager, on "Basic Setup", **DO NOT** select an option that has ada, fortran and objc in their names
    e. In this same screen, change to "All Packages", "MinGW", and select all options that have pthreads (they are 3)



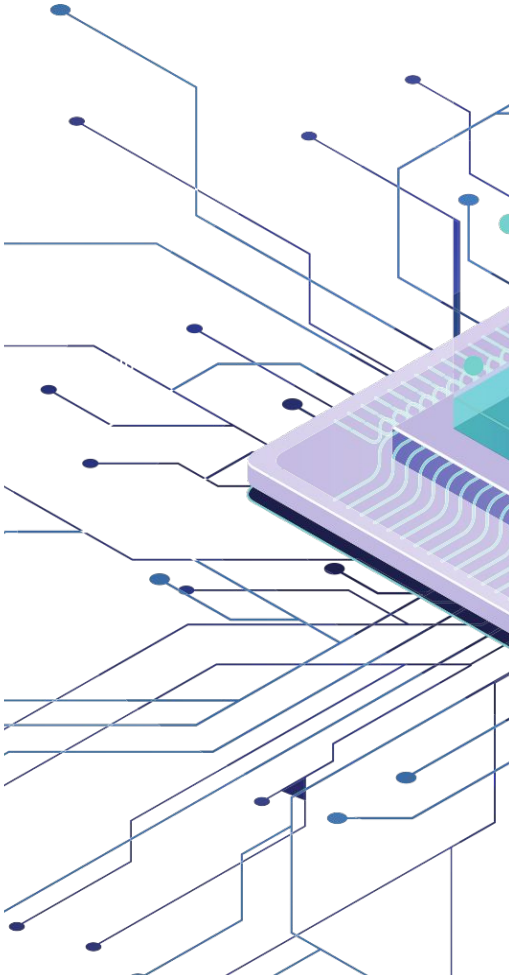
Installation

Windows

1. Checked all options, go to “Installation -> Apply”
2. Now we need to add the environment variables
   a. At the Start menu, look for “environment variables”
   b. Click on the “Set environment variables” option
   c. In the next box, click on “Environment Variables”
   d. At “System Variables”, search for Path and edit
   e. In this new window, click on edit and add

      `C:\MinGW\bin`

   f. Click Ok in all next boxes, until all them are closed



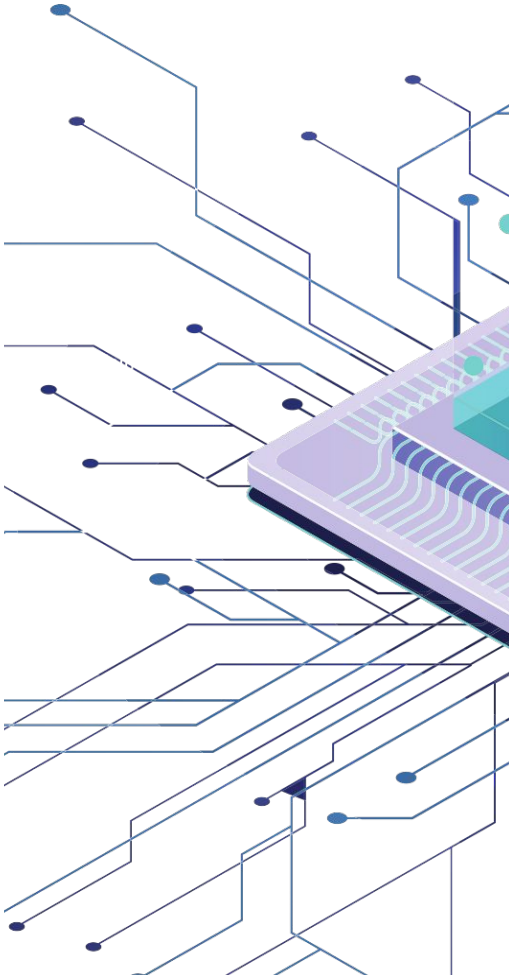
Installation

Windows

1. Open a windows terminal
   a. Search for CMD
2. On terminal, try

```
gcc -v
```

3. Showing the version, it is all set



Directives and Pragmas

OpenMP is based in **directives** and **pragmas**

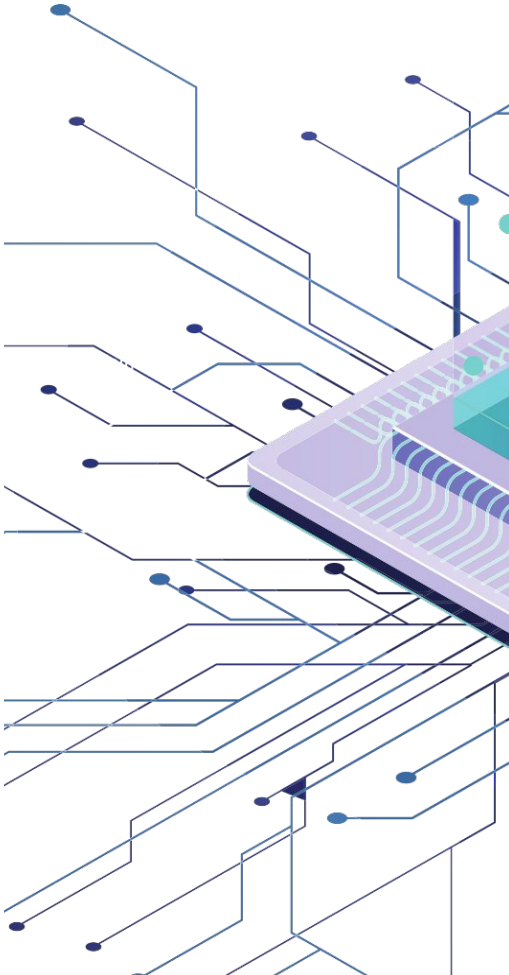
This makes the parallel programming easy

A **directive** is a special line that indicates to the compiler that we are starting a parallel region

```
#pragma omp …
```

To parallelize a loop, for example, we can use the directive

```
#pragma omp parallel for
```



Directives and Pragmas

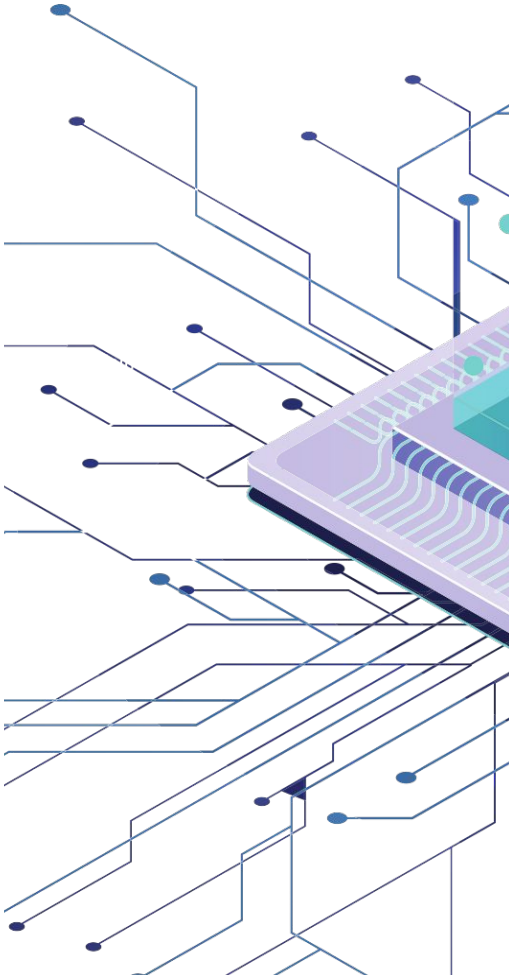
Scope of Variables

Seamless to the sequential programming, variables has scopes, that are the parts of the code where they are visible

In OpenMP, a variable scope refers to the set of threads that can access a variable in parallel

- Variables that can be accessed by all threads from a group has a **shared scope**
- Variables that can only be accessed by one thread only has a **private scope**

The **default** scope to variables declared **before** a parallel block is **shared**



Directives and Pragmas

Clauses

- shared(var1,var2,...)

  Variables that will be shared through all threads (same memory local)
- private(var1,var2,..)

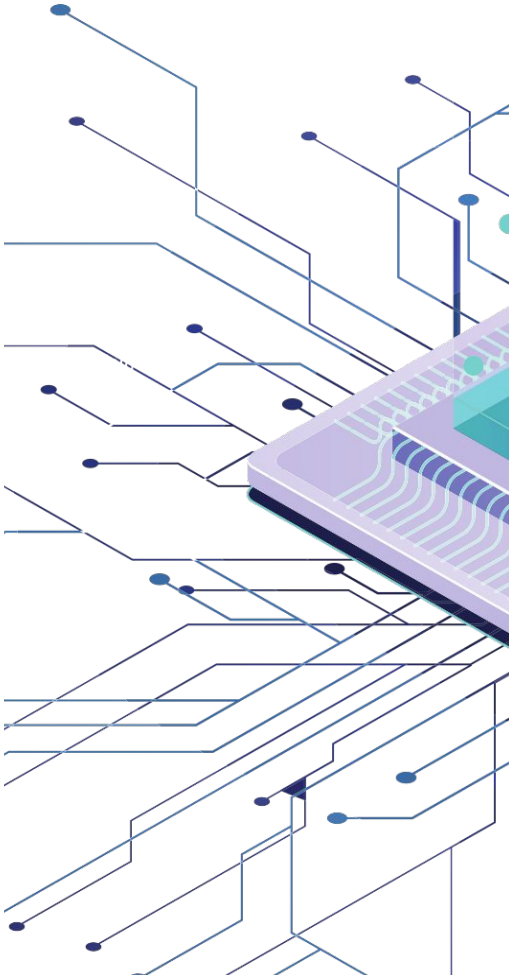
  Each thread has its own copy of each variable (not shared)
- firstprivate(var1,var2,...)

  Private variables that are only initiated when the parallel region starts
- lastprivate(var1,var2,...)

  Private variables that has their values saved only on the last iteration
- schedule(type [,chunk])

  Controls how the loop iterations are spread among threads
- reduction(operator|intrinsic:var1,var2...)

  Make sure that a reduce operation is executed safely



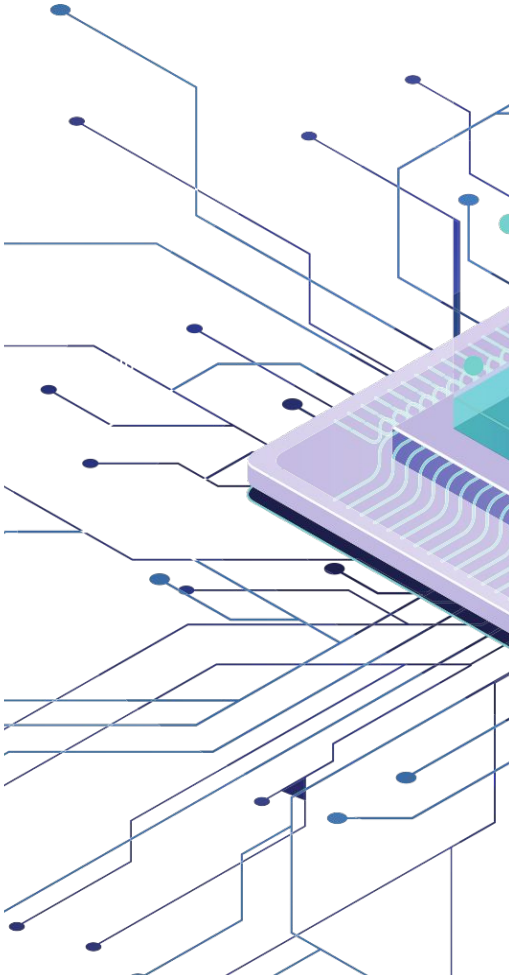
Directives and Pragmas

Hello World with OpenMP

```
#include <omp.h>
#include <cstdio>
int main() {
    #pragma omp parallel
    printf("Hello from thread %d, nthreads %d\n",
            omp_get_thread_num(), omp_get_num_threads());
}
```

To compile

```
gcc -o helloworld -fopenmp helloworld.cpp
```



Directives and Pragmas

For Loops with OpenMP

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

#define MAX 20

int busy(int i) {
    printf("iteration %d on thread %d, nthreads %d\n",
            i, omp_get_thread_num(), omp_get_num_threads());
    return 0;
}

int main() {
    int i;
    #pragma omp parallel for
    for (i=0; i < MAX; i++) {
        busy(i);
    }
}
```