Simple OpenMP

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

Oregon State University
Computer Graphics

OpenMP Multithreaded Programming

- OpenMP stands for "Open Multi-Processing"
- It is run by a consortium of companies, labs, and universities
- OpenMP (IMHO) gives you the biggest multithread benefit per amount of work you have to put into using it

Oregon State University
Computer Graphics

Much of your use of OpenMP will be accomplished by issuing C/C++ "pragmas" to tell the compiler how to build the threads into the executable

#pragma omp directive [clause]

That's it! That's where the compiler comes in.

But, as you are about to find out, doing parallel processing ***at all*** is not difficult.

The trick is doing parallel processing ***well***.
That's where *you* come in.

Using OpenMP in Linux:

g++ -o proj proj.cpp -lm -fopenmp

Using OpenMP in Microsoft Visual Studio:

1. Go to the Project menu ⟶ Project Properties
2. Change the setting Configuration Properties ⟶ C/C++ ⟶ Language ⟶ OpenMP Support to **"Yes (/openmp)"**

Threads

We will get into more detail pretty soon, but for now, know that a thread is an independent execution path for your code to take.

Threads are at their very best when each one can run on a separate hardware core.

Seeing if OpenMP is Supported on Your System:


```
#ifndef  _OPENMP
    fprintf( stderr, "OpenMP is not supported – sorry!\n" );
    exit( 0 );
#endif
```

How to find out how many cores your system has:

```
int  numprocs  =  omp_get_num_procs( );
```

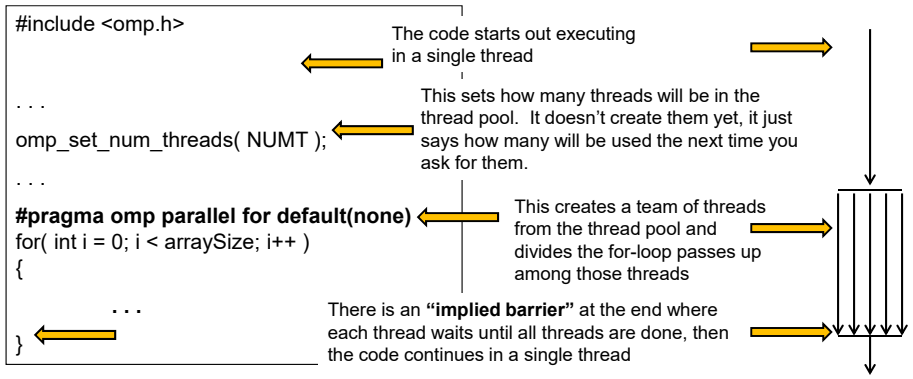
How to specify how many OpenMP threads you want to reserve starting now:

```
omp_set_num_threads( num );
```

How to use one thread per core:

```
omp_set_num_threads(   omp_get_num_procs( )   );
```

Creating OpenMP threads for a for loop



This tells the compiler to parallelize the for-loop into multiple threads. Each thread automatically gets its own personal copy of the variable *i* because it is defined within the for-loop body.

The **default(none)** directive forces you to explicitly declare all variables declared outside the parallel region to be either private or shared while they are in the parallel region. Variables declared within the for-loop statement are automatically private