OpenMP threading: parallel regions

Paolo Burgio
paolo.burgio@unimore.it

Outline

- ✓ Expressing parallelism
  - Understanding parallel threads
- ✓ ~~Memory~~ Data management
  - Data clauses
- ✓ Synchronization
  - Barriers, locks, critical sections
- ✓ Work partitioning
  - Loops, sections, single work, tasks…
- ✓ Execution devices
  - Target

Thread-centric exec. models

- ✓ Programs written in C are implicitly sequential
  - One thread traverses all of the instructions
  - Any form of parallelism must be explicitly/manually coded
  - Start sequential..then create a team of threads
- ✓ E.g., with Pthreads
  - Expose to the programmer "OS-like" threads
  - Units of scheduling
- ✓ Also OpenMP provides a way to do that
  - OpenMP <= 2.5 implements a *thread-centric execution model*
  - Specify the so-called parallel regions

Underlined: Keywords

A history of OpenMP

- ✓ 1997
  - OpenMP for Fortran 1.0
- ✓ 1998
  - OpenMP for C/C++ 1.0
- ✓ 2000
  - OpenMP for Fortran 2.0
- ✓ 2002
  - *OpenMP for C/C++ 2.5*
- ✓ 2008
  - OpenMP 3.0
- ✓ 2011
  - *OpenMP 3.1*
- ✓ 2014
  - *OpenMP 4.5*

Thread -centric

Regular, loop-based parallelism

Task- centric

Irregular, dynamic parallelism

Devices

Heterogeneous parallelism, *à la* GP-GPU

Exercise

Let's
code!

- ✓ Spawn a team of N parallel (P)Threads
  - Each printing "Hello Parallel World"

PThreads: recap

eng.wikipedia.org

POSIX Threads, usually referred to as Pthreads, is an execution model that exists **independently from a language**, as well as a **parallel execution model**. It allows a program to control **multiple different flows of work** that overlap in time.

pthread.h

```
int pthread_create (pthread_t *ID,
                    pthread_attr_t *attr,
                    void *(*body)(void *),
                    void * arg );

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);

int pthread_join (pthread_t th,
                  void **thread_return);
```

your_src.c

```
void *my_pthread_fn(void *arg)
{
  …
}
```

pragma omp parallel construct

```
#pragma omp parallel [clause [[,]clause]...] new-line
  structured-block
```

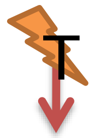
Where clauses can be:

```
if([parallel :] scalar-expression)
num_threads (integer-expression)
default(shared | none)
firstprivate (list)
private (list)
shared (list)
copyin (list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
```

Creating a parreg

- ✓ Master-slave, fork-join execution model
  - Master thread spawns a team of Slave threads
  - They all perform computation in parallel
  - At the end of the parallel region, implicit barrier

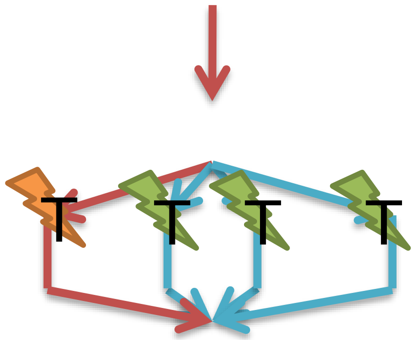
```
int main()
{

  /* Sequential code */

  #pragma omp parallel num_threads(4)
  {


    /* Parallel code */


  } // Parreg end: (implicit) barrier

  /* (More) sequential code */

}
```



Creating a parreg

- ✓ Master-slave, fork-join execution model
  - Master thread spawns a team of Slave threads
  - They all perform computation in parallel
  - At the end of the parallel region, implicit barrier

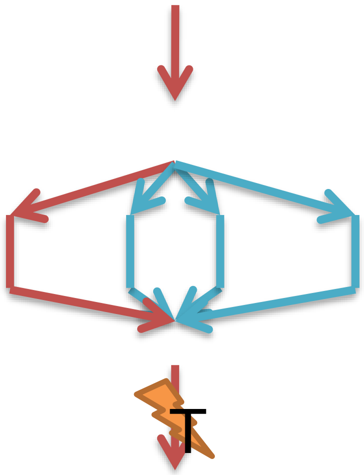
```
int main()
{

  /* Sequential code */

  #pragma omp parallel num_threads(4)
  {


    /* Parallel code */


  } // Parreg end: (implicit) barrier

  /* (More) sequential code */

}
```



Creating a parreg

- ✓ Master-slave, fork-join execution model
  - Master thread spawns a team of Slave threads
  - They all perform computation in parallel
  - At the end of the parallel region, implicit barrier

```
int main()
{

  /* Sequential code */

  #pragma omp parallel num_threads(4)
  {


    /* Parallel code */


  } // Parreg end: (implicit) barrier

  /* (More) sequential code */

}
```



Exercise

Let's code!

- ✓ Spawn a team of parallel (OMP)Threads
  - Each printing "Hello Parallel World"
  - No matter how many threads
- ✓ Don't forget the `-fopenmp` switch
  - Compiler-dependant!

Compiler	Compiler Options
GNU (**gcc**, g++, gfortran)	`-fopenmp`
Intel (icc ifort)	`-openmp`
Portland Group (pgcc,pgCC,pgf77,pgf90)	`-mp`

Thread control

- ✓ OpenMP provides ways to
  - Retrieve thread ID
  - Retrieve number of threads
  - Set the number of threads
  - Specify threads-to-cores affinity
  - …

Get thread ID

omp.h

```c
/*
 * The omp_get_thread_num routine returns
 * the thread number, within the current team,
 * of the calling thread.
 */
int omp_get_thread_num(void);
```

- ✓ Function call
  - Returns an integer
  - Can be used everywhere inside your code
    - Also in sequential parts
- ✓ Don't forget to `#include <omp.h>`!!
- ✓ Master thread (typically) has ID #0

T

Exercise

Let's
code!

- ✓ Spawn a team of parallel (OMP)Threads
  - Each printing "Hello Parallel World. I am thread #<*tid*>"
  - Also, print "Hello Sequential World. I am thread #<*tid*>" before and after parreg
  - What do you see?

Get the number of threads

omp.h

```
/*
 * The omp_get_num_threads routine returns
 * the number of threads in the current team.
 */
int omp_get_num_threads(void);
```

- Function call
  - Returns an integer
  - Can be used everywhere inside your code
    - Also in sequential parts
  - Don't forget to `#include <omp.h>`!!
- BTW
  - …thread ID from `omp_get_thread_num` is always < this value..

Exercise

Let's
code!

- ✓ Spawn a team of parallel (OMP)Threads
  - Each printing "Hello Parallel World. I am thread #<*tid*> *out of* <*num*>"
  - Also, print "Hello Sequential World. I am thread #<*tid*> *out of* <*num*>" before and after parreg
  - What do you see?

Set the number of threads

- ✓ "This, we already saw ☺"
  - NO(t completely)!
- ✓ In OpenMP, several ways to do this
  - Implementation-specific default
- ✓ In order of priority..
  1. OpenMP `num_threads` clause
  2. Function APIs (explicit function call)
  3. Environmental vars (at the <u>OS</u> level)

Set the number of threads (3)

```bash
# The OMP_NUM_THREADS environment variable sets
# the number of threads to use for parallel regions

export OMP_NUM_THREADS=4
```

- ✓ Unix environmental variable
  - (Might use `setenv`, `set` or distro-specific commands)

Set the number of threads (2)

omp.h

```
/*
 * The omp_set_num_threads routine affects the number of threads
 * to be used for subsequent parallel regions that do not specify
 * a num_threads clause, by setting the value of the first
 * element of the nthreads-var ICV of the current task.
 */
void omp_set_num_threads(int num_threads);
```

- ✓ Function call
  - Accepts an integer
  - Can be used everywhere inside your code
    - Also in sequential parts
  - Overrides value from `OMP_NUM_THREADS`
  - Affects all of the subsequent parallel regions

- ✓ Don't forget to `#include <omp.h>`!!

Set the number of threads (1)

```
#pragma omp parallel [clause [[,]clause]...] new-line
  structured-block
```

Where clauses can be:

```
if([parallel :] scalar-expression)
num_threads (integer-expression)
default(shared | none)
firstprivate (list)
private (list)
shared (list)
copyin (list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
```

Exercise

Let's
code!

- ✓ Spawn a team of parallel (OMP)Threads
  - Each printing "Hello Parallel World. I am thread #*<tid>* *out of <num>*"
  - Also, print "Hello Sequential World. I am thread #*<tid>* *out of <num>*" before and after parreg
  - Play with
    - `OMP_NUM_THREADS=...`
    - `omp_set_num_threads(...)`
    - `num_threads(...)`
- ✓ Do it at home

The `if` clause

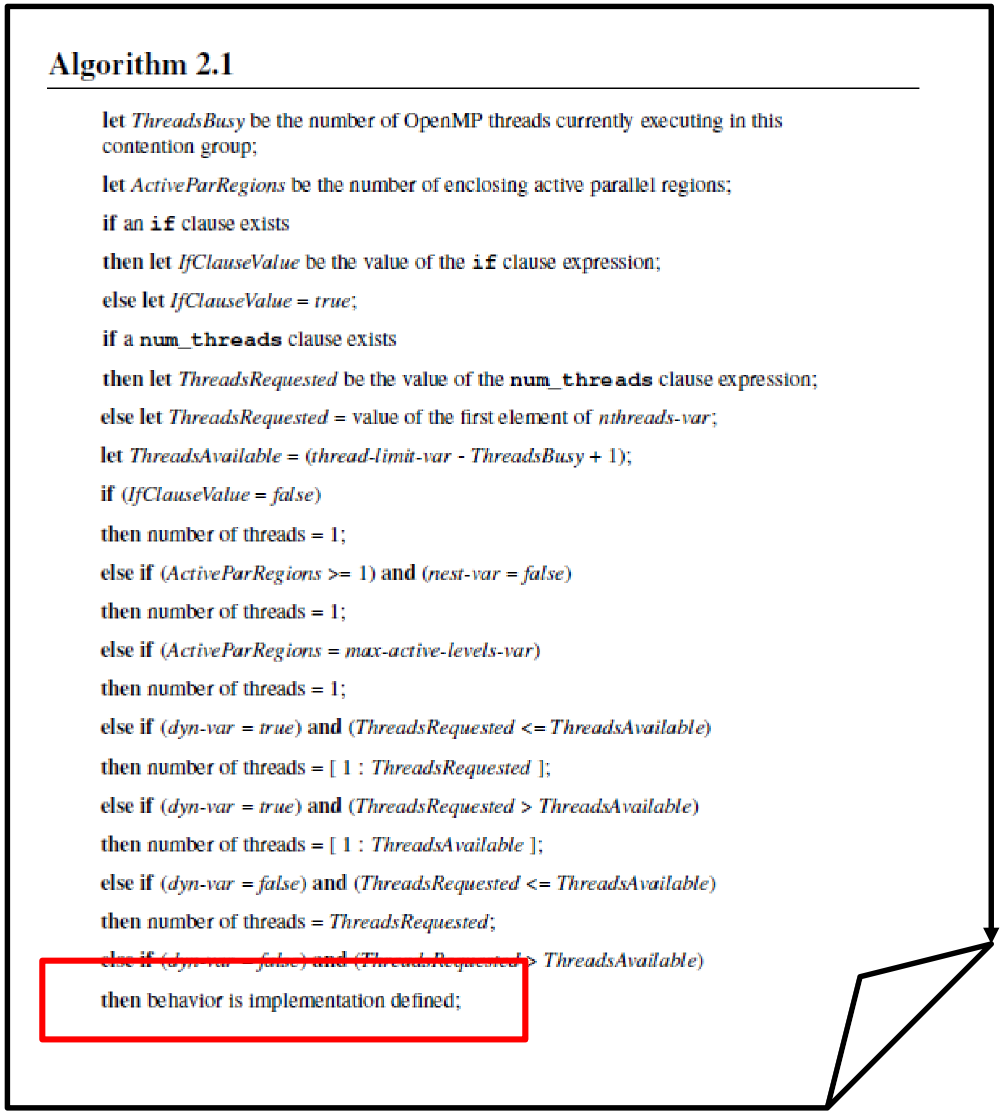
```
#pragma omp parallel [clause [[,]clause]...] new-line
  structured-block
```

Where clauses can be:

```
if([parallel :] scalar-expression)
num_threads (integer-expression)
default(shared | none)
firstprivate (list)
private (list)
shared (list)
copyin (list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
```

- ✓ If `scalar-expression` is `false`, then spawn a single-thread region
- ✓ We will see it also in other constructs…
  - *"Can be used in **combined constructs**, in this case programmer must specify which one it refers to (in this case, with the* `parallel` *specifier)"*

Algorithm that determines #threads



Algorithm 2.1

```
let ThreadsBusy be the number of OpenMP threads currently executing in this
contention group;
let ActiveParRegions be the number of enclosing active parallel regions;
if an if clause exists
then let IfClauseValue be the value of the if clause expression;
else let IfClauseValue = true;
if a num_threads clause exists
then let ThreadsRequested be the value of the num_threads clause expression;
else let ThreadsRequested = value of the first element of nthreads-var;
let ThreadsAvailable = (thread-limit-var - ThreadsBusy + 1);
if (IfClauseValue = false)
then number of threads = 1;
else if (ActiveParRegions >= 1) and (nest-var = false)
then number of threads = 1;
else if (ActiveParRegions = max-active-levels-var)
then number of threads = 1;
else if (dyn-var = true) and (ThreadsRequested <= ThreadsAvailable)
then number of threads = [ 1 : ThreadsRequested ];
else if (dyn-var = true) and (ThreadsRequested > ThreadsAvailable)
then number of threads = [ 1 : ThreadsAvailable ];
else if (dyn-var = false) and (ThreadsRequested <= ThreadsAvailable)
then number of threads = ThreadsRequested;
else if (dyn-var = false) and (ThreadsRequested > ThreadsAvailable)
then behavior is implementation defined;
```

- ✓ OpenMP Specifications
  - – Section 2.1
  - – http://www.openmp.org

Even more control…

- ✓ OpenMP provides fine-grain tuning of all the main "control knobs"
  - Dynamic thread number adjustment
  - Nesting level
  - Threads stack size
  - …
- ✓ More and more with every new version of specifications

Nested parallel regions

- ✓ One can create a parallel region within a parallel region
  - – A new team of thread is created
- ✓ Enabled-disabled via environmental var, or library call

Nested parallel regions control

Bash Shell

```
# The OMP_NESTED environment variable controls nested parallelism
# by setting the initial value of the nest-var ICV.
# The behavior of the program is implementation defined
# if the value of OMP_NESTED is neither true nor false.
setenv OMP_NESTED false

# The OMP_MAX_ACTIVE_LEVELS environment variable controls
# the maximum number of nested active parallel regions
setenv OMP_MAX_ACTIVE_LEVELS 4
```

omp.h

```
/*
 * The omp_set_nested routine enables or disables
 * nested parallelism
 */
void omp_set_nested(int nested);

/*
 * The omp_get_team_size routine returns,
 * for a given nested level of the current thread,
 * the size of the thread team
 * to which the ancestor or the current thread belongs. */
int omp_get_team_size(int level);
```

Nested parallel regions

- Easy to lose control..
  - Too many threads!
  - Their number explodes
  - Be ready to debug..

Set the number of (nested) threads

```
# The OMP_NUM_THREADS environment variable sets
# the number of threads to use for parallel regions

setenv OMP_NUM_THREADS 4,8,4
```

- ✓ Unix environmental variable
- ✓ Commas separate nesting levels
  - 4 threads at level 0
  - 8 threads at level 1
  - 4 threads at level 2
  - …

Dynamic # threads adjustment

- ✓ The OpenMP implementation might decide to dynamically adjust the number of thread within a parreg
  - Aka the team size
  - Under heavy load might be reduced
- ✓ Also this can be disabled

Dynamic # threads adjustment

Bash Shell

```
# The OMP_DYNAMIC environment variable controls dynamic adjustment
# of the number of threads to use for executing parallel regions
setenv OMP_DYNAMIC true
```

omp.h

```
/*
 * The omp_set_dynamic routine enables or disables dynamic adjustment
 * of the number of threads available for the execution of subsequent
 * parallel regions by setting the value of the dyn-var ICV.
 */
void omp_set_dynamic(int dynamic_threads);

/*
 * The omp_get_dynamic routine returns the value of the dyn-var ICV,
 * which determines whether dynamic adjustment of the number of threads
 * is enabled or disabled.
 */
int omp_get_dynamic(void);
```

Threads stack size

- ✓ Can specify low-level details such as the stack size
  - – Why only via environmental var?

Bash Shell

```
# The OMP_STACKSIZE environment variable controls the size of the stack
# for threads created by the OpenMP implementation,
# by setting the value of the stacksize-var ICV.
# The environment variable does not control the size of the stack
# for an initial thread.
# The value of this environment variable takes the form:
#      size | sizeB | sizeK | sizeM | sizeG

setenv OMP_STACKSIZE 2000500B
setenv OMP_STACKSIZE "3000 k "
setenv OMP_STACKSIZE 10M
setenv OMP_STACKSIZE " 10 M "
setenv OMP_STACKSIZE "20 m "
setenv OMP_STACKSIZE " 1G"
setenv OMP_STACKSIZE 20000
```

Internal Control Variables

ICV

OpenMP specifications

*An OpenMP implementation must act **as if** there are **internal control variables (ICVs)** that control the behavior of an OpenMP program. These ICVs store information such as the number of threads to use for future **parallel** regions, the schedule to use for worksharing loops and whether nested parallelism is enabled or not.*

The ICVs are given values at various times (described below) during the execution of the program. They are initialized by the implementation itself and may be given values through OpenMP environment variables and through calls to OpenMP API routines.

The program can retrieve the values of these ICVs only through OpenMP API routines.

For purposes of exposition, this document refers to the ICVs by certain names, but an implementation is not required to use these names or to offer any way to access the variables other than through the ways shown in Section 2.3.2 on page 37.

The proc_bind clause

```
#pragma omp parallel [clause [[,]clause]...] new-line
  structured-block
```

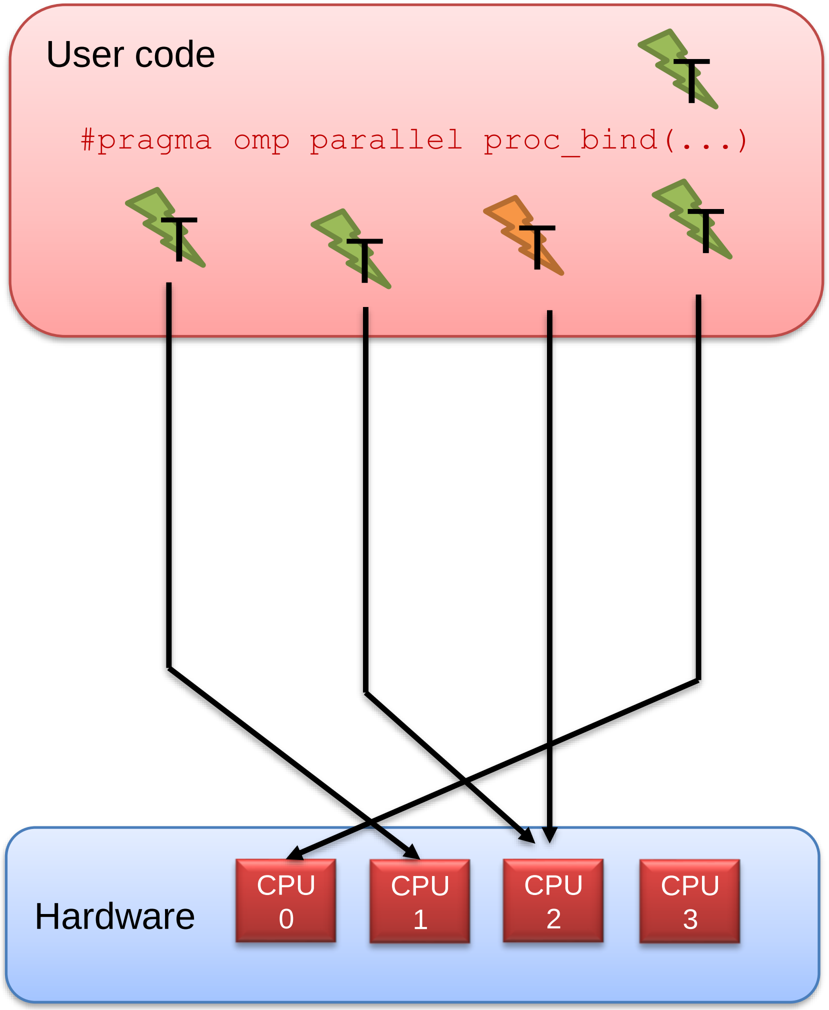
Where clauses can be:

```
if([parallel :] scalar-expression)
num_threads (integer-expression)
default(shared | none)
firstprivate (list)
private (list)
shared (list)
copyin (list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
```

- ✓ Lets you specify *where* threads *should* be scheduled
- ✓ OMP has no notion of core/processor, but rather of place partitions
  - – Why?
- ✓ It might not be possible
  - – In this case, behavior is implementation-defined

Thread affinity

- ✓ `master`: same place as the master thread
- ✓ `close` to master thread
  - Same place partition
- ✓ `spread`: sparse distribution of the team among the places of parent's place partition



Thread affinity: pros. vs cons.

- ✓ Used (in some systems) to boost performance
- ✓ If threads are in the same ~~core~~ place, they can exchange data more efficiently
  - E.g., through caches
- ✓ If threads are in the samee ~~core~~ place, they are implicitly sequentialized by the OS
  - And multi/hyper-threading?
  - Might not always be available

Get/set thread affinity (3)

Bash Shell

```
# The OMP_PROC_BIND environment variable sets the initial value
# of the bind-var ICV. (...) The values of the list set the thread
# affinity policy to be used for parallel regions at the
# corresponding nested level
setenv OMP_PROC_BIND false
setenv OMP_PROC_BIND "spread, spread, close"
```

- ✓ Unix environmental variable
  - (Might use `setenv`, `set` or distro-specific commands)

Get thread affinity (2)

omp.h

```
/*
 * The C/ C++ header file (omp.h) define the valid constants.
 * The valid constants must include the following
 */
typedef enum omp_proc_bind_t {
 omp_proc_bind_false = 0,
 omp_proc_bind_true = 1,
 omp_proc_bind_master = 2,
 omp_proc_bind_close = 3,
 omp_proc_bind_spread = 4
} omp_proc_bind_t;

/*
 * The omp_get_proc_bind routine returns the thread affinity policy
 * to be used for the subsequent nested parallel regions
 * that do not specify a proc_bind clause.
 */
omp_proc_bind_t omp_get_proc_bind(void);
```

- ✓ Function call
  - No `omp_set_proc_bind …`
- ✓ Don't forget to `#include <omp.h>`!!

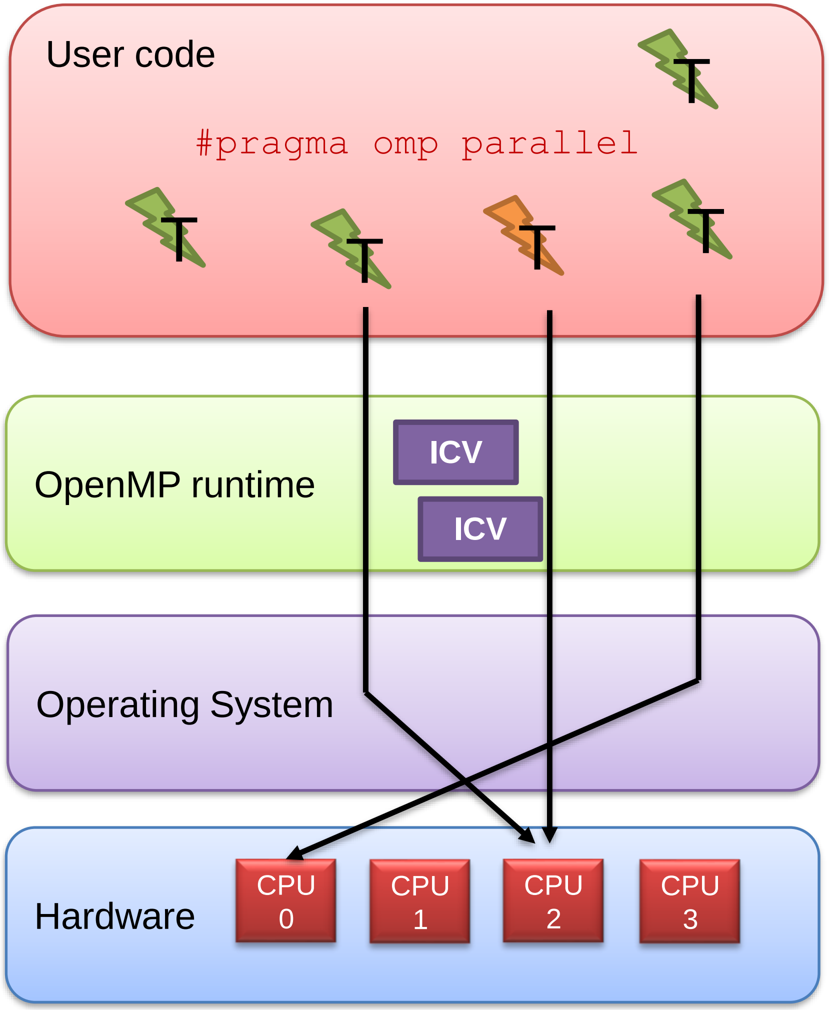
Under the hood

- ✓ You have control on # threads
  - – Partly
- ✓ You have parial control on where the threads are scheduled
  - – Affinity
- ✓ You have no control on the actual scheduling!
  - – Demanded to OS + runtime

- ✓ …"OS and runtime"?

OpenMP software stack

- ✓ Multi-layer stack
  - – Engineered for portability



How to run the examples

Let's code!

- ✓ Download the `Code/` folder from the course website

- ✓ Compile
- ✓ `$ gcc -fopenmp code.c -o code`

- ✓ Run (Unix/Linux)

```
$ ./code
```

- ✓ Run (Win/Cygwin)

```
$ ./code.exe
```

References

- ✓ "Calcolo parallelo" website
  - http://algogroup.unimore.it/people/marko/courses/programmazione_parallela/
- ✓ My contacts
  - paolo.burgio@unimore.it
  - http://hipert.mat.unimore.it/people/paolob/
- ✓ PThreads
  - https://computing.llnl.gov/tutorials/pthreads/
  - http://php.net/manual/en/book.pthreads.php
  - http://www.google.com
- ✓ OpenMP specifications
  - http://www.openmp.org
  - http://www.openmp.org/mp-documents/openmp-4.5.pdf
  - http://www.openmp.org/mp-documents/OpenMP-4.5-1115-CPP-web.pdf