Introduction to OpenMP

Computational Science II (CAAM 520)

Christopher Thiele

Rice University, Spring 2020

Overview

- What is OpenMP?
- When to use OpenMP
- Using OpenMP in C code
- Compiling, linking, and running OpenMP code
- The fork-join model

What is OpenMP

OpenMP (Open Multi-Processing) is an API for multi-processing, mainly multi-threading, in C, C++, and Fortran.

- 1997: OpenMP 1.0 (Fortran)
- 1998: First OpenMP standard for C and C++
- 2000: OpenMP 2.0 (mainly parallelization of loops)
- 2008: OpenMP 3.0 (task parallelism)
- 2013: OpenMP 4.0 (accelerators, SIMD)
- 2018: OpenMP 5.0

When to use OpenMP

OpenMP implements shared memory parallelism with multi-threading (and accelerators).

Hence, it is suitable for

- multi-core CPUs,
- multi-CPU systems, and
- systems with accelerators.

It is not suitable for distributed memory environments such as computer clusters.

Using OpenMP in C code

OpenMP consists of two main components: ***preprocessor directives*** and a ***library*** of functions.

```c
#include <omp.h>

// ...

{
  // Use an OpenMP preprocessor directive.
  #pragma omp parallel
  {
    // Call an OpenMP library function.
    const int thread_num = omp_get_thread_num();
  }
}
```

Using OpenMP in C code

Why this design?

Pros:

- Preprocessor directives can be used to annotate and parallelize existing code.
- If the compiler does not support OpenMP, it can simply ignore the directives.

Cons:

- Directives can be somewhat limiting.
- Compilers must support OpenMP.
- Some compilers are bad at this: Microsoft Visual Studio supports OpenMP 2.0 (2000).

Compiling, linking, and running OpenMP code

Consider the following example:

```
#pragma omp parallel
{
  const int thread_num = omp_get_thread_num();
  const int num_threads = omp_get_num_threads();
  printf("hello, world from thread %d/%d!\n",
         thread_num, num_threads);
}
```

→ By default, the compiler will ignore the OpenMP directive, and the linker will not find `omp_get_num_threads()` etc.

Compiling, linking, and running OpenMP code

Consider the following example:

```
#pragma omp parallel
{
  const int thread_num = omp_get_thread_num();
  const int num_threads = omp_get_num_threads();
  printf("hello, world from thread %d/%d!\n",
         thread_num, num_threads);
}
```

→ By default, the compiler will ignore the OpenMP directive, and the linker will not find `omp_get_num_threads()` etc.

Compiling, linking, and running OpenMP code

We must instruct the compiler to consider OpenMP directives.

For GCC, use the `-fopenmp` flag.

If compilation and linking are done separately, use the `-fopenmp` flag for both!

Compiling, linking, and running OpenMP code

We can run OpenMP applications just like any other application.

How does OpenMP know how many threads to use?

- The `OMP_NUM_THREADS` environment variable can be used to set the initial number of threads.
- The number of threads can be modified using the `omp_set_num_threads()` function anywhere in the code.
- The number of threads can be set in the OpenMP directive, e.g.,

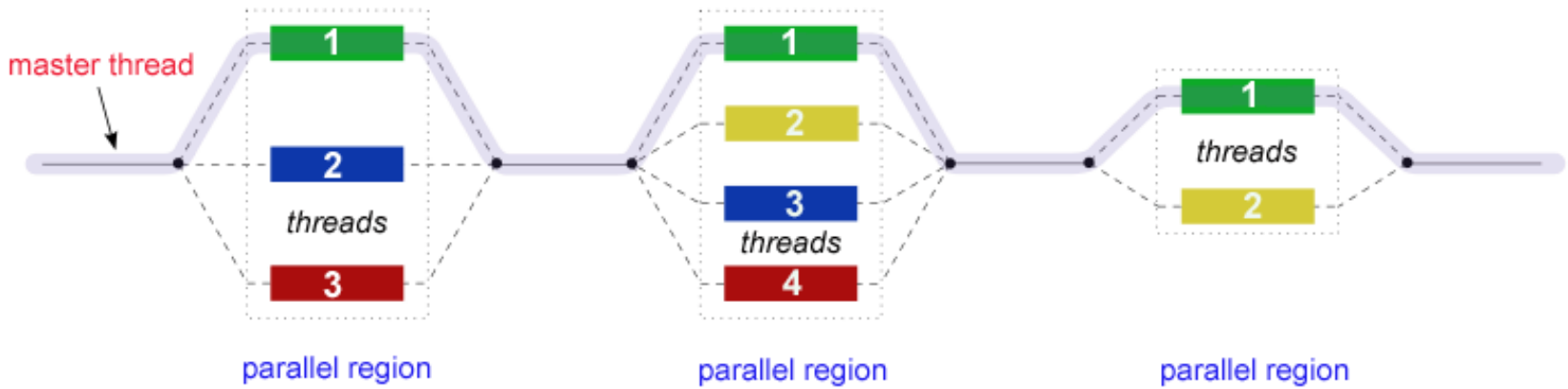
```
#pragma omp parallel num_threads(4)
```

The fork-join model

When entering a parallel region, an OpenMP application ***forks*** into multiple threads.

All threads ***join*** with the ***master thread*** when leaving the parallel region.

OpenMP will create and finish threads automatically as needed.



The fork-join model

Recall that threads are scheduled and executed ***independently*** by the OS.

→ They are not executed in any particular order, in one piece, etc.!

Recall that threads are a software concept.

→ It is possible to run more threads than there are CPU cores (oversubscription).