OpenMP

Diego Fabregat-Traver and Prof. Paolo Bientinesi

HPAC, RWTH Aachen
fabregat@aices.rwth-aachen.de

WS15/16

OpenMP

- API for shared-memory parallelism
- Steered by the OpenMP ARB (industry, research)
- Supported by compilers on most platforms
- Not a programming language. Mainly annotations to the (sequential) code.
- OpenMP API consists of:
  - Compiler directives
  - Library routines
  - Environment variables
- Simple to use, high-level, incremental parallelism
- Performance oriented
- Data (and task) parallelism

(Very brief) History of OpenMP

- SC97: Group HPC experts (industry, research) presented OpenMP, to propose a unified model to program shared-memory systems.
- A company was set up to own and maintain the new standard: The openmp architecture review board (openmparb)
- People efforts on: extending the standard, developing implementations, teaching and spreading the word, cOMPunity for the interaction between vendors, researchers and users.
- Originally primarily designed to exploit concurrency in structured loop nests.

Main ideas

- User gives a high-level specification of the portions of code to be executed in parallel

```
int main( ... )
{
    ...
    #pragma omp parallel
    {
        <region executed by multiple threads>
    }
    ...
}
```

`pragma` (pragmatic): tell the compiler to use some compiler-dependent feature/extension.

Main ideas (II)

- User may provide additional information on how to parallelize
  - `#pragma omp parallel num_threads(4)`
  - `omp_set_schedule( static | dynamic | ... );`
  - `omp_set_lock( lock_var );`
- OpenMP takes care of the low level details of creating threads, execution, assigning work, ...
- Provides relatively easy variable scoping, synchronization and primitives to avoid data races.
- Usage:
  - `#include "omp.h"`
  - `[gcc|icc] -fopenmp <source.c> -o <executable.x>`

Hello world!

Exercise 1: Warming up

- Write an OpenMP multi-threaded program where each thread prints "Hello world!".

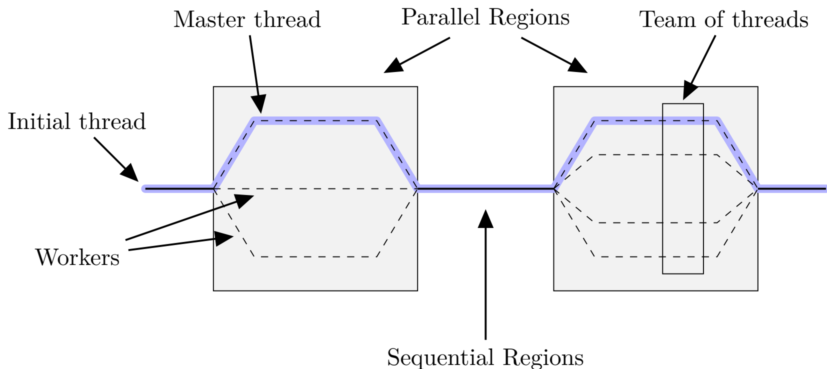
```
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
    printf("Hello world!\n");

    return 0;
}
```

Hint:

`#pragma omp parallel`

Main ideas (III)

Fork-join paradigm



Incremental parallelism

- A common approach to writing OpenMP programs:
  - Identify paralellism in your sequential code
  - Incremental parallelism: introduce directives in one portion of the code, leave the rest untoched
  - When tested, move on to next region to be parallelized until target speedup is achieved
- Let me insist: writing correct, fast, parallel code is hard
  - Data race conditions, deadlocks, false sharing, overhead, ...
- We will discuss some potential issues and bottlenecks

First steps

- Directives:
  - Syntax: `#pragma omp <construct> [<clause> [<clause>]]`
  - Most constructs apply to structured blocks
  - One entry point, one exit point
- Routines (some examples):
  - `omp_set_num_threads( int nthreads );`
  - `int id = omp_get_num_threads();`
  - `int id = omp_get_thread_num();`
- Environment variables (an example):
  - `export OMP_NUM_THREADS=4; ./program.x`

Hello world! I'm thread X!

Exercise 1b

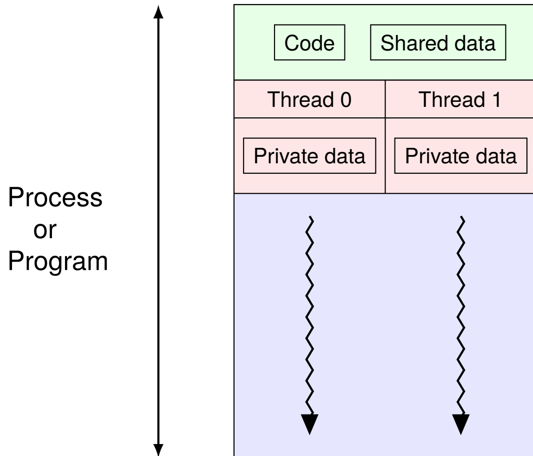
- Extend exercise 1 (below) so that 4 threads execute the parallel region and each of them prints also its thread id.

```
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"

int main( void )
{
    #pragma omp parallel
    printf("Hello world!\n");

    return 0;
}
```

Hints:

- `#pragma omp parallel num_threads(...)`
- `omp_get_num_threads()`
- `omp_set_num_threads(...)`
- `omp_get_thread_num(...)`

Variable Scope



Exercise 2 (axpy.c)

- Use the `#pragma omp parallel` construct to parallelize the code below so that 4 threads collaborate in the computation of *z*. Pay attention to shared vs private variables!

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i, N = 10;
    double x[N], y[N], z[N], alpha = 5.0;

    for( i = 0; i < N; i++ ) {
        x[i] = i;
        y[i] = 2.0*i;
    }

    for(i = 0; i < N; i++)
        z[i] = alpha * x[i] + y[i];
    // Print results. Should output [0, 7, 14, 21, ...]
    return 0;
}
```

Hints:

- `#pragma omp parallel num_threads(...)`
- `omp_set_num_threads(...)`
- `omp_get_num_threads(...)`
- `omp_get_thread_num(...)`
- Challenge: split iterations of the loop among threads