FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



Lecture 8. Introduction to OpenMP (Part 1)

Dr. Bazargul Matkerim, Professor-Researcher

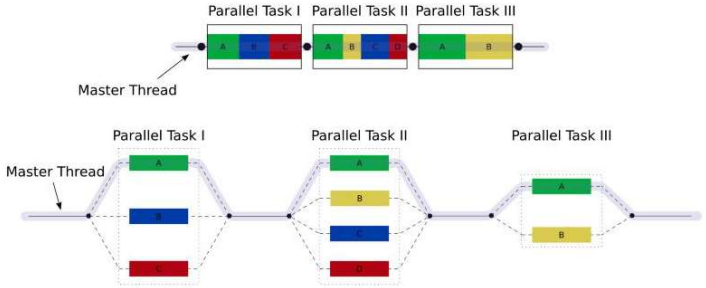
Outline

1. **What is OpenMP**
   API, history, processes and threads

2. **Programming and execution model**
   shared memory, fork-join

3. **Parallel region and work-sharing**
   directives, number of threads, for loop

4. **Data scope, reduction, scheduling**
   private, reduction(+:sum), schedule



Source: Fork–join model, commons.wikimedia.org (CC BY 3.0)

What is OpenMP

Open specifications for **M**ulti **P**rocessing

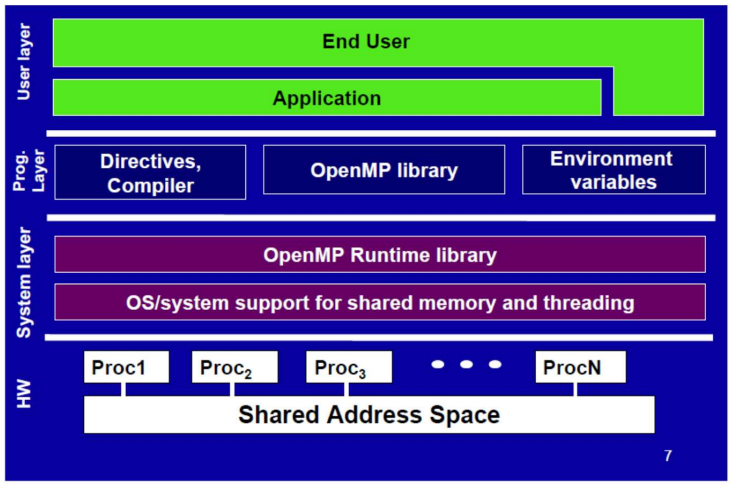
An **API** for multi-threaded, shared memory parallelism

Components:

- compiler directives
- runtime library routines
- environment variables

Portable: **C/C++ and Fortran**, Unix/Linux and Windows

History: 1997 first version (Fortran), 3.1 in 2011



Source: Lecture 13 material “Introduction to OpenMP (Part 1)”, slide 4

Processes and Threads

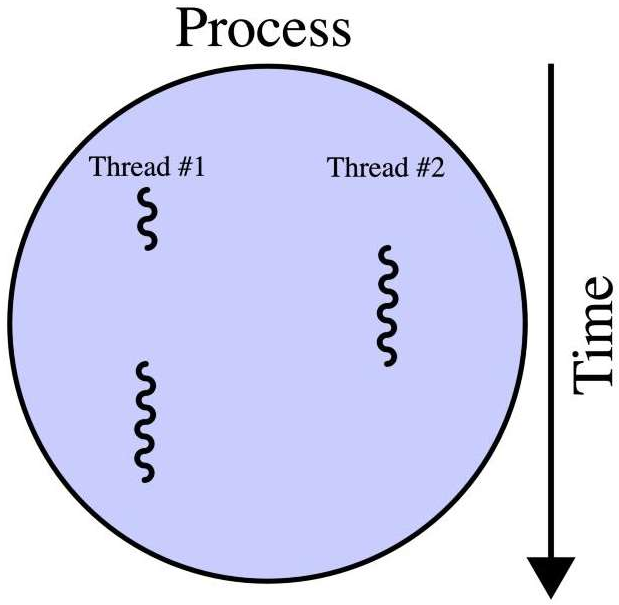
Process: a program in execution, own address space

Thread: the smallest unit of processing that can be scheduled

Threads of a process:

- share the address space (global and dynamic data)
- have their own stack, registers, program counter

Threads communicate through **shared variables**



Source: Multithreaded process, commons.wikimedia.org (CC BY-SA 3.0)

OpenMP Programming Model

Shared memory, thread-based parallelism

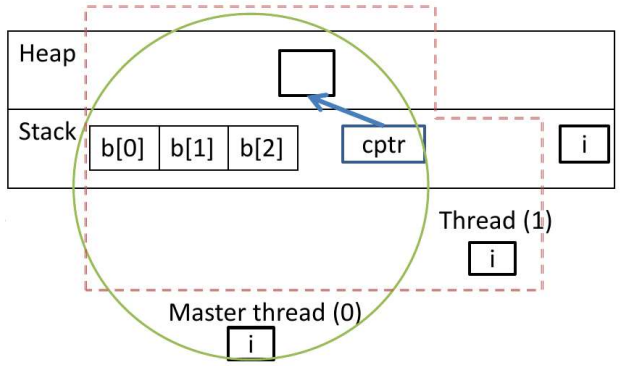
Explicit parallelism: the programmer has full control

Compiler directive based

OpenMP is not:

- for distributed-memory systems
- required to check data races or deadlocks
- guaranteed to be the most efficient

Memory model: threads may **cache** their data



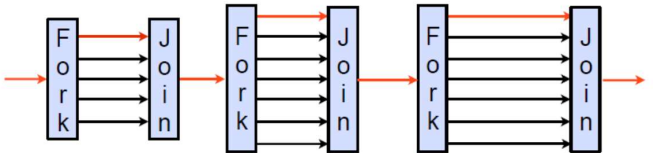
Source: Lecture 13 material “Introduction to OpenMP (Part 1)”, slide 28

Fork-Join Parallelism

Program starts with the **master thread**

Parallel region: **FORK**
- a team of threads is created
- master gets thread id 0

End of region: **JOIN**
- threads synchronize and terminate
- only the master thread continues



Source: Lecture 13 material “Introduction to OpenMP (Part 1)”, slide 10

Master thread is shown in red

OpenMP Code Structure and Directives

```
#include <stdio.h>
#include "omp.h"
int main() {
  #pragma omp parallel
  {
    int id = omp_get_thread_num();
    printf("Hello from %d\n", id);
  }
}
```

Compile: gcc -fopenmp hello.c Run: OMP_NUM_THREADS=4 ./a.out

Source: Lecture 13 material, slides 12–15

Format: #pragma omp directive [clauses]

Case sensitive

Only one directive name per directive

Applies to one **structured block**

Long lines continue with a backslash

Parallel Region and Number of Threads

```
#pragma omp parallel [clause list]
   if (scalar_expression)
   num_threads (integer_expression)
   private (list)    shared (list)
   firstprivate (list)
   default (shared | none)
```

Number of threads (in order of precedence):

1. if clause 2. num_threads clause
3. omp_set_num_threads()
4. OMP_NUM_THREADS 5. default: number of cores

Example

```
#pragma omp parallel num_threads(4) \
        private(tid)
{
  tid = omp_get_thread_num();
  printf("Hello from %d\n", tid);
}
```

Threads are numbered from **0** (master) to **N – 1**

Work-Sharing: the for Directive

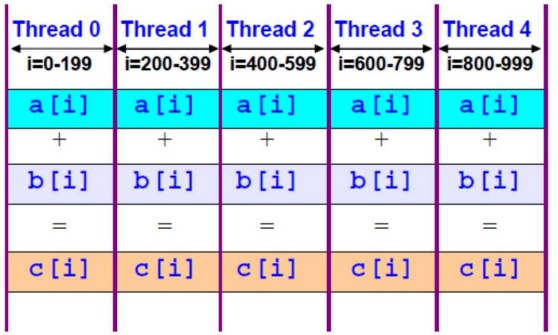
Work-sharing **divides** the work, does not create new threads

Types:

- for: loop iterations (data parallelism)
- sections: concurrent tasks
- single: one thread only

Implicit **barrier** at the end of for

```
#pragma omp parallel for
for (i = 0; i < 1000; i++)
    c[i] = a[i] + b[i];
```



Source: Lecture 13 material “Introduction to OpenMP (Part 1)”, slide 27

Data Scope Clauses

shared(list): one copy for all threads (default)

private(list): own copy, **not initialized**

firstprivate(list): own copy, **initialized** from the master

lastprivate(list): value of the **last iteration** is kept

The loop index of parallel for is private

```
int j;
#pragma omp parallel for private(j)
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++)
    a[i][j] = i + j;
```

Without private(j) all threads would share j: a race condition

Reduction

```
#pragma omp parallel for reduction(+:sum)
for (i = 0; i < N; i++) {
    x = (i + 0.5) * dx;
    sum += 4.0 / (1.0 + x*x);
}
pi = sum * dx;
```

Each thread gets a **private copy**

Copies are combined with the operator at the end

Example: π

$$\pi = \int_0^1 4/(1+x^2)\,dx \approx \sum_{i=0}^{N-1} [4/(1+x_i^2)]\,\Delta x$$

Operator	Initial Value
+	0
*	1
-	0
&	~0
|	0
^	0
&&	1
||	0

Source: Lecture 13 material “Introduction to OpenMP (Part 1)”, slide 37

Scheduling of Loop Iterations

```
schedule(scheduling_class[, parameter])
```

static: equal chunks, decided **in advance**

dynamic: next chunk when a thread is **free**

guided: chunk size **decreases**

runtime: from OMP_SCHEDULE

nowait: no barrier at the end of the loop

Thread	0	1	2	3
*no chunk**	1-4	5-8	9-12	13-16
chunk = 2	1-2 9-10	3-4 11-12	5-6 13-14	7-8 15-16

Source: Lecture 13 material “Introduction to OpenMP (Part 1)”, slide 42

16 iterations, 4 threads, static scheduling