Introduction to OpenMP

Matic Brank

LECAD laboratory, FS UL

Outline

What is OpenMP?

- Standard programming model for shared memory parallel programming
- Portable across all shared-memory architectures
- It allows incremental parallelization
- Compiler based extensions to existing programming languages
- Fortran and C/C++ binding

Outline

What is OpenMP?

OpenMP -> Open specifications for Multi Processing

- API for shared-memory parallel computing
- Open standard for portable and scalable parallel programming
- Flexible and easy to implement
- Specification for a set of compiler directives, library routines and environment variables
- Designed for Fortran and C/C++

Ownership and timeline

Ownership

- OpenMP ARB (Architecture review board)
  - Its mission is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable
  - Jointly defined by a group of major computer hardware and software vendors and major parallel computing user facilities (such as AMD, Intel, IBM, HP, Fujitsu, Microsoft,…)

Release history

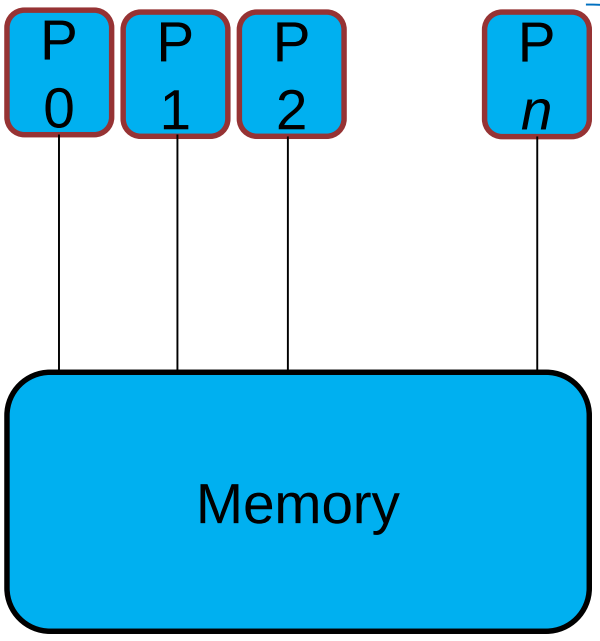
- 1997 OpenMP - Fortran 1.0
- 1998 OpenMP - C/C++ 1.0
- 1999 OpenMP - Fortran 1.1
- 2000 OpenMP - Fortran 2.0
- 2002 OpenMP - C/C++ 2.0
- 2005 OpenMP 2.5
- 2008 OpenMP 3.0
- 2013 OpenMP 4.0
- 2018 OpenMP 5.0 stable release
- 2020 OpenMP 5.1 release

OpenMP – Main terminology

- **OpenMP thread**: a running process specified by OpenMP
- **Thread team**: a set of threads which cooperate on a task
- **Master thread**: Main thread which coordinates the parallel jobs
- **Thread safety**: This term refers to correct execution of multiple threads
- **OpenMP directive**: OpenMP line of code for compilers with OpenMP
- **Construct**: an OpenMP executable directive
- **Clause**: controls the scoping of the variables during execution

OpenMP – Programming model

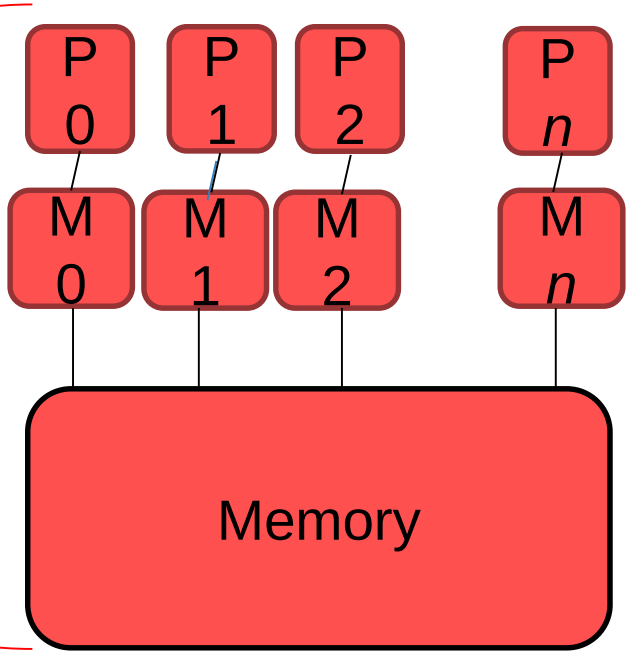
- Primarily designed for shared memory multiprocessors



A canonical shared memory architecture

All of the processors are able to directly access all of the memory in the machine through a logically direct connection

Each processor in the system is only capable of directly addressing memory (M0-M*n*) that is physically associated with him

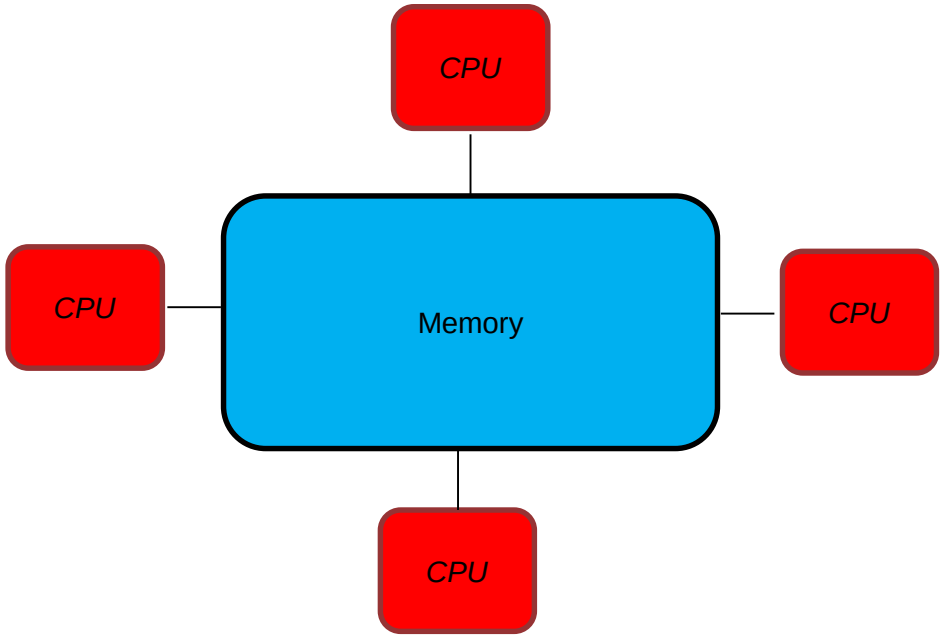


A canonical message passing (non-shared memory) architecture

OpenMP – Programming model

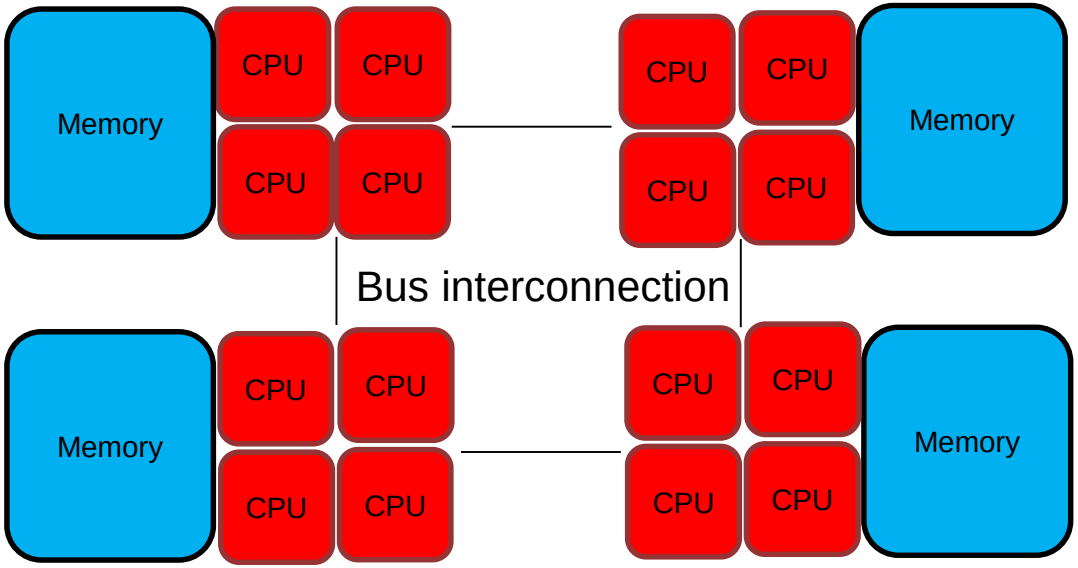
- NOTE: Cache-Coherency (cc) ▫ To ensure cache consistency (i.e. local cache has the most up-to-date copy of a shared memory resource), cache coherency protocols are implemented on modern systems.

- Shared memory architecture



UMA

Uniform Memory Access, SMP (Symmetric Multi Processing)
The latency to access any address in the logical memory space is the same for each CPU

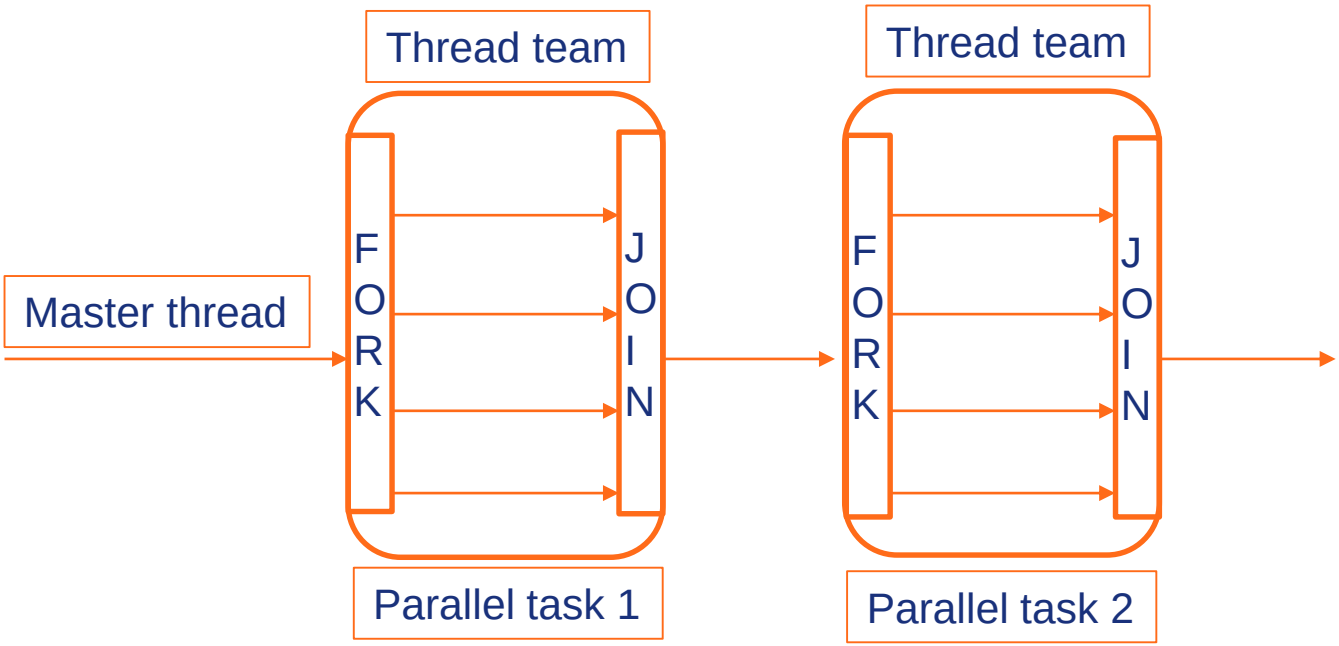


NUMA

Non-Uniform Memory Access
The latency to access any address in the logical memory space is determined by the physical distance from the CPU.

OpenMP – Execution model

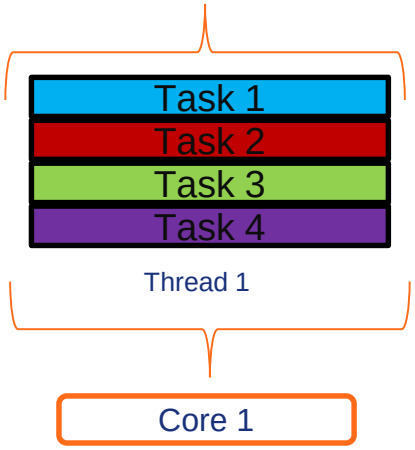
- Thread based parallelism
- Compiler directive based parallelism
- Explicit parallelism
- Fork-Join model
- Dynamic Threads
- Nested parallelism



OpenMP – Execution model

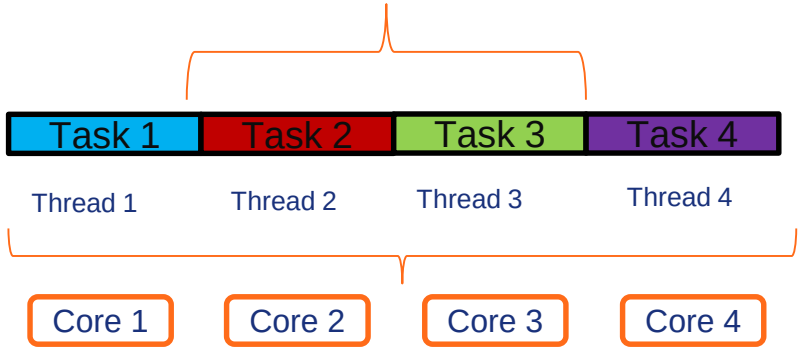
- Thread based parallelism

Single threading



Execution time = $\sum_{i=1}^{4} Task_i$

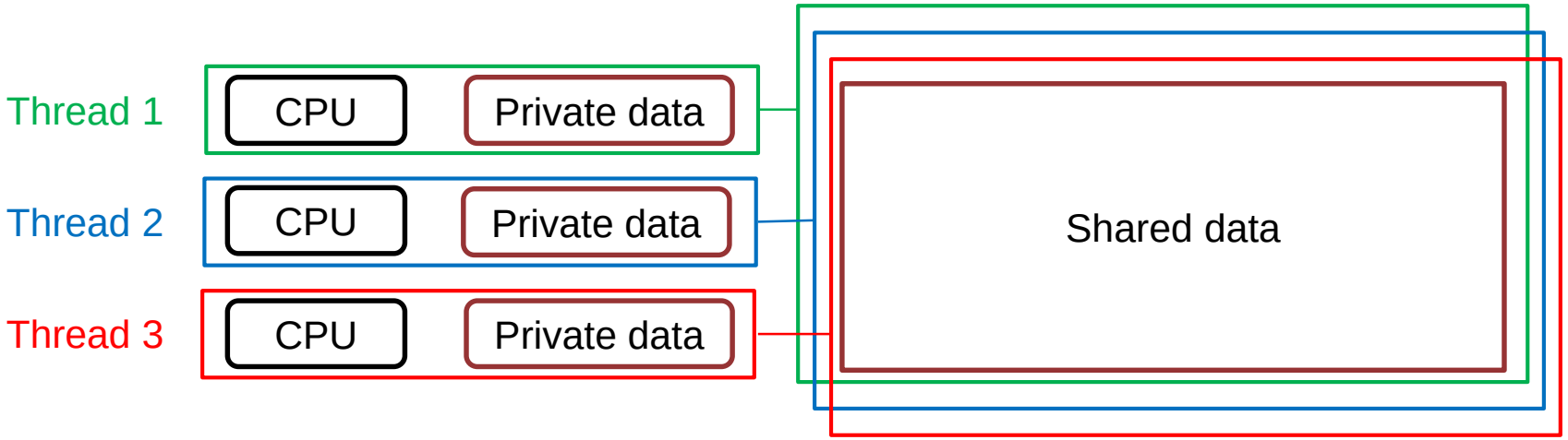
Thread based parallelism



Execution time = $MAX(Task_i)$

OpenMP – Memory model

- All threads have access to the same memory
- Threads can share data with other threads, but also have private data
- Threads can be synchronized
- Threads cache their data



OpenMP – Memory model

- Compiler directives and Clauses: appear as comments and execute when appropriate OpenMP flag is specified
  - Parallel construct
  - Work-sharing constructs
  - Synchronization constructs
  - Data attribute clauses
- C/C++ OpenMP comment:

```
#pragma omp directive-name [clause[clause]…]
```

OpenMP – Memory model

- Compiling

	Compiler	Flag
Intel	icc (C) icpc (C++) Ifort (Fortran)	**-openmp**
GNU	gcc (C) g++ (C++) g77/gfortran (Fortran)	**-fopenmp**
PGI	pgcc (C) pgCC (C++) pg77/pgfortran (Fortran)	**-mp**

- Example of command in terminal:

```
gcc -fopenmp -o executable foo.c
```

Compiler `gcc` compiles file `foo.c` with OpenMP flag into executable file named `executable`

- Notes for GCC compiler:
  - From GCC 6.1, OpenMP 4.5 is fully supported for C and C++.
  - From GCC 7.1, OpenMP 4.5 is partially supported for Fortran.
  - From GCC 9.1, OpenMP 5.0 is partially supported for C and C++

Note: For full list of vendors and compilers refer to https://www.openmp.org//resources/openmp-compilers-tools/

OpenMP – Main terminology

- Purpose of runtime functions is to manage the parallel processes
  - `omp_set_num_threads(n)` – set the desired number of threads
  - `omp_get_num_threads()` – returns the current number of threads
  - `omp_get_thread_num()` – returns the ID of this thread
  - `omp_in_parallel()` – return .true. if inside parallel region
- Correct usage in code:
  - For C/C++ add `#include<omp.h>` to the code
  - For Fortran add `use omp_lib`

Note: For full list of OpenMP runtime functions refer to https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-functions?view=vs-2019

OpenMP – Environment variables

- Purpose of environment variables is to control the execution of parallel program at runtime. These variables are not specified in the code itself but in the environment in which the parallel program is executed.
  - `OMP_NUM_THREADS` – specifies the number of threads to use
  - `OMP_PLACES` – specifies on which CPUs the threads should be placed
  - `OMP_DISPLAY_ENV` – show OpenMP version and environment
- Correct usage in code:
  - Environment csh/tcsh: `setenv OMP_NUM_THREADS n`
  - Environment ksh/sh/bash: `export OMP_NUM_THREADS=n`

Note: For full list of OpenMP environment variables for GCC compiler refer to https://gcc.gnu.org/onlinedocs/libgomp/Environment-Variables.html

OpenMP – Parallel construct

- Parallel construct is the fundamental construct in OpenMP
  - Every thread executes the same statements which are inside the parallel region simultaneously
  - At the end of the parallel region there is an implicit barrier for synchronization

C/C++:

```
#pragma omp parallel
{
…
}
```

Fortran:

```
!$omp parallel
…
!$omp end parallel
```