

Updated on 09/10/2025

Sign up

OpenMP Training

3 days (21 hours)

Overview

Our OpenMP training course will enable you to discover and master this powerful API dedicated to parallel programming on shared memory architectures. You'll learn how to take full advantage of multi-core processors to make your applications faster, more efficient and ready for intensive computing.

In this course, you'll start with the basics of parallelism: creating threads, managing memory sharing, and the first `#pragma` directives for structuring your code into simple parallel regions.

You'll then learn how to parallelize loops, manipulate shared or private variables, and aggregate results without errors using the reduction directive. The aim: to write safe, high-performance parallel code.

We'll also cover dynamic scheduling, fine-tuned synchronization with `critical`, `atomic` and `barrier`, and the creation of independent tasks via `sections` and `task`.

As with all our training courses, this one will be presented with the latest version of [OpenMP](#).

Objectives

- Understand the principles of shared-memory parallelism and OpenMP's use cases for accelerating CPU processing.
- Use OpenMP directives to create parallel regions and make efficient use of processor processor cores
- Parallelize calculation loops with `parallel for`, manage shared and private variables, and secure aggregations with `reduction`
- Control thread synchronization with `critical`, `atomic` and `barrier` directives, and structure your into independent sections

- Control work scheduling between threads with `schedule`, and optimize performance according to workload
- Implement dynamic tasks with `task` and `taskwait` to parallelize non-uniform
- Analyze performance with `omp_get_wtime`, identify critical points and avoid common concurrency errors
- Develop an end-to-end mini-project, applying the concepts learned to a real-life, compute-intensive case.
- intensive case study
- Adopt good OpenMP development practices to produce clear, efficient and easily maintainable code.

Target audience

- Full-stack developer
- Front-end developer

Prerequisites

- Good command of the C or C++ language
- Notions of compilation and use of the command line
- Basic knowledge of algorithms and manipulation of tables or data structures

OUR OpenMP TRAINING PROGRAM

Introduction to parallelism and OpenMP

- Modern processors have several cores: to take full advantage of the hardware, calculations must be carried out in parallel.
- Parallelism improves performance, especially for heavy processing (scientific calculations, AI, simulations...).
- Two main types of parallelism :
 - Shared memory: all threads share the same RAM (OpenMP)
 - Distributed memory: each process has its own memory (MPI)
- OpenMP (Open Multi-Processing) is an API for C, C++ and Fortran.
- Allows parallelization without manual thread management (vs `pthread` or `std::thread`).
- Based on compiler directives (`#pragma omp`), it's easy to integrate into existing code.
- Compatible with GCC, Clang, Intel, MSVC...
- Compiling: `gcc` or `clang` with `-fopenmp` option

First steps with threads

- Basic directive
- Use `"omp_get_thread_num()"` to find out the thread identity.
- `"omp_get_threads_num()"`: number of threads in use
- `"omp_set_num_threads(n)"`: set the number before the parallel region.

- Objective: display a message from each thread
- Observation: threads do not execute in order

Parallelizing loops

- Allows you to distribute loop iterations between threads:
- By default, some variables are shared (global), others private (local to each thread).
- Apply a transformation to a large array in parallel
- Measure time savings with "omp_get_wtime()".

Reducing and aggregating results

- If several threads modify a shared variable (e.g. sum), this causes a race condition.
- OpenMP directive for collision-free aggregation of results
- Compare sequential version, naive parallel version (bug), version with reduction
- Analyze errors linked to shared variables

Task control and synchronization

- #pragma omp barrier: waits for all threads to arrive
- #pragma omp critical: protected section, only one thread at a time
- #pragma omp atomic: faster but limited atomic operation
- Simulate 3-stage processing (read, calculate, save) with sections

Performance management and scheduling

- Work distribution control :
 - Static: fixed shares
 - Dynamic: on-the-fly scheduling
 - Guided: dynamic but decreasing
- Test different scheduling strategies on a loop with varying processing times

Parallelizing a real project

- Sequential version
- Dependency analysis
- Progressive parallelization with OpenMP
- Optimization: avoid costly memory accesses
- Parallelizing a program of your choice (image filtering, sorting, statistics, etc.)
 - Identify critical loops
 - Adding directives
 - Measuring performance
 - Avoiding race conditions

Best practices and prospects

- Poorly declared variables (shared vs. private)
- Concurrent access without critical or reduction
- Too many threads = CPU overload
- Always measure performance
- Progressive parallelization
- Promote independent processing
- Comparison with MPI, CUDA, OpenACC
- OpenMP + SIMD
- Profiling with gprof, perf, Intel VTune

Companies concerned

This course is aimed at both individuals and companies, large or small, wishing to train their teams in a new advanced computer technology, or to acquire specific business knowledge or modern methods.

Positioning on entry to training

Positioning at the start of training complies with Qualiopi quality criteria. As soon as registration is finalized, the learner receives a self-assessment questionnaire which enables us to assess his or her estimated level of proficiency in different types of technology, as well as his or her expectations and personal objectives for the forthcoming course, within the limits imposed by the selected format. This questionnaire also enables us to anticipate any connection or security difficulties within the company (intra-company or virtual classroom) which could be problematic for the follow-up and smooth running of the training session.

Teaching methods

Practical training: 60% hands-on, 40% theory. Training material distributed in digital format to all participants.

Organization

The course alternates theoretical input from the trainer, supported by examples, with brainstorming sessions and group work.

Validation

At the end of the session, a multiple-choice questionnaire verifies the correct acquisition of skills.

Certification

A certificate will be awarded to each trainee who has completed the entire course.