Introduction

Let's take a look at the concepts to look forward to in this chapter!

WE'LL COVER THE FOLLOWING

- What you'll learn in this chapter

What you'll learn in this chapter

Concurrency and Parallelism are core aspects of any modern programming language. Before C++11 there was no standard support in the language for threading - you could use third - party libraries or System APIs. Modern C++ started to bring more and more necessary features: threads, atomics, locks, `std::async` and futures.

C++17 gives us a way to parallelise most of the standard library algorithms. With a powerful and yet straightforward abstraction layer, you can leverage more computing power out of a machine.

In this chapter you'll learn:

- What's on the way for C++ regarding parallelism and concurrency
- Why `std::thread` is not enough
- What are the execution policies
- How to run parallel algorithms
- Which algorithms were parallelized
- What the new algorithms are
- Expected performance of parallel execution
- Examples of parallel execution and benchmarks

If we look at the computers that surround us, we can observe that most of them are multi-processors units. Even mobile phones have four or even eight cores. Not to mention graphics cards that are equipped with hundreds (or

even thousands) of small computing cores.

The trend towards multicore machines was summarised perfectly in a famous article by Herb Sutter The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. The article appeared in 2006.

A glance around is all it takes to see that this trend is not slowing down.

While for a simple application there's probably no need to use the full computing capacity of your machine, there are applications which do require just that. Gaming, fast, responsive apps, graphics processing, video/music editing, data processing, financial, servers and many more types of systems. Spawning threads on a CPU and processing tasks concurrently is one way to achieve that.

With C++11/14 we've finally got threading into the standard library. You can now create instances of `std::thread` and not just depend on third-party libraries or a system API. What's more, there's also async processing with futures (`std::async`).

Multithreading is a significant aspect of modern C++. In the C++ Committee, there's a separate group - "SG1, Concurrency" that works on bringing more features like this to the standard.

What's on the way?

- Coroutines
- Atomic Smart pointers
- Transactional Memory
- Barriers
- Tasks blocks
- Parallelism
- Compute
- Executors
- Heterogeneous programming models support

As you can see, the plan is to expose as much of your machine's computing power as possible, directly from The Standard Library.

Let’s get started.