Programming Parallel Computers

Introduction

These are the lecture notes of the Aalto University course **CS-E4580 Programming Parallel Computers**. The exercises and practical instructions are available in the **exercises tab**. There you will also find an open online version of this course that you can follow if you are self-studying this material!

Why parallelism?

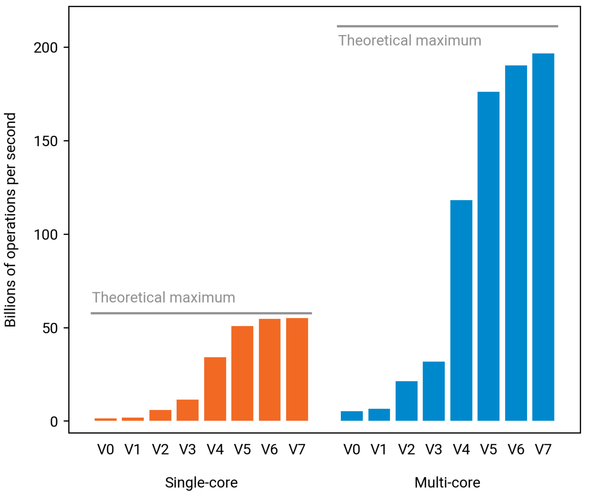
All modern computers have massively parallel processors. The CPU of a normal desktop or laptop computer may easily have **hundreds** of arithmetic operations in progress simultaneously:

- there are multiple **CPU cores**
- each core has multiple **execution units** that can be used simultaneously
- each execution unit can perform **wide vector operations**, and
- the execution units are **pipelined** so one does not need to wait for the previous instruction to finish before starting the next one.

However, the programmer has to be aware of these opportunities of parallelism and know how to exploit them. Otherwise, your program may easily **waste more than 99% of the computing power** of your CPU.

Programming modern CPUs

The following figure highlights the importance of exploiting parallelism in practice, on a normal desktop computer with a modern 4-core Intel CPU. Here "V0" is a **baseline solution** that would look perfectly reasonable if we were programming an old-fashioned sequential computer. However, on a modern CPU it turns out that we are using only 2% of the theoretical single-core performance, and only 0.6% of the theoretical multi-core performance. We are leaving 99.4% of the computing power unused!



We will walk through this example in our lectures, and develop a sequence of faster solutions (V1, V2, …) that make a much better use of the vast computing resources that we have in a modern computer; V7 is **42** times faster than the baseline solution already on a single core, and **151 times faster** when we use all 4 cores.

As we can see, achieving good performance on a modern CPU is **much more than simply using multiple threads!**

Programming modern GPUs

Besides the CPU, another massively parallel resource that we have nowadays in virtually any modern computer is the GPU, the graphics processing unit. While originally intended for computations related to graphics processing, modern GPUs have evolved into general-purpose processors that can be used in all kinds of computations.

However, while GPUs could be used in any kind of computations, a typical C++ program uses exactly 0% of the power of the GPU. The programmer needs to explicitly write code for the GPU. As we will see, this does not need to be difficult — but one has to be aware of some key concepts and tools.

Course idea and prerequisites

The main theme of this course is that exploiting **parallelism is necessary** in any kind of performance-critical applications nowadays, but **it can also be easy**.

Our goal is to **show the good parts**: how to get the job done, with minimal effort, in practice. We aim at making parallel programming something that one does casually as a natural part of everyday computer programming.

However, we will also put a lot of emphasis on developing **understanding** of performance-engineering for modern hardware. In the advanced material we will look at the assembly code produced by the compiler, discuss how the CPU executes it, and learn how to **predict** the performance of a piece of code. We aim at demystifying hardware — or at least the parts that are most relevant from the perspective of getting good performance in practice.

We will assume a good understanding of computer programming, algorithms and data structures, and a working knowledge of either C or C++ programming language. However, we do not assume any knowledge of parallel computing, multi-threading, or GPUs.