Compiler Optimizations for Parallel Programs

Authors: Andrés Morilla (xmoril01), Tomáš Kučma (xkucma00)

Optimizations are an important part of practically every compiler. Unlike application-specific optimizations, which have to be done manually, often requiring extensive expertise and non-trivial time investment, compiler optimizations are generic enough to be automatically applicable in a wide variety of cases while having a significant impact.

This makes them one of the simplest ways to increase computational performance at minimal cost. Therefore, how well these optimizations can be performed by the compiler is an important factor when choosing and evaluating compilers. However, from the perspective of compiler development, implementing these automatic optimizations is impeded by various difficulties. Many problems require parallelism to be solved in a reasonable time. This need is highlighted even more because, in recent years, the growth of core frequencies in new CPUs has slowed down significantly. One of the ways that this is compensated for is by increasing the number of cores, which require parallelism for maximal utilization. This poses a challenge to compilers, as they are generally designed with sequential execution in mind.

Abstractions are used to ensure compiler compatibility with parallel programs — for example, by representing parallel parts of a program with a single function call. This obfuscates the actual code and limits the possibilities for automatic optimizations. This presentation will explain how compiler parallelism awareness can help us counteract these issues, where it is unnecessary, and what particular challenges parallelism awareness brings. The specifics of the problem will be explained using the intermediate representation code of the LLVM compiler toolchain — LLVM-IR.