Topic: Parallelism in Modern Compilers (abstract)

Author: Ing. Vojtěch Nikl
Specialization: Parallel systems, high-performance computing, system architectures
Ph.D. Supervisor: Prof. Ing. Václav Dvořák, DrSc.
Course: Modern Theoretical Computer Science
Academic Year: 2014/2015

Parallel compilers help accelerate the software development process by compiling code quickly and efficiently. A large number of computationally intensive problems can be divided into smaller problems and each of them can be given to an independent processor with shared or independent resources. Various techniques have been developed like vectorization and dependency graph analysis in order to extract parallelizable segments in a piece of code written for a serial compiler.

A serial compiler typically consists of these sections: lexical analyzer, which converts input text into a stream of tokens (lexeme) and constructs a symbol table. Syntax analyzer checks whether the input adheres to rules specified by the grammer. If the input is correct, the intermediate code is generated, optimized and converted into native code of the current machine. Each of these parts can internally run in parallel and shorten the running time of a compilation, however the main question is what kind of pieces a program should be divided into and how these pieces may be rearranged. This involves granularity, level, and degree of parallelism and analysis of the dependencies among the candidates of parallel execution.

Vectorization exploits parallelism at granularity level of machine instruction. A computer program is converted from a scalar implementation, which processes a single pair of operands at a time, to a vector implementation, which processes one operation on multiple pairs of operands at once. Vectorization is typically applied on loops.

Other sources of parallelism are loop interchange and scalar expansion. Loop interchange switches the nesting order of two loops in a perfect nest and moves the parallelism to the innermost loop level. Scalar expansion replaces scalars by a temporary array, which allows to distribute a loop around the statement blocks.

One of the main barries to parallelism are dependencies. A program is a collection of statements, the ordering and scheduling of which depends on dependence constraints. Dependencies are broadly classified into two categories: data dependencies, where statements compute data that are used by other statements, and control dependencies, which arise from the ordered flow of control in a program. The result of this analysis is a dependency graph, which helps divide the program into separate parts and determing the processing order of these parts. It is possible to derive an evaluation order or the absence of an evaluation order that respects the given dependencies from the dependency graph.