Context: Instruction Scheduling and Register Allocation

Two of the most fundamental tasks in a compiler's code generation backend are instruction scheduling and register allocation. Instruction scheduling involves transforming the control- and data-flow information of the input program into a partial order of instructions. Register allocation, on the other hand, involves mapping the values used by the program into CPU registers for fast execution. These tasks are typically performed sequentially, with instruction scheduling being done first and followed by register allocation. However, recent research has shown that integrating these two tasks can lead to better code generation results.

Scheduling Heuristics

Various scheduling techniques have been proposed in the literature, each with its own advantages and disadvantages. In a basic block, scheduling attempts to minimize the number of guarded scanner instructions by mapping them to instructions that can fit. This involves finding code points where instructions can be placed. The resulting schedule can then be used for register allocation.

Rescheduling

The approach can use an existing instruction scheduler to compute the initial schedule. It may be beneficial to modify the scheduler to optimize for the register allocator. For example, the register allocator may attempt to hoist instructions that end live ranges to eliminate conflicts. The resulting instruction stream can then be scheduled with the register allocator.

Register Allocation

Register allocation attempts to map the values used by the program into CPU registers for fast execution. This involves assigning registers to variables and inserting spill code when necessary. The register allocator may need to reschedule the instructions to reduce register pressure. It optimistically assumes that the register allocator will reschedule to reduce register pressure and avoid spilling. However, as discussed above, we believe that some spills may be profitable if they allow the code to be scheduled more aggressively.

Implementation Status and Evaluation

The rescheduling register allocator will be implemented by extending the LLVM compiler framework. We will evaluate the performance of the generated code and compare it against other scheduling techniques including IPS. At the time of printing of this poster, the implementation is at an early stage, but we plan to present it at the current status.

Challenges and Research Directions

One challenge is to develop an integrated code generator that can handle the interaction between instruction scheduling and register allocation. Another challenge is to develop a unified framework for code generation that can handle both scheduling and register allocation. Finally, we need to investigate the implications of our code generator on other aspects of the compiler.

References


Acknowledgments

This work is supported by the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) grant F1390 (Optimality for Explicitly Parallel Programs).