Register Allocation and Instruction Scheduling in Unison

Roberto Castañeda Lozano
Swedish Institute of Computer Science
School of ICT, KTH Royal Institute of Technology, Sweden
rcas@sics.se

Mats Carlsson
Swedish Institute of Computer Science
matsc@sics.se

Gabriel Hjort Blindell
Christian Schulte
School of ICT, KTH Royal Institute of Technology, Sweden
Swedish Institute of Computer Science
{ghb,cschulte}@kth.se

Abstract

This paper describes Unison, a simple, flexible, and potentially optimal software tool that performs register allocation and instruction scheduling in integration using combinatorial optimization. The tool can be used as an alternative or as a complement to traditional approaches, which are fast but complex and suboptimal. Unison is most suitable whenever high-quality code is required and longer compilation times can be tolerated (such as in embedded systems or library releases), or the targeted processors are so irregular that traditional compilers fail to generate satisfactory code.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—compilers, code generation, optimization; D.3.2 [*Programming Languages*]: Language Classifications—constraint and logic languages; I.2.8 [*Artificial Intelligence*]: Problem Solving, Control Methods, and Search—backtracking, scheduling

Keywords combinatorial optimization; register allocation; instruction scheduling

1. Introduction

Register allocation (assigning program variables to processor registers or memory) and instruction scheduling (reordering processor instructions to increase throughput) are central problems in optimizing compilers. Given the hard combinatorial nature of these problems and their interdependencies, traditional compilers resort to heuristic algorithms and phase decoupling, which trades code quality and flexibility for compilation speed.

This paper describes *Unison* [1, 3, 4], a simple, flexible, and potentially optimal software tool that performs integrated register allocation and instruction scheduling using combinatorial optimization. Unison formalizes both problems as combinatorial models and solves them simultaneously, taking into account their interdependencies, considering the involved trade-offs, and exploring the full solution space to deliver optimal assembly code. Unlike earlier combinatorial approaches, Unison captures register allocation in its full scope while robustly scaling to medium-size functions. A detailed comparison with related approaches is available in a survey by Castañeda Lozano and Schulte [2, Sect. 5].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CC'16, March 17–18, 2016, Barcelona, Spain
© 2016 ACM. 978-1-4503-4241-4/16/03...$15.00
http://dx.doi.org/10.1145/2892208.2892237

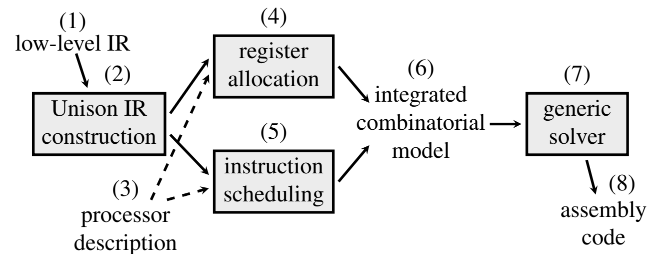

Figure 1. Unison's approach.

Approach. Unison approaches register allocation and instruction scheduling as shown in Figure 1. A low-level intermediate representation (IR) of a function, where processor instructions are already selected, is taken as input (1). The input function is transformed into Unison IR (2), which exposes the structure of the program and the multiple decisions involved in the problem. Taking the Unison IR of the function and a description of the processor (3), a combinatorial model of each of the problems is formulated (4,5) consisting of variables representing the problem decisions, program and processor constraints over the variables, and a cost function to be minimized. The combinatorial models are then integrated into a single model (6) which is solved by a generic solver (7), delivering assembly code (8) that is potentially optimal.

2. Unison IR

Unison operates on a custom IR (Unison IR), which exposes underlying structures of the program (such as live ranges) that are key to register allocation and instruction scheduling, and problem decisions such as whether to spill a certain program variable (called *temporary* at Unison's level). Unison IR has the following distinguishing features:

linear static single assignment form (LSSA) LSSA [3] is a stricter form of static single assignment (SSA) form in which temporaries are local to basic blocks, and relations across blocks are captured by a generalization of the ϕ-congruence. This form is key to representing liveness in the combinatorial model.

optional copies Unison IR includes optional copy instructions [3] which can be inactivated or implemented by alternative processor instructions (such as register-to-register moves, store, and

load instructions) to support different register allocation decisions.

alternative temporaries Unison IR allows instructions to use alternative temporaries that hold the same value. Alternative temporaries [4] improve the capabilities of register allocation by enabling two key optimizations: spill code optimization and ultimate coalescing.

Unison IR can be easily constructed from a low-level, SSA-based IR with processor instructions.

3. Combinatorial Model

The core of Unison is a combinatorial model of register allocation and instruction scheduling. Combinatorial models are formed by variables (typically Boolean and integer variables representing the problem decision), constraints over the variables (representing relations among the problem decisions that must hold in any solution), and a cost function (a function of the variables to be minimized).

Register allocation. The register allocation model includes variables to decide which register is assigned to each temporary (including stack locations which are just modeled as any other register bank), which temporaries are used by each instruction, which copies are activated, and which instruction implements each of the active copies. The use of LSSA makes it possible to model a basic block's register assignment as a rectangle packing problem where each rectangle represents a live range, similarly to the approach of Pereira and Palsberg [6]. Register assignment is extended to entire functions by constraints that assign congruent temporaries to the same registers. Additional constraints ensure that the temporaries used and defined by active instructions are assigned to compatible registers, and that the definers of used temporaries are activated.

Instruction scheduling. The instruction scheduling model includes variables to decide which issue cycle is assigned to each instruction, and constraints to enforce precedences dictated by data and control dependencies and to ensure that the capacity of processor resources such as functional units is not exceeded.

Integration. The register allocation and instruction scheduling models are related by the live ranges of the temporaries. The integrated model relates each live range with the the issue cycles of the definer and user(s) of the corresponding temporary.

The cost function of the integrated model can be adjusted for speed or code size optimization. The speed cost function is the sum of the estimated execution cycles of each basic block weighted by the estimated execution frequency; the code size cost function is simply the sum of the size of each active instruction.

4. Status

Unison has been shown to be practical and effective for medium-size functions. Experiments with different benchmarks (SPECint 2006, MediaBench) show that Unison can generate code of similar quality to LLVM [5] (a traditional, state-of-the-art compiler) for simple processors such as MIPS32 and better code for more complex processors such as Hexagon V4, a very long instruction word (VLIW) processor included in Qualcomm's Snapdragon system-on-chip [7]. The tool scales to medium-size functions of up to some thousand instructions and can improve the generated code progressively as more compilation time is invested.

LLVM interface. Although designed as a standalone tool, Unison provides an interface to LLVM's `llc` code generator, where Unison IR is converted from and to `llc`'s Machine IR (MIR) in between `llc`'s `PreRA` and `PreEmit` phases. An upper bound for the cost function can be obtained by running first `llc`'s original register allocator and instruction scheduler.

Currently, Unison supports Hexagon, MIPS32, and a proprietary processor. Support for more processors can be easily gained since processor descriptions can be automatically extracted from LLVM's *TableGen* information.

Solvers. The Unison toolchain is solver-independent. The default solver is based on the constraint programming system Gecode (`www.gecode.org`), but alternative implementations are available in SICStus Prolog (`sicstus.sics.se`) and the MiniZinc modeling language (`www.minizinc.org`), which gives access to a wide variety of local search, integer programming, Boolean satisfiability and constraint solvers.

Scope and subproblems. Currently, the tool supports global register allocation with its entire range of common subproblems (register assignment, spilling, coalescing, live range splitting, multiple register banks, register packing for aliased registers, spill code optimization, and rematerialization) and local instruction scheduling for single and multiple-issue processors. Additionally, the tool integrates practical aspects related to register allocation and instruction scheduling such as calling conventions (including scheduling of related spill code) and stack handling. Furthermore, the model is flexible enough to integrate optimizations specific to certain processors such as memory operands, two-address conversion, and register bank assignment.

Future work. Future work on the combinatorial model includes extending the scope of instruction scheduling (starting from superblocks as the immediately next step), and improving the accuracy of the speed cost function in the face of unpredictable processor features like cache memories. Another potential line of work is to improve the compilation time and scalability of Unison by integrating different solving techniques. Longer-term goals include full integration with instruction selection (which often has dependencies with instruction scheduling and register allocation) and adding support for multi-objective optimization as well as other optimization goals such as energy consumption.

Acknowledgments

This research has been partially funded by LM Ericsson AB and the Swedish Research Council (VR 621-2011-6229). Mikael Almgren, Erik Ekström, Bevin Hansson, Jan Tomljanović, and Kim-Anh Tran have collaborated in the development of Unison. The authors are grateful for helpful comments from the anonymous reviewers.

References

[1] R. Castañeda Lozano. *Integrated Register Allocation and Instruction Scheduling with Constraint Programming*. Licentiate thesis. KTH Royal Institute of Technology, Sweden, 2014.

[2] R. Castañeda Lozano and C. Schulte. Survey on combinatorial register allocation and instruction scheduling. Technical report, SCALE, KTH Royal Institute of Technology & Swedish Institute of Computer Science, 2014. Archived at `arXiv:1409.7628 [cs.PL]`.

[3] R. Castañeda Lozano, M. Carlsson, F. Drejhammar, and C. Schulte. Constraint-based register allocation and instruction scheduling. In *CP*, volume 7514 of *LNCS*, pages 750–766. Springer, 2012.

[4] R. Castañeda Lozano, M. Carlsson, G. Hjort Blindell, and C. Schulte. Combinatorial spill code optimization and ultimate coalescing. In *LCTES*, pages 23–32. ACM, 2014.

[5] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *CGO*, pages 75–88. IEEE, 2004.

[6] F. M. Q. Pereira and J. Palsberg. Register allocation by puzzle solving. pages 216–226. ACM, 2008.

[7] *Hexagon V4 Programmer's Reference Manual*. Qualcomm, Aug. 2013.