Abstract. We solve the problem of integrating modulo scheduling with
instruction selection (including cluster assignment), instruction schedul-
ing and register allocation, with optimal spill code generation and schedul-
ing. Our method is based on integer linear programming. We prove that our
algorithm delivers optimal results in finite time for a certain class of archi-
tectures. We believe that these results are interesting both from a theoreti-
cal point of view and as a reference point when devising heuristic methods.

1 Introduction

Many computationally intensive programs spend most of their execution time
in a few inner loops. This makes it important to have good methods for code
generation for loops, since small improvements per loop iteration can have a
large impact on overall performance.

The back end of a compiler transforms an intermediate representation into
executable code. This transformation is usually performed in three phases: in-
struction selection selects which instructions to use, instruction scheduling maps
each instruction to a time slot and register allocation selects in which registers
a value is to be stored. Furthermore the back end can also contain various op-
timization phases, e.g. modulo scheduling for loops where the goal is to overlap
iterations of the loop and thereby increase the throughput.

It is beneficial to integrate the phases of the code generation since this gives
more opportunity for optimizations. However, this integration of phases comes
at the cost of a greatly increased size of the solution space. In previous work [6]
we gave an integer linear program formulation for integrating instruction selec-
tion, instruction scheduling and register allocation. In this paper we will show
how to extend that formulation to also do modulo scheduling for loops. In con-
trast to earlier approaches to optimal modulo scheduling, our method aims to
produce provably optimal modulo schedules with integrated cluster assignment
and instruction selection.

The remainder of this paper is organized as follows: In order to give a more ac-
cessible presentation of the integer linear programming formulation for integrated
modulo scheduling, we first give, in Section 2, an integer linear program for inte-
grated code generation of basic blocks, which is adapted from earlier work [6]. In
Section 3 we extend this formulation to modulo scheduling. Section 4 presents an algorithm for modulo scheduling, and proves that it is optimal for a certain class of architectures. Section 5 shows an experimental evaluation, Section 6 reviews some related work, and Section 7 concludes.

2 Integer Linear Programming Formulation

In this section we introduce the integer linear programming formulation for basic block scheduling. This model integrates instruction selection (including cluster assignment), instruction scheduling and register allocation.

2.1 Optimization Parameters and Variables

Data Flow Graph. The data flow graph of a basic block is modeled as a directed acyclic graph (DAG). The set $V$ is the set of intermediate representation (IR) nodes, the sets $E_1, E_2 \subset V \times V$ represents edges between operations and their first and second operand respectively. $E_m \subset V \times V$ represents data dependences in memory. The integer parameters $Op_i$ and $Outdg_i$ describe operators and out-degrees of the IR node $i \in V$, respectively.

Instruction Set. The instructions of the target machine are modeled by the set $P$ of patterns. $P$ consists of the set $P_1$ of singletons, which only cover one IR node, the set $P_2+$ of composites, which cover multiple IR nodes, and the set $P_0$ of patterns for non-issue instructions. The non-issue instructions are needed when there are IR nodes in $V$ that do not have to be covered by an instruction, e.g. an IR node representing a constant value. The IR is low level enough so that all patterns model exactly one (or zero in the case of constants) instructions of the target machine.

For each pattern $p \in P_2+ \cup P_1$ we have a set $B_p = \{1, \ldots, n_p\}$ of generic nodes for the pattern. For composites we have $n_p > 1$ and for singletons $n_p = 1$. For composite patterns $p \in P_2+$ we also have $EP_p \subset B_p \times B_p$, the set of edges between the generic pattern nodes. Each node $k \in B_p$ of the pattern $p \in P_2+ \cup P_1$ has an associated operator number $OP_{p,k}$ which relates to operators of IR nodes. Also, each $p \in P$ has a latency $L_p$, meaning that if $p$ is scheduled at time slot $t$ the result of $p$ is available at time slot $t + L_p$.

Resources and Register Sets. For the integer linear programming model we assume that the functional units are fully pipelined. Hence we can model the resources of the target machine with the set $F$ and the register banks by the set $RS$. The binary parameter $U_{p,f,o}$ is 1 iff the instruction with pattern $p \in P$ uses the resource $f \in F$ at time step $o$ relative to the issue time. Note that this allows for multiblock [9] and irregular reservation tables [15]. $R_r$ is a parameter describing the number of registers in the register bank $r \in RS$. The issue width is modeled by $\omega$, i.e. the maximum number of instructions that may be issued at any time slot.