An Integer Linear Programming Model of Software Pipelining for the MIPS R8000 Processor

Artour Stoutchinin
EE Department, University of Delaware, USA *

Abstract. In parallelizing the code for high-performance processors, software pipelining of innermost loops is of fundamental importance. In order to benefit from software pipelining, two separate tasks need to be performed: (i) software pipelining proper (find the rate-optimal legal schedule), and (ii) register allocation (allocate registers to the found schedule). Software pipelining and register allocation can be formulated as an integer linear programming (ILP) problem, aiming to produce optimal schedules. In this paper, we discuss the application of the integer linear programming to software pipelining on the MIPS R8000 superscalar microprocessor. Some of the results were presented in the PLDI96 [14], where they were compared to the MIPSpro software pipeliner. In this paper we further extend the ILP model for the MIPS R8000 by including memory optimization and present the entire model in detail.

1 Introduction

In the recent years, the concept of instruction-level parallelism played a central role in the microprocessor design of all the major CPU manufacturers. Several processors, such as the DEC Alpha 21064, the IBM Power PC, the MIPS R8000 and R10000, the Intel i860 and i960, the Sun Microsystems SPARC, etc., derive their benefit from instruction-level parallelism.

Instruction-level parallel processors take advantage of the parallelism in programs by performing multiple machine level operations simultaneously. A typical such processor (a superscalar or VLIW processor), provides multiple pipelined functional units in parallel, thus allowing the simultaneous issue of multiple operations per clock cycle [12]. In order to take advantage of instruction-level parallelism, compilation techniques are needed that expose parallelism in programs written in a high-level language.

1.1 Exact Modulo Scheduling Approach

The subject of this paper, software pipelining, and more precisely modulo scheduling, is a compiler parallelization technique which has recently received a lot of

* important part of this work was done while in McGill University in Montreal
attention and is successfully implemented in production compilers [4, 14]. Because of its computational complexity, heuristic algorithms are used for modulo scheduling. One question that heuristic approaches leave unanswered is "How well do these methods do their job?". Indeed, is there any room for improvement? These types of questions led to the development of exact scheduling methods that guarantee the optimality of their results. The main idea behind the exact methods is to represent the scheduling problem as an optimization problem with a set of linear scheduling constraints, and an objective function minimizing some cost criterion. A number of interesting results of using the linear and integer linear programming approach for software pipelining have been published recently [10, 7, 2, 1, 5].

This paper concentrates on the development of the integer linear programming software pipeliner for the MIPS R8000 microprocessor based on the previous work by E. Altman [1]. In developing such software pipeliner our main interests were to study how well the ILP approach would work when targeted to a real processor. After the publication of the first measurement of runtime performance for ILP-based generation for software pipelines [14], one of the important questions left unanswered was how much uncertainty have been introduced by the memory system in our experiments? In this paper we analyze this question and present a new set of experimental results with the memory effects taken care of by an improved ILP model.

1.2 MIPS R8000 Main Features

MIPS R8000 is a pipelined superscalar processor which allows multiple issue of up to four instructions chosen from two integer, two memory, and four floating-point instruction types per clock. The R8000 contains two arithmetic logic units (ALU), one shifter, one multiply-divide unit and two address generation units. Two independent memory instructions are supported per cycle.

Floating-point coprocessor (FPU) performs all the floating point functions. It contains two execution datapaths each capable of multiply-adds, simple multiplies, adds, divides, square roots and conversions. The two datapaths are completely symmetric and indistinguishable for the software - the compiler simply knows that it can schedule two floating-point operations per cycle. Floating-point loads and stores go directly to/off chip external cache. External cache latency is hidden by decoupling the coprocessor from the R8000 pipeline. Floating-point instructions are dispatched into a queue where they can wait for resource contention and data dependencies to clear without holding up the integer dispatching. If a floating-point load instruction is immediately followed by a compute instruction that uses the result of the load, the queue allows both instructions to be dispatched together as if the load had no latency at all. The integer pipeline is immediately free to continue on other instructions. The load instruction proceeds down the external cache pipeline, in the meantime the compute instruction waits in the floating point instruction queue until the load data is available. By decoupling floating-point operations, a limited form of out-of-order execution of floating-point instructions is achieved.