Enhancing Instruction Scheduling with a Block-Structured ISA

Stephen Melvin¹ and Yale Patt²

Received July 8, 1993

It is now generally recognized that not enough parallelism exists within the small basic blocks of most general purpose programs to satisfy high performance processors. Thus, a wide variety of techniques have been developed to exploit instruction level parallelism across basic block boundaries. In this paper we discuss some previous techniques along with their hardware and software requirements. Then we propose a new paradigm for an instruction set architecture (ISA): block-structuring. This new paradigm is presented, its hardware and software requirements are discussed and the results from a simulation study are presented. We show that a block-structured ISA utilizes both dynamic and compile-time mechanisms for exploiting instruction level parallelism and has significant performance advantages over a conventional ISA.

KEY WORDS: Instruction scheduling; instruction level parallelism; superscalar; VLIW; instruction set architecture.

1. INTRODUCTION

A basic block is defined by Aho et al.,¹ as a sequence of consecutive statements in which the flow of control enters at the beginning and leaves at the end. When basic blocks are small, performance suffers for several reasons. First, the instruction supply hardware is taxed. Changes in the flow of control expose the latency in prefetching and decoding from a new target address. In addition, small basic blocks limit multiple instruction per cycle execution. Issuing a large number of instructions in one cycle becomes impractical or at least very hardware intensive. Small basic blocks

¹ P.O. Box 2400, Berkeley, California 94702-0400. (E-mail: melvin@zytek.com).
² Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109-2122. (E-mail: patt@eecs.umich.edu).
also complicate the task of the compiler in finding operations to overlap. As machines continue in the current trend of wider issue sizes, these are guaranteed to become more significant problems.

Exploiting instruction level parallelism across multiple basic blocks is straightforward in some instances. Many scientific programs have basic blocks that are fairly easy to enlarge. For example, when the bounds of a loop are known statically and there are no conditional tests inside, the loop can be trivially unrolled to provide a larger basic block. General purpose programs, however, generally fall into a different category. They often have small basic blocks and conditional branches that are hard to predict statically.

There has been a wide variety of techniques developed to exploit parallelism across multiple basic blocks. Early techniques involved global instruction scheduling by the compiler to move code between basic blocks. Currently there is a trend to supplement these techniques with architectural constructs (such as conditional instructions) and hardware mechanisms (such as speculative execution). There is also a trend toward the use of dynamic scheduling to further enhance the exploitation of instruction level parallelism. Dynamic scheduling is a microarchitectural mechanism that separates instruction issue from instruction execution. This mechanism has been implemented and proposed in many variations. The tag forwarding scheme of the IBM 360/91 originated the core idea behind dynamic scheduling. HPS generalized the concept of tag forwarding to encompass all operations within a processor, including memory operations, and with enough backup state to allow dynamic branch prediction and precise exceptions. Dynamic scheduling has particular advantages under variability in memory latency (e.g., cache hits vs. misses) and for dynamic memory disambiguation (e.g., when the compiler can't guarantee the independence of two memory references).

In this paper the idea of a block-structured instruction set architecture (ISA) will be introduced, which represents a logical extension of static and dynamic scheduling concepts. A block-structured ISA treats an entire group of instructions as an atomic unit, much as conventional ISAs treat individual instructions. This concept has several important implications that will be discussed. We will show that a block-structured ISA allows the compiler more flexibility in instruction scheduling, so that it can be more effective in uncovering global parallelism. Furthermore, the hardware can execute instructions in parallel more efficiently, and the instruction supply bottleneck is widened.

This paper is divided into six sections. In Section 2, we define some terms and provide some general background. Section 3 presents a survey of previous techniques for exploiting instruction level parallelism across multiple basic blocks. The hardware and software requirements of each are