Design and Validation of a Simultaneous Multi-Threaded DLX Processor

Hans Jacobson

Abstract— **Modern day computer systems rely on two forms of parallelism to achieve high performance, parallelism between individual instructions of a program (ILP) and parallelism between individual threads (TLP). Superscalar processors exploit ILP by issuing several instructions per clock, and multiprocessors (MP) exploit TLP by running different threads in parallel on different processors.**

A fundamental limitation of these approaches to exploit parallelism is that processor resources are statically partitioned. If TLP is low, processors in a MP system will be idle, and if ILP is low, issue slots in a superscalar processor will be wasted. As a consequence, the hardware cannot adapt to changing levels of ILP and TLP and resource utilization tend to be low.

Since resource utilization is low there is potential to achieve higher performance if somehow useful instructions could be found to fill up the wasted issue slots. This paper explores a method called simultaneous multithreading (SMT) that addresses the utilization problem by letting multiple threads compete for the resources of a single processor each clock cycle thus increasing the potential ILP available.

I. Introduction

To achieve high performance, modern day computer systems rely on two forms of parallelism in program execution. Wide issue superscalar processors try to exploit instruction level parallelism (ILP) that exists within a single program and issue multiple instructions per cycle. However, even aggressive superscalar implementations that use dynamic hardware scheduling to extract parallelism cannot take full advantage of the resources of a wide issue processor due to inherent control and data dependencies between instructions of a single program. Since the resources in the superscalar case are statically allocated to a single program, resources (issue slots) are wasted when there is not sufficient ILP available in that program. Figure 1(a) illustrates the *vertical* and *horizontal* issue slot waste that can take place in a superscalar processor. Horizontal waste occurs when the scheduling logic cannot find enough instructions to issue to fill up all issue slots this cycle, i.e. there is a lack of ILP available. Vertical waste may occur when a cache miss or data dependencies hinders the scheduling logic to issue any instruction this cycle.

Multiprocessors (MP) try to exploit thread level parallelism (TLP) that exists either between parallel threads derived from a single program, or between completely independently executing programs. The individual processors in the MP system can suffer from vertical and horizontal issue waste as in the superscalar case. In addition, an MP system can suffer from *thread shortage* which leaves some processors without a program to execute. Resources in these idle processors are thus wasted due to lack of TLP as shown in Figure 1(b). A typical example of thread shortage is when a program that has been parallelized into multiple threads has to go through a sequential section of code.

Multithreaded (MT) processors [1] allow several thread contexts to be active. Each cycle, one context is selected and instructions from that thread are issued. MT processors can thus address the problem of vertical issue slot waste. Whenever a certain thread cannot issue any instructions this cycle, another thread that can issue is selected as illustrated in Figure 1(c). While MT addresses the vertical waste problem, the limitation that only one thread can issue per cycle still leaves the problem of horizontal waste.

Simultaneously multithreaded (SMT) processors also allow several thread contexts to be active. Each cycle, instructions can be issued from multiple threads. SMT processors thus address both vertical and horizontal waste. Whenever a thread cannot issue any instructions during a cycle, all other threads can still issue so vertical waste is reduced. Whenever a thread cannot fill all issue slots during a cycle, instructions from other threads can compete for and fill up these slots thus reducing horizontal waste. These situations are illustrated in Figure 1(d).

Statically partitioning processor resources puts a limitation to how much parallelism can be exploited. The superscalar and MP processors statically partition the individual processor resources to be used by only allowing one thread to execute at a time. MT processors improve upon this concept by allowing multiple threads to be on standby but still only allow one thread to use the processor resources per cycle. An SMT processor on the other hand has the ability to dynamically adapt to varying levels of TLP and ILP since each cycle multiple threads compete for available issue slots. By allowing multiple threads to issue instructions each cycle, TLP is effectively transformed into ILP since there is no control or data dependency between instructions belonging to different threads. Subsequently, given the same amount of resources, SMT has the potential to do more useful work compared to the other approaches. This has also been indicated by a comparative study of SMT and MP architectures [2].

Project goals

The focus of this project has been the development, implementation, validation, and evaluation of a simultaneous multithreaded microprocessor architecture running DLX native code. In this paper we will focus on the architecture implementation and performance analyses of the processor. We are mainly interested in finding out how simultaneous multithreading can help improve instruction throughput on

The author is with the Department of Computer Science, University of Utah, Salt Lake City, U.S.A. E-mail: hans@cs.utah.edu