Simultaneous Multithreading VLIW Processor Architecture

Ferreira, Victor M. Goulart
Department of Computer Science and Communication Engineering, Faculty of Information Science and Electrical Engineering, Kyushu University

Yasuura, Hiroto
Department of Computer Science and Communication Engineering, Faculty of Information Science and Electrical Engineering, Kyushu University

出版情報：Seventh International Symposium on High-Performance Computer Architecture, 2001-01
バージョン：
権利関係：

Simultaneous Multithreading VLIW Processor Architecture

Victor M. Goulart Ferreira and Hiroto Yasuura
Dept. of Computer Science and Communication Engineering
Kyushu University
{goulart, yasuura}@c.csce.kyushu-u.ac.jp

Abstract

This paper introduces the concept of a novel architecture, ***SMTVLIW: Simultaneous Multithreading VLIW Machine****, which incorporates real-time task scheduling at the microarchitecture level of VLIW-like processors. This architecture permits to simultaneously run an arbitrary number of threads, building very long instruction words on-the-fly, orchestrated by an RTOS for dynamic load balancing and fine-grain power control of the system. The SSMTVLIW (Static SMTVLIW) and DSMTVLIW (Dynamic SMTVLIW) machines are also introduced.*

Keywords: *SMT, VLIW Architectures, RTOS, Thread-based Architectures, Dynamic Compilation.*

1. Introduction

In the near future it would be possible to integrate thousands of functional units on a single chip, making Very Large Data Path (VLDP) processors with high issue widths feasible. Compared to superscalar, VLIW processors have simpler control logic and wider issue widths, being faster and more power efficient.

Multiple instruction issue has the potential to increase performance, but is limited by instruction dependencies and long-latency operations. The effects of these are known as *horizontal waste* and *vertical waste*[1]. Traditional multithreading, found in machines like HEP, Tera and MIT's Alewife, hides memory and functional units latencies, attacking vertical waste. Simultaneous multithreading, in contrast, attacks both horizontal and vertical waste[6]. Performance and power consumption must be taken into account specially in dynamic systems, where the set of tasks as well as their associated priorities change over time.

This paper presents the concept of a new architecture, called *SMTVLIW—Simultaneous Multithreading VLIW Machine*. Our proposed scheme improves the scheduling of tasks in Real-time Systems for VLIW machines, taking into account both demands (high performance and low power). Section 2 presents the ideal SMTVLIW, followed by its subdivisions: SSMTVLIW and DSMTVLIW. Section 3 describes the fundamental elements in the architecture, followed by a taxonomic classification in Section 4. Section 5 concludes the paper.

[1] Vertical waste is introduced when the processor issues no instruction in a cycle, and horizontal waste when not all issue slots can be filled in a cycle.

2. The Ideal SMTVLIW

As other SMT machines, the SMTVLIW architecture explores intra- and inter-thread parallelism to maximize processor utilization. However, instead of a superscalar execution paradigm, we explore a VLIW one.

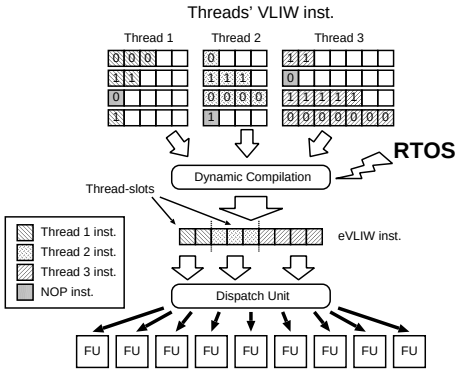

Figure 1. The computation model of the SMTVLIW.

An RTOS, guided by the power budget and task/threads deadlines, directly influence the way instructions are executed in the microarchitecture. This is done by adjusting the computation power, i.e. the number of FU's in fact used to perform the operations. Figure 1 presents the computation model of the SMTVLIW machine.

According to the way instructions are assembled, one can distinguish SMTVLIW machines into two types: SSMTVLIW (Static SMTVLIW) and DSMTVLIW (Dynamic SMTVLIW), as shown in Figure 2.

2.1 SSMTVLIW – Static SMTVLIW

In the Static SMTVLIW, the VLIW instructions with code from multiple threads, are preassembled at compilation time, giving more opportunities for code compaction

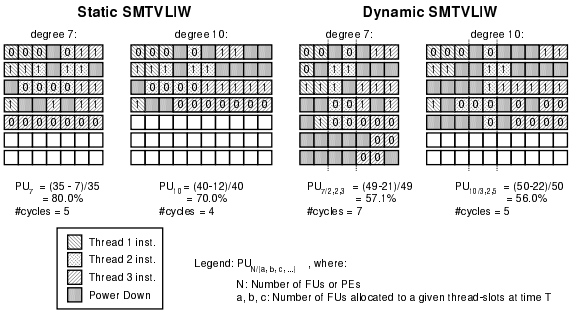

Figure 2. According to the influence of the RTOS, different eVLIW inst. may be build for the SSMTVLIW and DSMTVLIW. Thread-slots FU's allotments in the DSMTVLIW may vary during program execution, environment changes or stricter deadlines/power budgets.

and better resource scheduling. In order to reduce interconnection problems with the FU's, *thread slots* may be set where each thread's instructions are arranged.

The dynamic compilation part of the architecture just has to check FU's latencies and communication/interruption matters, stopping the dispatch of part(s) of the VLIW.

2.2 DSMTVLIW – Dynamic SMTVLIW

In the Dynamic SMTVLIW, the set of threads as well as the allocated computational power for a given task is changed on-the-fly by an RTOS, according to the threads' deadlines and priorities. The RTOS may decide to utilize less parallelism than actually available in a thread in order to reduce power consumption. This time, the computational power or throughput of each application may be decided by the user or set automatically depending on the environment.

3. Fundamental Components

This section describes the fundamental elements to support a SMTVLIW processor: the compilation process and the RTOS.

3.1 Dynamic Compilation

In traditional VLIW systems, the compiler must identify those operations that can be simultaneously issued building VLIW's for the processor's datapath. Thus, code generated to a VLIW processor with 5 FU's will not run in a processor with 4 FU's. This is known as the object-code compatibility problem and has limited broad utilization of VLIW systems. However, a number of works have presented efficient ways to eliminate this bottleneck[1, 2, 3]. These techniques, generally known as dynamic compilation, focused on solving different latencies of FU's and making code compatible between processors in a given family of VLIW systems.

The compiler for the SMTVLIW machine generates code similarly as an EPIC compiler does, and tries to identify the maximum ILP in a given cycle, independently of the real characteristics of the processor's datapath. Then, dynamic compilation is used to assemble the VLIW instructions, called *eVLIW—Effective VLIW*, that will be actually passed to the dispatch unit.

3.2 Real-Time Operating System

A key component for fine-grain adjustments of performance and power consumption of the SMTVLIW is its RTOS. The RTOS dictates the way eVLIW instructions will be assembled. It independently determines the computing power to be allocated to a given task or thread, also controlling the power supply cut off of those parts of the processor not used; this happens when the set of threads does not use certain parts of the datapath and/or when power consumption becomes the major concern.

4. SMTVLIW Classification

Here we identify the uniqueness of SMTVLIW, comparing it to other systems. The classification (Table 1) is based on the presence or not of simultaneous multithreading; the way programs are compiled (the presence or not of dynamic compilation); and if the RTOS directly influences the execution at the microarchitecture level[2].

	Single Thread VLIW	**Multithread VLIW**
Static Compilation	Well-known (RTness and RTless)	VIM[4] (RTless)
Dynamic Compilation	DIF[3] (RTless) DAISY[1] (RTless) Crusoe[5] (RTless)	SSMTVLIW (RTless) DSMTVLIW (RTness)

Table 1. Taxonomic Classification of the SMTVLIW.

5. Summary and Conclusions

This paper has presented the SMTVLIW machine which incorporates real-time scheduling at the microarchitecture level of VLIW-like processors. According to its RTOS awareness SMTVLIW machines can be classified into Static SMTVLIW or Dynamic SMTVLIW.

References

[1] K. Ebcioğlu and E. R. Altman. DAISY: Dynamic compilation for 100% architectural compatibility. In *Proc. of The 24th ISCA*, pages 26–37, June 1997.

[2] M. Franklin and M. Smotherman. A fill-unit approach to multiple instruction issue. In *Proc. of The 27th MICRO*, pages 162–171, 1994.

[3] R. Nair and M. E. Hopkins. Exploiting instruction level parallelism in processors by caching scheduled groups. In *Proc. of The 24th ISCA*, pages 13–25, June 1997.

[4] H. Shimajiri and T. Yoshida. A VLIW processor using both thread-level and instruction-level parallelism. Technical report, IEICE (CPSY2000-7), May 2000. (In Japanese).

[5] Transmeta Inc. The crusoe processor. http://www.tensilica.com.

[6] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proc. of The 22nd ISCA*, pages 392–403, June 1995.

[2] RT control in Crusoe does not influence the way instructions are executed inside the VLIW core engine, but the clock frequency of the system.