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

http://hdl.handle.net/2324/7648

出版情報: Seventh International Symposium on High-Performance Computer Architecture, 2001-01
バージョン: accepted
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\(^1\). 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.

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.

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.

\(^1\)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.
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.

<table>
<thead>
<tr>
<th>Static SMTVLIW</th>
<th>Dynamic SMTVLIW</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td><strong>CPU</strong></td>
</tr>
<tr>
<td><strong>FUs</strong></td>
<td><strong>FUs</strong></td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td><strong>Power</strong></td>
</tr>
<tr>
<td><strong>Thread Slots</strong></td>
<td><strong>Thread Slots</strong></td>
</tr>
<tr>
<td><strong>Instruction Set</strong></td>
<td><strong>Instruction Set</strong></td>
</tr>
<tr>
<td><strong>SMTVLIW</strong></td>
<td><strong>SMTVLIW</strong></td>
</tr>
<tr>
<td><strong>DSMTVLIW</strong></td>
<td><strong>DSMTVLIW</strong></td>
</tr>
<tr>
<td><strong>VIM</strong></td>
<td><strong>VIM</strong></td>
</tr>
<tr>
<td><strong>DSMTVLIW</strong></td>
<td><strong>DSMTVLIW</strong></td>
</tr>
<tr>
<td><strong>RTOS</strong></td>
<td><strong>RTOS</strong></td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td><strong>Compiler</strong></td>
</tr>
</tbody>
</table>

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


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