Interleaved Multithreading on a MIPS Processor

Felipe Madruga, Philippe Olivier Alexandre Navaux
Universidade Federal do Rio Grande do Sul
Instituto de Informatica
Caixa Postal 15.064 91.501-970 Porto Alegre RS Brazil
flmadruga,navaux@inf.ufrgs.br

Abstract

The work exposed here intend to discuss the implementation of Interleaved Multithreading on the miniMIPS pipeline. We interleave instructions from two threads in a way that there is no need to predict branches neither flush the pipeline. The IMT technique was applied on a MIPS-I soft core called miniMIPS. Along the paper, we show practical issues and talk about performance and occupation in a FPGA synthesis of the modified core.

1. Introduction

The increased frequency of processors together with the extraction of Instruction Level Parallelism (ILP) have brought problems. Among these problems, we can cite the severe penalty for branch missprediction and complexity of the pipelines. Moreover, the speed in accessing memory have not followed the advance in speed of execution of instructions, causing the penalty for a cache miss to be of several cycles. These factors make the processor stay stalled in up to 75% of cycles [3]. To keep the processor busy with events of great latency, as cited above, were created techniques called hardware multithreading[8].

To illustrate the theme and develop properly, this document deals with the modification of the internal organization of MiniMIPS processor so that it has the functionality of explicit Multithreading interleaving the execution of two threads, to remove the parts of the organization responsible for treatment of dependencies.

This interleaving means that the treatment of data and control dependencies can be eliminated, since the execution of a thread does not execute in contiguous stages within the pipe. Here, we talk about the elimination of the treatment of dependencies and insertion of another thread in the pipeline with IMT, considering performance and area.

Experiments were made by modificating the hardware description of the processor and using specialized tools. Throughout the text this work is examinated. It is important to emphasize that this approach inted to increase the throughput, not the execution time of a single task. Our objective is to analyze the impact of the IMT technique in the core.

The organization of this document is as described below. In section 2 some arguments about the context of this work are given, in section 3, we discuss the proposed changes in the organization of miniMIPS processor and, finally, are made closing comments in the last section.

2. Motivation

2.1. Interleaved Multithreading

Basically, the hardware multithreading is based on maintaining more than one context in the processors registers, so that the instructions of a context execute when there is hardware available to do so. Interleaved Multithreading (IMT), Blocked Multithreading (BMT) and Simultaneous Multithreading (SMT) are categories that the majority of the implementations fits, as can be seen in the classification of Figure 1[8]. In BMT, the switching between contexts is caused by high-latency events or explicitly. The SMT is applied using different execution units for different threads, and its use is more appropriate in superescalar processors. In IMT [2], one instruction of each thread is fetched from memory at a cycle, which means that the instructions are interleaved along the pipeline.

Some studies [5][7] show that the IMT technique can bring benefits to the design of processors. The IMT, also called multi-fine grain, is one in which, in each cycle instructions of a thread are fetchched.

In Figure 2[8], are shown the exchange of context in different models of hardware in multithreading processor with one execution unit. In (a), is shown a processor with a thread, performing a high-latency event and inserting bubbles in the pipeline. In (b) is shown the IMT, and (c), is illustrated the BMT.

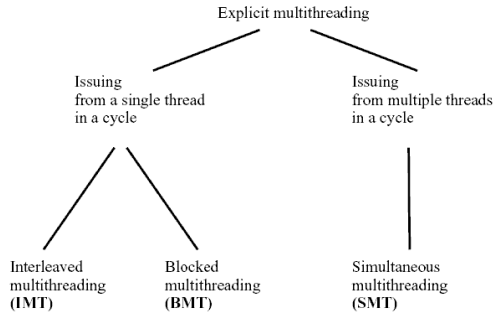

Figure 1. Hardware Multithreading Taxonomy.

Given that the IMT eliminates dependencies of control and data in the pipeline, the pipeline hazards should not appear and the processor can be more easily constructed, without the need for complex ways of forwarding or branch predictions.

The technique has been implemented in the IMT Denelcor Heterogeneous Processor Element (HEP) [6], and the Niagara processor [3]. However, the best example of implementation is the network processor LEXRA LX4580 [1].

The HEP processor is designed to have up to sixteen processors with a maximum 128 threads per processor. There were a large number of registers dynamically allocated and managed so that the threads share them.

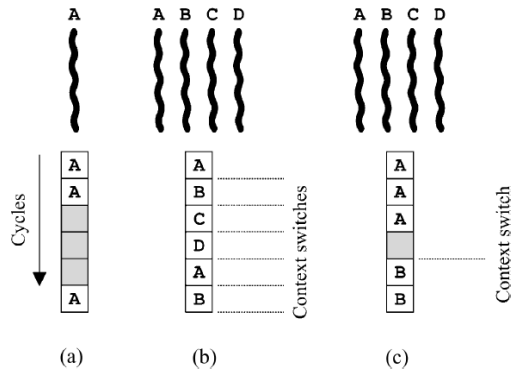

Figure 2. Multithreading on Singlescalar Processors.

The network processor LX4580 implements the instruction set of the MIPS32 to complete the processing in software for IP packets. It has seven stages of pipeline and four threads, whose instructions are fetched in a circular order. In this processor, the IMT benefits a better use of the pipeline, removing the bubbles that are normally included in branch misspredictions, with conditional branches and stalls because of memory access.

2.2. The miniMIPS Processor

The MiniMIPS [4] is a processor described in VHDL that implements fifty-one instructions of the MIPS-I standard, and its code is freely distributed under Gnu Public License. Hence, it has a well known organization and it is reasonably well documented.

The internal structure of the miniMIPS processor is shown in Figure 3. The top entity of the processor is composed of the following entities.

- **renvoi:** takes care of data dependency, decides some control signals and does operand bypassing (BYP);
- **syscop:** operates on the co-processor calls, interruptions and exceptions;
- **predict:** it is responsible for the branch prediction, monitorates some control signals of each pipeline stage and flushes the pipe on a misspredic-tion (BRP);
- **banc:** Register Bank (BCR);
- **pf:** PC Calculation (PF);
- **ei:** Instruction Fetch Stage (EI);
- **di:** Decoding Stage (DI);
- **ex:** Instruction Execution (EX);
- **mem:** Pipeline stage where some instruction access the memory (MEM);
- **bus_ctrl:** handles the interface with the memory.

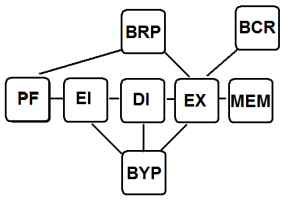

Figure 3. miniMIPS Organization.

The miniMIPS pipeline is composed of the following five stages.

- Address Calculation (PF);
- Instruction Extraction (EI);
- Instruction Decoding (DI);
- Execution (EX);
- Memory Access (MEM).

3. Changes Made

The solution proposed here and analyzed using IMT aimed to eliminate the bubbles created by data and control dependencies in the pipeline, not by cycles lost due to access to memory. Doing this, one can simplify the pipeline and keep it busy and without the need to clean it in conflict situations.

In its original version, the miniMIPS pipeline works as table 1. There a instruction flow without branches, called A, is exposed. Where there is a conditional branch, it will only be resolved in the fourth stage, which leaves the three following instructions depending on the result of the branch. To address this problem, you can: rearrange the instructions (which would be hard, because there are too many branches in a typical program); insert bubbles in the pipeline or use a predictor of branches.

Ciclos	PF	EI	DI	EX	MEM
1	A1				
2	A2	A1			
3	A3	A2	A1		
4	A4	A3	A2	A1	
5	A5	A4	A3	A2	A1

Table 1. miniMIPS Pipeline

Designers of miniMIPS solved the problem using a predictor based on a branch history table. A misspredict causes the loss of three instructions that are already in the pipeline. Table 2 shows the example of a branch that has been taken and, in the execution stage (EX), it is discovered that it should not have been taken.

Ciclos	PF	EI	DI	EX	MEM
1	A1				
2	A58	A1			
3	A59	A58	A1		
4	A60	A59	A58	A1	
5	A2	CLEAR	CLEAR	CLEAR	A1

Table 2. Misspredict on The miniMIPS

The implementation of IMT in the pipeline makes control dependence be of only one instruction (as explained in Table 3, with two threads: A and B). In the same way as in the processor LX4580, we withdraw the branch predictor from the project, considering a branch delay slot. To illustrate this, table 3 is showed. Assuming that the instruction A1 is a conditional branch, the instruction A2 is fetched before A1 reach the forth stage (this explains the branch delay slot), however A3 is fetched after A1 pass through the forth stage. Given this, A3 will always be the right instruction to fetch. Thus, we can eliminate the branch predictor.

Ciclos	PF	EI	DI	EX	MEM
1	A1				
2	B1	A1			
3	A2	B1	A1		
4	B2	A2	B1	A1	
5	A3	B2	A2	B1	A1

Table 3. miniMIPS Pipeline with IMT

To make the execution of two independent threads possible, it is necessary to provide basically more one program counter (PC) and one register bank. The chosen way to do so was to double the entities pf and Banc cited in the previous section and add a selection logic in their buses.

The solution used to synchronize the stages so that do not cause inconsistencies was make a bit counter that selects the duplicated entities. Were associated multiplexers and demultiplexers to the PC and register bank, these being selected by the earlier cited bit counter.

With the changes described above, the organization of miniMIPS (shown in Figure 3) becames as the way illustrated in Figure 4.

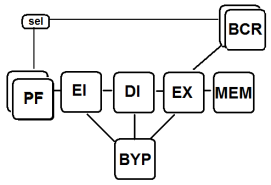

Figure 4. miniMIPS with IMT.

The following considerations are made about the impact of changes made. It is also made a performance analysis of applications in this new structure.

3.1. Area Occupation Impact

The increase in area tends to be considerable, because it was doubled a component that occupies much of the processor, the register bank. The synthesis of the bank registes resulted in 992 Flip-Flops, about half of Flip-Flops of the implementation of the original miniMIPS. Nevertheless, the increase was mad mainly by the register set, what indicates that in larger processors will make a smaller increase, proportionately.

Table 4, consider the area occupied by the original processor and with IMT. Also, in the last column, it is showed the percentage increase in the corresponding matter. The data generated were obtained using the Xilinx ISE 10.1 tool. It was considered the implementation in a FPGA XC2V100 from the Virtex2 family.

	Without IMT	With IMT	Increase
Slices	2578	2997	15.5%
Flip-Flops	1919	2712	41.3%
4-LUTs	4865	5322	9.4%
Max-Freq	64.309MHz	62.418MHz	-3.0%

Table 4. Comparison on a FPGA Synthesis

3.2. Performance Impact

The execution time of a unique thread is not the target of the inclusion of IMT in the pipeline miniMIPS. The idea is to increase the troughtput (number of tasks completed over time).

In the original miniMIPS, the execution time is linked to the number of cycles per instruction (CPI). The CPI, in turn, due to the branch predictor, depends on the accuracy of forecasts and the number of branches of the program. The CPI is related to the jumps and predictor as follows

$$CPI = (pp * pb) + (1 - pb) + pb * (1 - pp) * 4$$

where, pb is the percentage of jumps in the program and pp is the percentage of correct predictor of violations. For example, if a program has 20% of branches and hits 70% of predictions we have CPI = 1.18. But in the case with IMT, the CPI is equal to 1 for the two threads. But a sequential program is equal to the CPI 2. Let's look at three examples to verify the performance of the processor miniMIPS with IMT.

The first example is two completely independent applications that need to be executed. Each of them will take twice the time to be executed, however, when one end the other ends as well. It occurs in the same way as if the two were executed one by one in a processor with CPI equal to one, not counting the cost of exchanging context that with IMT is nil.

In the second example, let's consider an application that can be tottaly parallelized, as a matrix multiplication. The task would be fully completed with CPI equal to one.

Consider now two independent applications in the presence of an operating system that should treat them so that the two perform at the same time in a processor without IMT. There will be all the overhead of an context switch, clock interruption treatment (in case of an operating system with preemption), such as the shorter the implementation of each task, more switching, therefore more overhead. With IMT that overhead can be considerably reduced, due to the context switch with no cost.

4. Conclusions

Analyzing the results, we see that the increase in area tends to be acceptable, however, it is possible that an implementation with two processors whole miniMIPS can take better advantage of applications that are not parallel or require a better performance in terms of execution time, but this obviously occupy more area. Besides, it is important to say that the critical path was not affected by the modification.

The elimination of a branch predictor is a desirable thing because the execution time of a program would not depend this factor. In [1], it is shown that branch predictors can have very poor performance when operating in certain workloads.

As future work, would be possible to analyse the feasability on a situation with a memory hierarchy and occupation of cycles in which the processor waiting for data from a high-latency access memory. Another possible improvement is considering sharing some of the registers, in an attempt to reduce the area occupied. Moreover, tests on the power consumption and a more comprehensive validation is shown necessary.

References

[1] H. P. K. Gelinas, B. Finegrained hardware multi-threading: A cpu architecture for high-touch packed processing. lexra inc., waltham, ma. white paper. 2002.

[2] M. H. J. Laudon, A. Gupta. Interleaving: A multithreading technique targeting multiprocessors and workstations. *Proc. of the Sixth ASPLOS-VI. 1994*.

[3] A. K. O. K. Kongetira, P. Niagara: 32-way multithreaded sparc processor. *IEEE Comput. Soc. 2005*.

[4] L. M. O. S. S. Hangout, S. Jan. The minimips project. *available online at http://www.opencores.org/projects.cgi/web/minimips/overview 2005*.

[5] J. L. S. Kapil, H. McGhan. A chip multithreaded processor for network-facing workloads. *IEEE Micro. 2004*.

[6] B. J. Smith. Architecture and applications of the hep multiprocessor computer system. *SPIE. 1981*.

[7] B. H. H. J. Stephan Suijkerbuijk. Implementing hardware multithreading in a vliw architecture. *International Conference on Parallel and Distributed Computing Systems. 2005*.

[8] J. S. Theo Ungerer, Borut Robic. A survey of processors with explicit multithreading. *ACM Computing Surveys. 2003*.