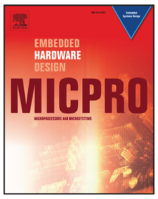
Microprocessors and Microsystems 000 (2016) 1–13

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro



Efficient resource sharing algorithm for physical register file in simultaneous multi-threading processors

Yilin Zhang[a,*], Wei-Ming Lin[b]

[a] Advanced Micro Devices, Inc., Austin, TX, 78735, United States
[b] Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249-0669, United States

A R T I C L E I N F O

Article history:
Received 23 September 2015
Revised 10 February 2016
Accepted 2 June 2016
Available online xxx

Keywords:
Simultaneous multi-threading
Superscalar
Physical register file
Register renaming
IPC

A B S T R A C T

Simultaneous Multi-Threading (SMT) processors increase performance by allowing concurrent execution of multiple independent threads with sharing of key datapath components and better utilization of the resources. An SMT processor usually maintains a shared register file to accommodate multiple threads for register renaming. By supporting inter-thread sharing of the physical registers, an SMT system processor can reduce the number of registers that would have been required in multiple superscalar processors while achieving a comparable throughput. However, congested shared resources due to slower threads can easily lead to inefficient usage of the resources and thus an undesirable performance outcome. In this paper, we propose an allocation algorithm at the architectural level for a better utilization of the shared register file. We show that, by limiting the number of the physical register entries each thread is allowed to occupy at any given time, the overall system throughput is enhanced by a substantial margin. An improvement in IPC of up to 44.6% and 32.7% is observed when the proposed technique is applied to a 4-threaded and a 6-threaded SMT system, respectively. Furthermore, a 4-threaded system with a physical register file of 160 entries can deliver a performance comparable to that of a default system with 256 entries, reflecting a resource saving of 37.5%.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Noting the resource utilization deficiencies in the traditional superscalar processors, Simultaneous Multi-Threading (SMT) offers an improved mechanism to enhance overall system performance by allowing concurrent execution of instructions from different threads. The most common characteristic of SMT processors is the sharing of key datapath components among multiple independent threads. Essentially SMT improves the overall performance by exploiting Thread-Level Parallelism (TLP) among threads to overcome the limits of Instruction-Level Parallelism (ILP) present in a single thread [1,2]. Subsequently, due to the sharing of resources, the amount of hardware required in an SMT system can be significantly less than that from employing multiple copies of superscalar processors while achieving a similar throughput.

There have been numerous research efforts targeted in improving the efficiency of the resource allocation and utilization in SMT systems. Some of them improve the fairness of the resource allocation among threads by modifying the fetch policy. For example, ICOUNT [3] assigns a higher fetching priority to a thread with fewer instructions in pre-issue stages; STALL and FLUSH [4] adopts a fetch policy to address issues from L2 cache misses; a dynamical fetch policy DCRA presented in [5] is a technique based on memory performance of each thread to exploit parallelism beyond stalled memory operations; PEEP [6] controls the fetch unit by exploiting the predictability of memory dependencies; SAFE-T in [7] and a speculation control technique in [8] both give higher fetching priorities to threads with higher branch prediction accuracy.

Note that, in an SMT system, the resources shared among threads normally include physical register file, various machine bandwidths (e.g., inter-stage bandwidth, read/write ports for register file and memory, etc.), inter-stage buffers (e.g., Issue Queue (IQ)), functional units, write buffer, etc. In some processors, Instruction Fetch Queue (IFQ) and *Re*-Order Buffer (ROB) are also shared among threads. Previous research works have examined the allocation of the shared buffers in the pipeline in order to achieve a more efficient utilization of the shared resources. For example, APRA dynamically assigns resources (IFQ, IQ and ROB) to threads according to changes of threads' behavior [9]. Hill-Climbing [10] is a learning-based algorithm that uses performance feedback to partition the shared hardware resources in the pipeline including IFQ, physical registers, ROB and IQ. In [11], a write buffer occupancy

* Corresponding author.
E-mail addresses: zhangyilin.bupt@gmail.com, yilin.zhang@amd.com (Y. Zhang), weiming.lin@utsa.edu (W.-M. Lin).

http://dx.doi.org/10.1016/j.micpro.2016.06.002
0141-9331/© 2016 Elsevier B.V. All rights reserved.

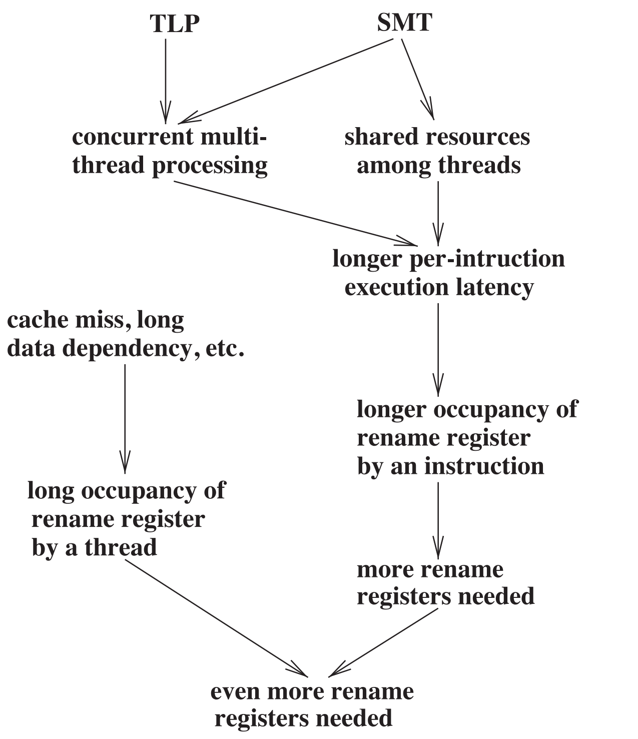

TLP
SMT
concurrent multi-thread processing
shared resources among threads
longer per-intruction execution latency
cache miss, long data dependency, etc.
longer occupancy of rename register by an instruction
long occupancy of rename register by a thread
more rename registers needed
even more rename registers needed

Fig. 1. Renaming register requirement in an SMT versus a single-threaded system.

capping technique is proposed for an efficient utilization on the shared write buffer. Some other research results have enhanced the overall performance by improving the utilization of IQ [12–15].

In this paper, we focus on the utilization of the physical register file which may significantly impact SMT's throughput and cost-effectiveness. A physical register file is used by the CPU for real-time renaming to resolve register name dependencies. A rename register is allocated to a destination register at the rename stage of an instruction and will not be released (deallocated) until the next instruction with the same destination register is safely committed. For a single-threaded superscalar CPU, the size of rename register file required depends on the prevalence of name dependencies within a time frame of program execution and the degree of ILP present to support concurrent processing. Usually neither is high enough to call for a very large size of register file to prevent the rename stage from becoming the bottleneck. That is, a stalled instruction (due to a cache miss or other reasons) occupying a renaming register will most likely stop most of subsequent instructions from proceeding even they are not delayed at the rename stage due the lack of renaming registers.

On the other hand, in a typical SMT system to take advantage of the TLP newly present, a much larger physical register file is required in order to accommodate register renaming for multiple threads to prevent a bottleneck at the rename stage. Fig. 1 briefly illustrates the reasoning for this requirement compared to a single-threaded system. Multiple threads running in an SMT system sharing several key resource components would no doubt lead to a longer expected latency per instruction, which in turn prolong occupancy duration of a rename register by an instruction. This reason itself would demand more rename registers per thread than the single-threaded system. Another factor comes from the real-time changing behavior of competing threads and the potentially long occupancy duration of a register from the time it is allocated to when it is released. That is, a renaming register occupied by a stalled instruction of a thread means the loss of a potential useful register that could have been used by another thread. This leads to an even higher demand for renaming registers. Employing a large register file may not only be cost inhibitive but could also lead to longer access latency and excessive area/power consumption. In addition, such a choice is exactly contradictory to the underlying design philosophy of an SMT in sharing resources that are supposed to be less than that from multiple copies of single-threaded systems. Thus, effectively managing a reasonable-sized register file in an SMT system is a must to achieve a desirable balance between throughput and cost.

There are several research results related to organization and utilization of the register file. In [16], the utilization of physical register file is increased by expediting the deallocation process of the "dead" registers. Although a very significant improvement in performance is reported, the proposed early deallocation process itself is not a "stand-alone" hardware modification but instead can only be achieved through the support of an intelligent compiler and operating system. Another technique proposed in [17] instead tries to delay the register allocation until the complete stage (versus the rename stage) so as to reduce its occupancy latency. However, such a delayed allocation could easily lead to a deadlock since such a not-yet-allocated instruction may not find a free physical register to commit to when it is completed. Extra hardware overhead and modifications to other stages of the pipeline are needed to mend this problem.

To better solve this problem with a stand-alone approach without adding much hardware requirement, we propose a very intuitive technique by limiting the maximal number of physical registers a thread is allowed to occupy at any moment. We believe that, if this cap value is properly selected, slower threads will no longer clog up the register file impeding faster threads' progress for higher throughput. The technique is at the architectural level and thus requires no modification to the operation system or the compiler. In addition, it is a completely stand-alone process at the renaming stage which does not require any modification to other stages in the pipeline. This technique can therefore be easily combined with any other advanced techniques designed for another stage for additive performance gain without having to worry much about tampering existing benefits from each other.

As our simulation results show, the proposed technique improves IPC (Instruction per Clock Cycle) by as high as 44.6% and 32.7% in 4-threaded and 6-threaded systems respectively. It also enhances Harmonic IPC (an indicator of execution fairness) by 42.6% and 32.7% respectively. From the resource utilization's perspective, with the proposed technique, the system is able to deliver the same throughput with a smaller number of registers. For example, a 4-threaded system with a physical register file of 160 entries can deliver a performance comparable to that of a default system with 256 entries, reflecting a resource saving of 37.5%.

The rest of this paper continues with a brief description on register renaming and how it has been implemented in the literature. Section 3 is then devoted to the introduction of simulation environment adopted by this research including the metrics used. Motivation of this research is clearly illustrated in Section 4 giving several key analyses of the systems' performance with respect to the register file. The proposed technique is described in Section 5 followed by the complete simulation results in Section 6. It is then wrapped up by several concluding remarks in the last section.

2. Register renaming

Register renaming is a technique used to avoid unnecessary serialization of program operations imposed due to the reuse of the registers by these operations. It is a key issue for the out-of-order execution and is extensively used in modern processors. Register renaming eliminates name dependencies (anti- and output-dependencies) on architectural registers by assigning

Download English Version:

https://daneshyari.com/en/article/4956893

Download Persian Version:

https://daneshyari.com/article/4956893

Daneshyari.com