Microprocessors and Microsystems 35 (2011) 683–694

Contents lists available at SciVerse ScienceDirect

Microprocessors and Microsystems

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

A phase adaptive cache hierarchy for SMT processors

Sonia López [a,*], Óscar Garnica [d], David H. Albonesi [b], Steven Dropsho [c], Juan Lanchares [d], José I. Hidalgo [d]

[a] Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY, USA
[b] Computer Systems Laboratory, Cornell University, Ithaca, NY, USA
[c] Google Inc., Zurich, Switzerland
[d] Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Spain

ARTICLE INFO

Article history:
Available online 31 August 2011

Keywords:
Adaptive caches
Reconfigurable caches
Cache memories
GALS
Simultaneous Multi-Threading

ABSTRACT

Resizable caches can trade-off capacity for access speed to dynamically match the needs of the workload. In single-threaded cores, resizable caches have demonstrated their ability to improve processor performance by adapting to the phases of the running application. In Simultaneous Multi-Threaded (SMT) cores, the caching needs can vary greatly across the number of threads and their characteristics, thus, offering even more opportunities to dynamically adjust cache resources to the workload.

In this paper, we demonstrate that the preferred control methodology for data cache reconfiguring in a SMT core changes as the number of running threads increases. In workloads with one or two threads, the resizable cache control algorithm should optimize for *cache miss* behavior because misses typically form the critical path. In contrast, with several independent threads running, we show that optimizing for *cache hit* behavior has more impact, since large SMT workloads have other threads to run during a cache miss. Moreover, we demonstrate that these seemingly diametrically opposed policies are closely related mathematically; the former minimizes the arithmetic mean cache access time (which we will call AMAT), while the latter minimizes its harmonic mean. We introduce an algorithm (HAMAT) that smoothly and naturally adjusts between the two strategies with the degree of multi-threading.

We extend a previously proposed Globally Asynchronous, Locally Synchronous (GALS) processor core with SMT support and dynamically resizable caches. We show that the HAMAT algorithm significantly outperforms the AMAT algorithm on four-thread workloads while matching its performance on one and two thread workloads. Moreover, HAMAT achieves overall performance improvements of 18.7%, 10.1%, and 14.2% on one, two, and four thread workloads, respectively, over the best fixed-configuration cache design.

1. Introduction

Simultaneous Multi-Threading (SMT) [1,2] is a widely used approach to increase the efficiency of the processor core. SMT designs enable multiple threads to simultaneously share many of the major hardware resources, thereby making use of resources that may lie partially unused when running a single thread. SMT processors have the advantage of dynamically trading off instruction-level parallelism (ILP) for thread-level parallelism (TLP). That is, hardware resources that are partially unoccupied due to insufficient single-thread ILP can be used by instructions from other threads. Thus, the SMT approach attacks two important sources of limited throughput: long latency operations (such as accesses to the external memory) and limited per thread parallelism. This leads to a significant boost in instruction throughput over a single-threaded processor with only a modest increase in hardware resources [3].

However, the threads sharing the resources compete for those resources. Depending on the phase of the execution and the needs of each thread, this competition might cause thread resource starvation; that is, one thread may monopolize the resources, not allowing the others to progress through the pipeline. This fairness problem has been addressed usually from the point of view of more efficient fetch policies or dynamic resource allocation [2,4–6], although these approaches are not always successful in predicting long latency phases, causing thread starvation for some SMT workloads. In this paper, we propose a new and complementary approach to obtaining good throughput and fairness in a SMT processor for a variety of workloads. We tackle long latency operations from a new perspective: instead of fetching threads based on long-latency behavior, we propose phase-adaptive reconfigurable caches in a Globally Asynchronous, Locally Synchronous (GALS) design that, in conjunction with a cache control strategy, reduces the average latency of cache operations for the active threads.

* Corresponding author.
E-mail addresses: slaeec@rit.edu (S. López), ogarnica@dacya.ucm.es (Ó. Garnica), albonesi@csl.cornell.edu (D.H. Albonesi), stevendropsho@google.com (S. Dropsho), julandan@dacya.ucm.es (J. Lanchares), hidalgo@dacya.ucm.es (J.I. Hidalgo).

0141-9331/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2011.08.008

Our approach attempts to resolve the problems related with long-latency operations at the source by reducing the average latency of cache accesses. Moreover, our techniques are orthogonal to previously proposed fetch and dynamic resource allocation policies.

Using a GALS design approach, we place the reconfigurable caches into an independent clock domain within which frequency can change in conjunction with the cache configuration. The configuration for any given period of execution is established by a control algorithm that makes a decision based on the cache behavior of the different active threads.

A similar GALS approach was used in [7] for a single-threaded processor. We demonstate that the cache control strategy of [7] is not as effective for dual and four thread SMT workloads as it is for single thread ones. If we take into account fairness (the harmonic mean of the per thread speedups [8]), the performance of the four thread workloads degrades significantly.

The intuition behind the limited scalability of this prior strategy is that it is constructed on the assumption that cache misses are on the critical path of a thread's computation. Thus, the original strategy attempts to minimize the total access time to reduce the cost of the cache misses. However, when there are multiple active threads in an SMT core, the overall performance is affected less from cache misses because other threads can run in the shadow of the miss. In this scenario, a better cache control strategy is one that selects cache configurations that greedily maximize the near term cache access rate to favor threads that use the cache efficiently. While these two strategies seem diametrically opposed, in Section 3 we show that they are closely related mathematically; the first minimizes the arithmetic mean cache access time, while the second minimizes its harmonic mean.

Since the number of active threads on a given core may vary at runtime, the most effective approach should use aspects of both strategies depending on the number of active threads, i.e., minimize total access time when there are few threads but maximize the access rate when there are many threads. We propose and evaluate such a hybrid approach in this paper.

To demonstrate the effectiveness of our proposed cache control strategy, we implement a quad-threaded core in the Simplescalar simulator. The core is optimized to run with a small, fast cache that can adjust to greater demands by dynamically *upsizing*. We adopt the *Accounting Cache* design of [9] for our resizable caches, but implement a new cache control algorithm that better balances multi-threaded needs compared to the original algorithm designed for the single threaded case. The GALS design, in particular, the *Multiple Clock Domain* (*MCD*) approach of [10], decouples the adaptive caches from the execution core. Unlike [10], our MCD processor supports SMT and a different domain organization.

Our results demonstrate that the new hybrid control algorithm is effective at reacting to single, dual, and quad thread workload phase behavior. Whereas the first control algorithm performs well for single and dual thread workloads, giving an 6.2% overall improvement over the best synchronous processor baseline, the benefits from using this algorithm disappear under quad thread loads, averaging −3.4% In sharp contrast, the proposed hybrid algorithm adjusts well to heavy SMT workloads and generates consistent performance benefits across all workloads: 18.7%, 10.1%, and 14.2% for one, two and four thread workloads, respectively.

Our contributions in this paper are the following:

- We propose an SMT architecture that uses a GALS clocking design whereby the resizable L1 Dcache and L2 cache are located in their own separately clocked domain.
- We adapt the Accounting Cache of [9] to run with multi-threaded workloads.
- We show that the algorithm proposed for single-threaded workloads in [10] is not effective in a four-way SMT environment and explain the reasons why this is the case.
- We propose a new algorithm that adapts its strategy depending on the number of running threads and demonstrate that this new algorithm outperforms the previous one in terms of throughput and fairness.

This paper extends on [11]. We present further detail on the cache architecture and algorithms, new results, and deeper insight on the cache performance behavior.

The rest of this paper is organized as follows. The next section discusses the adaptive SMT MCD microarchitecture, including the adaptive cache organizations. Section 3 presents the adaptive cache control algorithms. Our simulation infrastructure and benchmarks are described next, followed by our results. Finally, we discuss related work in Section 6, and present our conclusions in Section 7.

2. Adaptive SMT MCD microarchitecture

The *adaptive SMT MCD microarchitecture* highlighted in Fig. 1 has five independent clock domains, comprising of the front end (L1 ICache, branch prediction, rename and dispatch); integer processing core (issue queue, register file and execution units); floating-point processing core (issue queue, register file and execution units); load/store unit (load/store queue, L1 DCache and unified L2 cache); and ROB (Reorder Buffer). The load/store domain varies its frequency based on the cache configuration. The other domains run at fixed frequency at all times and since there is little interaction between them (and thus their interface introduces negligible synchronization cost), they are effectively one fixed-frequency execution core domain. External main memory operates at the same fixed base frequency as the processing core and is also non-adaptive.

The focus of this study is the load/store domain having reconfigurable L1/L2 caches. To adjust to varying SMT workloads, we have extended the baseline MCD model to include SMT support; moreover, only the L1 DCache and L2 cache of the load/store domain are adapted under the direction of control algorithms that we introduce later. This adaptive SMT MCD architecture has a base configuration that uses small cache sizes running at a high clock rate, but the caches can be upsized with a corresponding reduction in the clock rate of the load/store domain. In this study, all the non-adaptive domains – front end, integer, floating point, and main memory – run at a base frequency of 1.0 GHz. The L1 DCache and L2 cache are resized in tandem with the frequency of the load/store domain varied accordingly. The dynamic frequency control circuit within the load/store domain is a PLL clocking circuit based on industrial circuits [12,13]. The lock time in our experiments is normally distributed with a mean time of 15 μs and a range of 10–20 μs. As in the XScale processor [12], we assume that a domain is able to continue operating through a frequency change.

Data generated in one domain and needed in another must cross a domain boundary, potentially incurring synchronization costs. Our SMT MCD simulator models synchronization circuitry based on the work of Sjogren and Myers [14]. It imposes a delay of one cycle in the consumer domain whenever the distance between the edges of the two clocks is within 30% of the period of the faster clock. In [15], the authors showed that in a single-thread core, both superscalar execution (which allows instructions to cross domains in groups) and out-of-order execution (which reduces the impact of individual instruction latencies), tend to hide much of the synchronization costs. Further details on the baseline MCD model, including a description of the inter-domain