Fully Associative Cache Partitioning with Don't Care Bits for Real-Time Applications

Ali Chousein Rabi N. Mahapatra
Department of Computer Science, Texas A&M University
{chousein,rabi}@cs.tamu.edu

Abstract

The usage of cache memories in time-critical applications has been limited as caches introduce unpredictable execution behavior. Cache partitioning techniques have been developed to reduce the impact of unpredictability owing to context switch effects. However, partitioning reduces the cache size available for each task resulting in capacity related cache misses. This paper introduces a fully associative cache architecture for multi-tasking applications where effective partition sizes are increased by tag compression in the cache. The proposed scheme uses a few don't care cells in its least significant bits of the tag to aggregate multiple tag entries into a single entry. The experimental results indicate that the proposed scheme is context switch resilient when eight different real-time benchmarks use the cache concurrently. Further, this cache architecture requires less time and less energy to perform tag table search compared to contemporary fully associative caches of the same size.

1. Introduction

Cache memories are commonly used resources in multitasking environments. Upon a context switch, a newly scheduled task changes the cache contents by replacing existing entries used by other tasks. With the cache as a common resource, the tasks uncontrollably affect the execution times of each others due to unpredictable compulsory cache misses. This in turn adversely affects the tight bound estimation of the worst case execution times of critical applications. Partitioning the cache and allocating each partition exclusively to different tasks is a well known method of achieving a predictable environment. However this leads to cache misses due to the reduced cache size available in the partition for each task. In fully associative cache organization, although the cache misses due to the address contention is absent, the tasks' execution time becomes unpredictable due to compulsory and capacity type cache misses. Any attempt to keep these cache misses predictable deserves attention.

In the past, both hardware and software based cache partitioning have been proposed by many researchers [1-4]. The hardware partitioning method has been discussed in [1] where Kirk proposed the partitioning of direct mapped and set associative caches. Software based cache partitioning, introduced by Wolfe [2], is based on partitioning the address space of the processor. Mueller [3] discussed the compiler and linker support needed for automating software based cache partitioning. Liedtke et al. [4] introduce the method of free coloring of memory pages for improving software based cache partitioning in terms of memory space, required by each partition.

In this paper, we have introduced a hardware based cache partitioning mechanism for fully associative cache architecture. It features with a small tag table size to keep the search time low and a larger data table to hold enough items in the partitions. Using few don't care cells in its tag we adopt compaction of tag entries of fully associative cache memory. The don't care cells are implemented using ternary content addressable memory (TCAM) cells. Using a trace driven multitasking simulator and eight real-time benchmarks, our experiments show the miss ratio remains consistent and predictable in spite of varying task context switch period. The proposed architecture consumes less energy and requires less search time compared to contemporary fully associative cache architecture.

The organization of this paper is as follows. Section 2 gives a general overview of our research. Section 3 presents the fully associative cache architecture that is considered for partitioning. The experimental results that empirically justify the effectiveness of the proposed architecture are given in Section 4. Finally

Section 5 concludes this work. The terms *task* and *process* are used interchangeably throughout the text.

2. Research Overview

The proposed fully associative cache architecture employs TCAM cells in the last significant *L* bits of the tag entries to compact tag table. Each TCAM cell can store don't care state (x) in addition to the regular 0 and 1 binary states. This don't care state is used as a wild bit to aggregate multiple entries in the tag table to single entry to achieve tag table compaction. This scheme is different from traditional tag compression schemes [11]. As an example, the compaction can be effective up to eight times in reducing the tag table size with three don't care bits in the tag when program locality behaves favorably. Due to tag table compaction, it is possible to build large sized caches with fewer number of tag entries. This in turn provides larger working cache area for each task when partitioning takes place and improves the miss ratio performance.

The overhead to maintain the above compaction comes with a price of additional hardware but no additional time overhead. The aggregation process that is responsible for tag compaction is neither on the critical path of cache access nor needs extensive hardware. The details of the aggregation module are not discussed in this paper. The decoding of the compacted tag entry is done concurrently with the tag table search and does not affect the critical path of the cache access. The decoder is implemented using off-the-shelf de-multiplexers.

3. Fully Associative Cache for Partitioning

Fully associative cache supports the flexibility to place the contents of a memory location in any cache line. This means there is less contention for the same cache lines. In the sub-sections that follow we introduce the architectural design of the fully associative cache suitable for partitioning and discuss the cost of its implementation.

3.1 Architectural Design

Figure 1 shows the fully associative cache architecture with tag and data table entries. Each tag entry has additional bits to store the process ID number (PID) of a task T_i (PID^{Ti}). A tag entry ℓ_α is declared as private by setting bit *P* (not shown in this figure) in that entry.

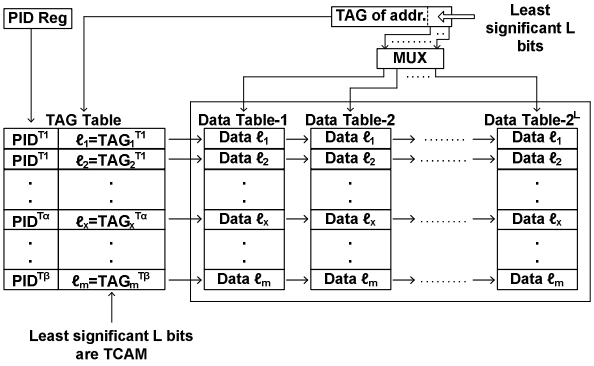

Figure 1. Fully associative cache architecture

In Figure 1, each tag table entry has been built using content addressable memory (CAM) cells in conjunction with a few bits of ternary content addressable memory (TCAM) cells. Traditionally the tag table entries in a fully associative cache are built with CAM cells only [7]. In [8] it has been shown that integration of CAM and TCAM cells in TLB design enhances the TLB reach and reduces miss ratio. In this design the tag table is relatively small in size and the data table with SRAM is large. We adopt this technique for designing fully associative cache architecture with a compact tag table. In Figure 1 the least significant *L* bits of each tag table entry ℓ_α are TCAM cells. The rest of the tag bits and the augmenting bits are CAM cells. There are 2^L many data lines associated with each tag table entry ℓ_x. By setting one or more TCAM cells of a tag table entry ℓ_x to don't care state, multiple SRAM entries (the number is always a power of 2) are associated with ℓ_x. The SRAM entries associated with ℓ_x are stored from left to right in the data tables. Let us say the entries ℓ_1 and ℓ_2 are used by task T_1 and the entry ℓ_m is used by task T_β as shown in Figure 1. If the private bits of ℓ_1, ℓ_2 and ℓ_m are set, then task T_1 owns a partition of $2 \cdot 2^L$ bytes, and task T_β owns a partition of 2^L bytes. Note that each task can use the maximum of the partition space allocated to it, if the program locality behaves favorably.

Like XScale processors [6], we use seven bits for the PID one bit for storing bit P. This requires one byte overhead for each tag table entry.

3.2 Aggregation of Tag Entries

Multiple SRAM entries are aggregated together such that they are associated with a single tag entry to create a consolidated partition segment in the cache. If a tag table entry has *D* TCAM cells set to don't care

state, then 2^D many SRAM entries are aggregated together. In the following we present the definitions and the steps involved in the aggregation.

Definition 1 – Basic Aggregation: *Two tag table entries ℓ_α and ℓ_β aggregate if the following three conditions are satisfied:*

1. *The entries ℓ_α and ℓ_β have at least one TCAM cell with a value other that don't care,*
2. *The entries ℓ_α and ℓ_β have the same number of TCAM cells set to don't care,*
3. *The entries ℓ_α and ℓ_β differ by the least significant TCAM bit only that is not set to don't care.*

If a new cache line ℓ_β is nadmitted to cache none of the TCAM cells in ℓ_β is set to don't care.

Definition 2 – Escalated Aggregation: *Escalated aggregation is performed between two tag table entries ℓ_α and ℓ_β that already exist in the cache. Aggregation is performed if the three conditions given in Definition 1 are satisfied.*

When a newly allocated line ℓ_β aggregates with an existing line ℓ_α, the type of this aggregation is basic. This basic aggregation sets the least significant TCAM cell of ℓ_α to don't care to obtain ℓ_α'. If ℓ_α' aggregates further with another entry in the cache, this second aggregation is known as escalated aggregation. It is possible to have a chain of escalated aggregations before a tag entry is maximally compacted.

Aggregation Steps: *When two tag table entries ℓ_α and ℓ_β aggregate, the following actions are taken.*

1. *If the least significant TCAM cell of ℓ_α that is not set to don't care is equal to zero, then ℓ_l=ℓ_α and ℓ_h=ℓ_β; else, ℓ_l=ℓ_β and ℓ_h=ℓ_α,*
2. *The least significant TCAM cell of ℓ_l that is not set to don't care, is set to don't care,*
3. *The SRAM entries associated with ℓ_h and that contain valid information are copied to the SRAM entries associated with ℓ_l, starting from the first entry from the left that does not store valid information,*
4. *If ℓ_h is not a newly allocated entry, it is set to invalid.*

Aggregation of the tag entries is done when a new entry is admitted to the cache and it does not affect the critical path of cache access.

4. Experimental Results

We designed experiments for studying the impact of context switch frequency and aggregation on cache miss performance when considered with and without cache partitioning. Traces of eight SNU Real-Time Benchmarks [9] (insertsort, jfdctint, ludcmp, matmul, minver, qsort, qurt and select) have been used as input to a trace driven fully associative instruction cache simulator that simulates the architecture explained in Section 3. The tag table consists of 64 entries and cache line size is 32 bytes. The experiments that allow aggregation use three TCAM cells in the least significant bits of the tag table entries. These parameters yield cache sizes of 16KB and 2KB when there is aggregation and there is not aggregation respectively. The traces are scheduled using the round-robin approach with a selectable context switch period which is measured in terms of instruction cycles. We measure the average cache miss ratio of eight benchmarks when the cache is partitioned and without being partitioned. The average cache miss ratio of both cases is measured with and without aggregation. When the cache is partitioned each task (a task corresponds to a benchmark in our experiments) is allocated an equal portion of the cache (the partition size is equal to the cache size divided by the number of tasks).

Figure 3 shows four graphs that display the effects of the context switch period on the average cache miss ratio. The top two and bottom two graphs are for 2KB (no aggregation) and for 16 KB (with aggregation) instruction caches respectively. When the cache is not partitioned a change in context switch frequency affects the cache miss ratio. In this case the cache miss ratio decreases with increasing context switch period as expected since the effect of compulsory misses is less there. However, when the cache is partitioned, the cache miss ratio becomes independent of the context switch frequency indicating the fact that such a scheme will not adversely affect the predictability of the worst case execution time. Anomalies occur when the cache miss ratio depends on context switch frequency. As seen in the bottom non-partitioned graph of Figure 3, when the cache is not partitioned and the context switch period is equal to 700 instruction cycles, the cache miss ratio is higher than when the context switch period is 650 instruction cycles. Such anomalies do not occur when the cache is partitioned, because the cache miss performance becomes independent of the context switch frequency.

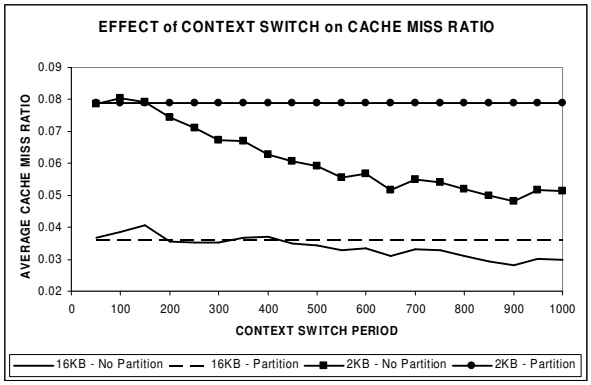

Figure 3. Effects of context switch on cache miss performance

We used the CACTI model [10] for estimating the time and energy needed for making tag table search for the proposed and the contemporary fully associative cache architectures. Table 1 displays the parameters supplied to CACTI. The tag table size of the proposed architecture with three TCAM cells is one eighth of the tag table size of contemporary architectures. To reflect this difference the size of the proposed cache architecture given as input to CACTI is one eighth of the contemporary cache sizes. Table 2 compares the time needed and the energy consumed for making tag table search by the proposed and the contemporary fully associative cache architectures.

CACTI PARAMETERS					
	Cache Size	Line Size	Assoc	Techn	Banks
PROPOSED	0.5KB–8KB	32	FA	0.18	1
CONTEMPORARY	4KB–64KB	32	FA	0.18	1

Table 1. CACTI parameters

	PROPOSED		CONTEMPORARY	
CACHE	TIME	ENERGY	TIME	ENERGY
4KB	1.29 ns	0.045 nJ	1.42 ns	0.243 nJ
8KB	1.45 ns	0.071 nJ	1.72 ns	0.407 nJ
16KB	1.50 ns	0.169 nJ	2.30 ns	0.735 nJ
32KB	1.50 ns	0.311 nJ	2.37 ns	1.537 nJ
64KB	1.80 ns	0.514 nJ	3.55 ns	2.851 nJ

Table 2. Time and energy requirements comparison

5. Conclusion and Future Work

We proposed a fully associative cache architecture that is easy to partition and has good performance in terms of access time and energy consumption. Future research will extend this work to set associative caches. We will also develop a demand driven adaptive cache partitioning and incorporate it into the TCAM based cache architecture research.

10. References

[1] D. B. Kirk, "SMART(Strategic Memory Allocation for Real-Time) Cache Design", *Proceedings of the Tenth Real-Time Systems Symposium*, 1989, pp. 229-237.

[2] A. Wolfe, "Software-Based Cache Partitioning for Real-Time Applications", *Workshop on Responsive Computer Systems*, 1993, pp. 174-180.

[3] F. Mueller, "Compiler Support for Software-Based Cache Partitioning", *ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995, pp. 125-133.

[4] Jochen Liedtke, Hermann Härtig and Michael Hohmuth, "OS-Controlled Cache Partitioning for Real-Time Systems", *Third IEEE Real-time Technology and Application Symposium*, 1997, pp. 213-224.

[5] Swagato Basumallick and Kevin D. Nilsen, "Cache Issues in Real-Time Systems", *Proceedings of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, 1994.

[6] Intel, "Intel® XScale™ Core, Developer's Manual", 2000.

[7] T. Kohonen, *Content-Addressable Memories*, Second Edition, Springer-Verlag, 1987.

[8] A. Kumar and R. Mahapatra. "Enhancing TLB Reach with Ternary-CAM Cells", *Technical Report* 2004, Texas A&M University

[9] http://archi.snu.ac.kr/realtime/benchmark/

[10] http://research.compaq.com/wrl/people/jouppi/CACTI.html

[11] A. R. Alameldeen and D. A. Wood, "Adaptive Cache Compression for High Performance Processors", *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA-31)*, June 2004