

AN ENERGY EFFICIENT TCAM ENHANCED CACHE ARCHITECTURE

A Thesis

by

JASON MATHEW SURPRISE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2005

Major Subject: Computer Science

AN ENERGY EFFICIENT TCAM ENHANCED CACHE ARCHITECTURE

A Thesis

by

JASON MATHEW SURPRISE

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Rabi N. Mahapatra
(Chair of Committee)

Eun Jung Kim
(Member)

Gwan Choi
(Member)

Valerie Taylor
(Head of Department)

May 2005

Major Subject: Computer Science

ABSTRACT

An Energy Efficient TCAM Enhanced Cache Architecture. (May 2005)

Jason Mathew Surprise, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Rabi N. Mahapatra

Microprocessors are used in a variety of systems ranging from high-performance super computers running scientific applications to battery powered cell phones performing real-time tasks. Due to the large disparity between processor clock speed and main memory access time, most modern processors include several caches, which consume more than half of the total chip area and power budget. As the performance gap between processors and memory has increased, the trend has been to increase the size of the on-chip caches. However, increasing the cache size also increases its access time and energy consumptions. This growing power dissipation problem is making traditional cooling and packaging techniques less effective thus requiring cache designers to focus more on architectural level energy efficiency than performance alone.

The goal of this thesis is to propose a new cache architecture and to evaluate its efficiency in terms of miss rate, system performance, energy consumption, and area overhead. The proposed architecture employs the use of a few Ternary-CAM (TCAM) cells in the tag array to enable dynamic compression of tag entries containing contiguous values. By dynamically compressing tag entries, the number of entries in the tag array can be reduced by 2^N , where N is the number of tag bits that can be compressed. The

architecture described in this thesis is applicable to any cache structure that uses Content Addressable Memory (CAM) cells to store tag bits.

To evaluate the effectiveness of the TCAM Enhanced Cache Architecture for a wide scope of applications, two case studies were performed – the L2 Data-TLB (DTLB) of a high-performance processor and the L1 instruction and data caches of a low-power embedded processor. Results indicate that a L2 DTLB implementing 3-bit tag compression can achieve 93% of the performance of a conventional L2 DTLB of the same size while reducing the on-chip energy consumption by 74% and the total area by 50%. Similarly, an embedded processor cache implementing 2-bit tag compression achieves 99% of the performance of a conventional cache while reducing the on-chip energy consumption by 33% and the total area by 10%.

To my parents, who worked very hard
so that I could pursue an education.

ACKNOWLEDGMENTS

I am very thankful to my advisor Dr. Rabi Mahapatra for his guidance throughout the course of my research. His support and willingness to work with me during late evening hours and over the weekends is greatly appreciated. I also want to thank Dr. Gwan Choi and Dr. Eun Jung Kim for being a part of my committee.

I would like to thank the students in Dr. Mahapatra's Embedded Systems Co-Design group for their suggestions that arose during numerous discussions we engaged in during the course of this research. I would especially like to thank Rupak Samanta who developed the VLSI circuit design of the Dynamic Aggregator Module and performed the SPICE simulations that were needed to complete many of the energy and area estimates. Without Rupak's help, it would not have been possible to complete this research. Finally, I would like to thank everyone who has either directly or indirectly helped me complete my research and thesis.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
ACKNOWLEDGMENTS	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	ix
LIST OF TABLES.....	x
CHAPTER	
I INTRODUCTION	1
A. Cache Basics	1
B. The Need for Virtual Memory and TLBs	4
C. The Growing Power Dissipation Problem.....	5
D. The Use of CAM Based Tag Arrays in Caches.....	5
E. Problem Statement.....	7
F. Related Work.....	9
G. Organization	13
II DYNAMIC TAG AGGREGATION.....	14
A. Tag Compression.....	14
B. Dynamic Compression	15
III TCAM ENHANCED CACHE ARCHITECTURE.....	18
A. CAM/TCAM Cache	18
B. Dynamic Aggregator Module.....	20
C. Impact on Cache Access Time	25
D. Set-Associative Implementation.....	26
E. Estimating Energy Consumption.....	27
IV EFFECTIVENESS OF DYNAMIC TAG AGGREGATION IN THE L2 DTLB	29
A. Simulation Methodology	29

CHAPTER	Page
B. Results and Analysis.....	30
V EFFECTIVENESS OF DYNAMIC TAG AGGREGATION IN THE L1 CACHES	37
A. Simulation Methodology	37
B. Results and Analysis.....	38
VI CONCLUSIONS AND FUTURE WORK.....	46
REFERENCES	48
VITA.....	53

LIST OF FIGURES

FIGURE	Page
1 Example of Tag Aggregation Process	14
2 Sequential TCAM Based Tag Aggregation Process.....	16
3 CAM/TCAM Cache with 3 TCAM Cells.....	19
4 CAM/TCAM Cache with DAM Hardware	21
5 Set-Associative Mappings for an Address of W Bits.....	27
6 L2 DTLB Miss Rates Normalized to Equivalent Tag DTLB.....	31
7 IPC Normalized to Equivalent Data DTLB.....	33
8 On-Chip Energy Consumption Normalized to Equivalent Data DTLB.....	34
9 Total Energy Consumption Normalized to Equivalent Tag DTLB.....	35
10 Instruction Cache Miss Rates Normalized to Equivalent Associativity Cache.....	39
11 L1 Data Cache Miss Rates Normalized to Equivalent Associativity Cache	40
12 IPC Normalized to Equivalent Data Caches.....	42
13 On-Chip Energy Consumption Normalized to Equivalent Data Caches.....	43
14 Total Energy Consumption Normalized to Equivalent Associativity Caches.....	44

LIST OF TABLES

TABLE	Page
1 L2 DTLB Test Configurations	30
2 Transistor Counts for L2 DTLB	36
3 L1 Cache Test Configurations.....	38
4 Transistor Counts for a 16 KB Cache.....	45

CHAPTER I

INTRODUCTION

Microprocessors are used in a variety of devices ranging from high-performance super computers running scientific applications to battery powered cell phones performing real-time tasks. It is well known that there exists a growing disparity between processor and main memory performance. Since 1986, processor performance has increased nearly 55% per year while DRAM performance has improved only 7% per year [1]. Moreover, the International Technology Roadmap for Semiconductors (ITRS) report predicts that this trend will continue for the foreseeable future [2]. This means that as new processors are developed with faster clock rates, each main memory access becomes more costly in terms of clock cycles. By far, the most widely employed solution to overcoming this performance gap is the use of a memory hierarchy consisting of on-chip caches that can be accessed at or nearly at the processor's native speed depending on the size and configuration of the caches.

A. CACHE BASICS

The most widely used cache hierarchy configuration is small, separate level 1 (L1) caches for instructions and data that can be accessed in a single clock cycle and a single, large unified level 2 (L2) cache containing both instructions and data that can be accessed in only a few clock cycles. When the processor requests data or instructions, the L1 caches

This thesis follows the style of *IEEE Transactions on Computers*.

are searched, and if the information is not found, the L2 cache is examined. If both the L1 and L2 caches do not contain the requested data, the information is retrieved from main memory and deposited into the caches.

Caches are effective at improving system performance because the applications executed by a processor obey the principle of locality, which states that programs are likely to reuse the data and instructions they used recently. For example, it is a widely used rule of thumb that applications spend 90% of their time executing 10% of their code. The principle of locality can be decomposed into spatial and temporal locality properties. The spatial locality property infers that data items whose addresses are close to each other are likely to be referenced around the same time. Due to the spatial locality property, data entries that surround a data item requested from main memory are also retrieved when a cache miss occurs since it is likely that they will be referenced in the near future. This group of data entries constitutes a single cache line. The temporal locality property infers that the data items that were recently requested are likely to be referenced in the near future. Based on these principles, it is clear why relatively small caches can result in significant performance increases.

A cache can be described based on four parameters – the number of sets, the associativity of each set, the cache line size, and the replacement policy. The entire address space accessible by the processor is partitioned based on the number of cache sets such that an address that maps to a particular set can be placed in any tag entry belonging to that set. The associativity of the set defines how many tag entries are in each set. There are two special cases for associativity. A cache having an associativity of one is typically referred

to as a direct mapped cache, and a cache that has only a single set containing all the tag entries is referred to as a fully associative cache. All other cache configurations are referred to as n -way set associative where n is the number of tag entries in each set.

A cache consists of a data array and a tag array such that each tag entry corresponds to a single cache line in the data array. The cache data array is typically composed of SRAM cells; however, the composition of the tag array depends on the configuration of the cache. Lowly-associative caches, from direct mapped to 16-way set associative, usually have SRAM based tag arrays that require the tags to be read from the tag array and then compared to the address. Highly-associative caches, from 32-way set associative to fully associative, typically store tag entries using Content Addressable Memory (CAM) cells. A major benefit of CAM cells over SRAM cells is that each CAM cell has its own comparator logic allowing much faster comparisons be made for a large number of tags.

Caches can incur three different types of misses – compulsory, conflict, and capacity misses. Compulsory misses result when the processor requests a data item for the first time. The size and configuration of the cache has no direct impact on the number of compulsory misses; however, they do have a significant effect on the other miss types. Conflict misses occur only in set associative caches when a cache entry belonging to a set is replaced by a new entry and later is requested again by the processor; the recurring miss occurs because of a conflict within the set. When the cache becomes full and the processor continues to request data items not in the cache, it must replace entries. If the replaced entries are later referenced again by the processor, a capacity miss occurs because the cache

was not large enough to hold all the data needed during program execution. Therefore, to obtain the lowest miss rates, a cache needs to be large and highly-associative.

B. THE NEED FOR VIRTUAL MEMORY AND TLBS

Typically, a system's physical main memory is not large enough to hold all the applications that the system is executing concurrently. To remedy this limitation, a virtual address space is defined that is significantly larger than the physical address space, thus allowing the small physical memory to be shared among the executing processes. Since the virtual address space is larger than the available physical memory, an application's virtual address must be translated to a physical address before its instructions and data can be accessed. The management of the virtual and physical address mappings is usually handled by the operating system.

Since virtual to physical address translation must be performed for every instruction and data access, the translation process has become one of the most critical operations in modern processors. In order to expedite address translation, a special cache called the Translation Lookaside Buffer (TLB) caches the most recently referenced translations. TLBs are in general small fully-associative caches that store a single address translation per cache line. The storage capacity of the TLB is determined based on the number of virtual to physical address translations that can be stored, and it is generally referred to as the *TLB reach*. Since the TLB must be accessed each clock cycle for an instruction fetch and data reference, a processor is usually designed with separate TLBs for instructions (ITLB) and data (DTLB).

C. THE GROWING POWER DISSIPATION PROBLEM

Over the past few decades, the semiconductor industry has stayed on par with Moore's Law by doubling the transistor density on a chip every two years. Similar to the processor-memory performance trend, this trend is also predicted to continue in the ITRS report [2]. By increasing transistor density, it becomes possible to place larger caches on the chip, which can increase system performance by reducing capacity misses. However, increasing the size of caches has serious drawbacks because as transistor counts increase so does the power dissipation. In fact, traditional cooling and packaging techniques are becoming less effective at reducing the adverse effects of increased power consumption making it more important that designers consider energy efficiency as a major design goal. Power consumption is especially problematic for microprocessors used in embedded systems since most of these systems rely on batteries to supply them with energy. Despite these power consumption issues, processors continue to be developed with increasingly larger on-chip caches making it necessary to evaluate alternative energy efficient architectures in order to satisfy both performance and power requirements.

D. THE USE OF CAM BASED TAG ARRAYS IN CACHES

Highly-associative caches tend to have lower overall miss rates than other types of caches. However, in spite of the lower miss rates, the on-chip energy needed to search the CAM based tag array and the area consumed by the CAM cells is often too expensive unless the cache is sufficiently small or unless main memory accesses must be minimized to reduce total power consumption. Two such cases that employ CAM based tag arrays are

the Translation Lookaside Buffer and the L1 instruction and data caches of low-power embedded processors.

Many modern low-power embedded processors such as XScale [3] and StrongARM [4, 5] utilize highly-associative caches in order to reduce miss rates thus lowering the number of energy expensive accesses to other memory systems. Highly-associative cache designs using CAM based tag arrays have been shown to be more power efficient and have comparable access times to traditional set-associative caches employing SRAM based tag arrays [6]. It is well known that low-power embedded processor caches using CAM based tag arrays consume a significant portion of the total chip area, 60% for XScale [3], as well as a significant portion of the total power, nearly 43% for StrongARM [4]. For these reasons, caches using CAM based tag arrays are a major candidate for power reduction optimizations.

While embedded processors are mainly concerned with minimizing total power consumption, high-performance processors are designed to maximize the average system performance. Since a significant amount of processing time is spent by the operating system kernel handling each TLB miss, high-performance processors use small fully-associative TLBs in order to minimize the number misses [7, 8, 9]. Also as instruction level parallelism, processor clock frequency, and application working set size increases, the impact of TLB performance on the total application processing time will continue to grow. Although TLB miss ratio should be kept as low as possible in order to achieve the best performance, improving it generally results in degradation of the TLB access time due to the inverse relationship among access time and cache capacity. Therefore, in order to

satisfy future performance requirements, it is necessary to develop architectures that increase the TLB reach without adversely affecting the access time or energy consumption.

Although highly-associative caches significantly reduce off-chip memory accesses, nearly 50% of the total cache access energy is spent searching the CAM tag array for a tag match [10]. Therefore, reducing the number of tag entries needed to access the data entries appears to provide a source for significant power reduction.

E. PROBLEM STATEMENT

Based on experimental results obtained in [11], TCAM based tag compression has been shown to effectively reduce the miss rates of a L2 DTLB; however, no results were given as to the effects on system performance or energy consumption. Also, no experiments were performed to examine the effectiveness of TCAM based tag compression on highly-associative caches. The architecture proposed in [11] incurs extra latency during an access because it uses pointers to map between the tag and data arrays. The goal of this thesis is two fold. The first goal is to propose an efficient TCAM enhanced cache architecture that uses tag compression (aggregation) to reduce the number of CAM based tag entries needed to map to the data array. The second goal is to evaluate the proposed architecture's efficiency based on system performance, energy consumption, and area overhead.

The proposed architecture employs the use of a few Ternary-CAM (TCAM) cells [12] in the tag array to enable dynamic compression of tag entries containing contiguous values. By compressing tags, a single tag entry is capable of mapping to multiple cache lines thus reducing the total number of tags needed to access all the data entries. The TCAM based

aggregation technique does not incur any decompression overhead when searching the cache since CAM and TCAM access times have been shown to be equivalent [13]. Also, since tag compression is only performed when a cache miss occurs and can be finished before the missed entry is retrieved from main memory, the compression latency is completely hidden by the memory access.

The tag aggregation scheme discussed in this thesis is applicable to any cache structure that employs CAM based tag arrays. As discussed previously, two distinct areas that use CAM based tag arrays are high-performance processor TLBs and low-power embedded processor caches. The benefits of tag aggregation can be examined under two lights. First, performing tag aggregation frees tag entries, which increases the number of address to data mappings that can be stored thus increasing the effective storage capacity of the cache and the overall system performance. In this case, the number of tag entries is the same in both the compressed and uncompressed caches. Although the capacity is increased, the cache access time is unaffected because it is predominantly influenced by the number of tag entries that must be searched. Since access time is critical to high-performance processors, this aspect of tag aggregation will be evaluated based on a case study of a high-performance processor's L2 DTLB. The other benefit of tag aggregation is that it reduces the number of tag entries required to map to a fixed number of data entries. In this case, the number of tag entries is reduced by 2^N , where N is the number of bits considered for aggregation. Although the storage capacity remains the same in both the compressed and uncompressed caches, the power and area consumed by the cache can be drastically decreased due to the reduction in tag entries. To evaluate this second facet of tag

compression, the L1 instruction and data caches of a low-power embedded processor will be taken as a case study.

F. RELATED WORK

1. Improving TLB Performance

Over the past few years, a considerable amount of research has been done to improve the TLB miss rate and performance; however, no previous work has examined the energy efficiency of a TLB employing tag compression in the CAM based tag array. Two popular approaches to decreasing the TLB miss rate are increasing the TLB reach and prefetching. TLB reach is the total physical address space mapped in the TLB and is typically equal to the product of the number of entries in the TLB and the page size of each entry. The simple solutions to increase TLB reach are to increase the page size and to increase the number of TLB entries; but neither approach is practical due to the area and timing constraints of most modern processors. The proposed TCAM enhanced architecture is able to effectively increase the TLB reach without affecting the access time.

Superpages have been proposed to increase TLB reach by combining several properly aligned pages having continuous virtual address spaces into a single “superpage”. Superpages have a single TLB entry, which frees up entries to store additional mappings thus increasing the TLB coverage. There are many issues involved with superpage management including allocation (copying or reservation based), fragmentation control, promotion, demotion, and eviction of superpages [14]. Online superpage promotion [15] is a scheme for dynamically creating superpages based on TLB misses occurring during

execution. Supporting superpages requires significant overhead in the Memory Management Unit (MMU) and the operating system [16]. The TCAM based compression scheme is similar in spirit to superpages; however, the design does not require support from the operating system or the MMU.

Shadow memory is an address space between virtual and physical address spaces. Shadow memory has been proposed to support superpages without needing much support from the operating system [17, 18, 19]. The TLB stores the virtual to shadow address space mapping rather than the virtual to physical translation. This scheme requires a secondary MMU and TLB in the main memory controller. This memory-controller TLB (MTLB) translates shadow addresses to physical addresses. Similar to the superpage schemes discussed previously, this scheme also requires significant architectural changes.

Another method used to decrease the TLB miss rate is to predict the TLB entries that will be used in the future and prefetch them before they are referenced. The authors in [8] and [20] consider prefetching TLB entries only during a cold start, which reduces only compulsory TLB misses. Since TLB misses tend to exhibit some regularity [21], detecting the access pattern can significantly help in reducing recurring TLB misses. Several pattern based prefetching schemes have been proposed including recency based prefetching [21] and distance prefetching [22]. Although prefetching techniques can be effective at reducing miss rates, their effectiveness is predominantly dependent on their prediction accuracy; however, the proposed scheme is focused on reducing miss rates by increasing the storage capacity of the TLB.

2. Cache Compression

Caches are known to consume significant portions of the processor power budget and have therefore been the focus of a considerable amount of power reduction research. Cache compression makes it possible to use a smaller cache to achieve significant power savings while maintaining nearly the same performance of a larger conventional cache. Compression techniques can be applied in a cache's data or tag array. The architecture being proposed in this paper targets power reduction through tag compression, which decreases the number of tags needed to address the data values.

Data compression has been shown to be effective at increasing performance due to its ability to increase the effective storage capacity [23, 24]. In [23], an adaptive compression policy is used, which can at most double the storage capacity of the cache. To implement their design, a considerable amount of overhead, 7% of a 4MB cache, is required. The Compression Cache in [24] centers on representing frequently occurring data values with an alternate encoding scheme allowing two compressed lines to be stored in the space of a single uncompressed entry. The proposed compression scheme differs in that it alters the representation of the tag values not the data values, so it can avoid the data decompression penalty suffered in [24]. The schemes discussed above place the compression and decompression latencies in the critical path of the cache access. As discussed in [25], decompression penalties can be so large that they might supersede the benefits of compression unless efforts are made to hide the latencies. The proposed scheme is unique in that there is no penalty for decompression and that the compression latency is completely hidden by the main memory access. Additionally, the TCAM enhanced

architecture requires minimal implementation overhead while increasing the storage capacity of a cache with an equivalent number of tag entries by up to 2^N times where N is the number of tag bits that can be compressed.

In [26], static tag compression was shown to considerably reduce cache energy consumption. To realize these energy savings, however, an application must be analyzed offline for tag regions that are suitable for compression. This scheme required major modifications be made to the operating system and underlying hardware architecture. The proposed tag compression technique performs dynamic aggregation of tag entries as the processor requests them, and it is able to obtain comparable energy reduction while only requiring minimal hardware overhead to perform the tag compaction. Another tag reduction method called Caching Address Tags (CAT) was proposed in [27]. The CAT cache was focused on reducing the area occupied by a direct mapped cache by storing only a small number of unique tags in the tag array and storing a pointer to the data entry's tag with each cache line in the data array. Their scheme enforces a reverse cache lookup policy where the data array is accessed first and then the tag is checked, so both the data and tag array are accessed even if a miss occurs. The CAT cache has several implementation difficulties related to replacing tag entries and invalidating all the associated data entries. Although the idea of tag entry reduction is similar in the TCAM based compression scheme, the proposed architecture does not require any pointers, and it focuses on area and power reduction for highly-associative caches using CAM based tag arrays.

G. ORGANIZATION

Chapter I gives motivation for the TCAM enhanced cache architecture and describes previous related work. The remaining chapters of the thesis are organized as follows: the Dynamic Tag Aggregation process is introduced in Chapter II. The details of the TCAM Enhanced Cache Architecture are given in Chapter III. The Simulation Methodology and Experimental Results of the L2 DTLB case study are in Chapter IV, and Chapter V details the experiments performed for the L1 Instruction and Data Cache case study. Chapter VI concludes the thesis and discusses future research directions.

CHAPTER II

DYNAMIC TAG AGGREGATION

A. TAG COMPRESSION

Tag Aggregation is the process of compressing several tag entries that differ only in their least significant bits into a single entry. Tag aggregation can reduce the energy and area consumed by a cache while maintaining comparable performance to an uncompressed cache. This is accomplished because tag aggregation reduces the number of tag entries needed to map to all the data array entries. Tag aggregation is also able to increase the effective storage capacity of a cache by increasing the number of data entries mapped by a single tag entry. Although the capacity is increased, the cache access time is unaffected because the number of tag entries that must be searched is the same in both the compressed and uncompressed caches.

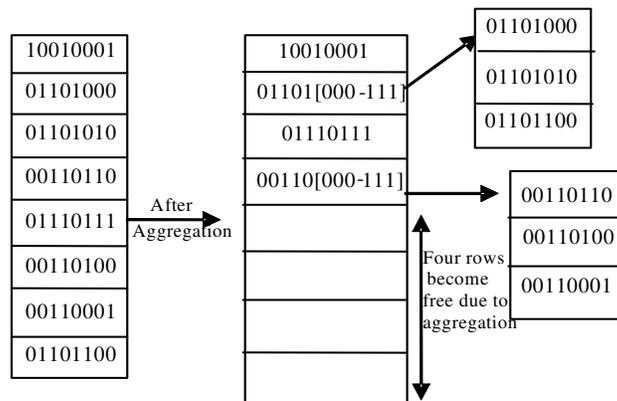


Figure 1: Example of Tag Aggregation Process

In Figure 1, the tag aggregation process is illustrated by an example where two entries are aggregated if they differ in only their three least significant bits. Let (10010001, 01101000, 01101010, 00110110, 01110111, 00110100, 00110001, and 01101100) be the tag values in a cache of size eight entries. Notice that the tag values 01101000, 01101010, and 01101100 have the same value except for their three least significant bits. After aggregation, a single entry corresponds to these three tags thus freeing two cache entries. Similarly with the 00110110, 00110100 and 00110001 tag values, an additional two entries are freed due to aggregation.

B. DYNAMIC COMPRESSION

Tag aggregation can be performed in one of two ways. Either the aggregation is done statically prior to executing the application, or the tags are aggregated dynamically during program execution. Static aggregation requires modifying the operating system and can needlessly load values that will not be referenced by the processor. Dynamic aggregation, on the other hand, only compresses entries after the processor has referenced them; therefore, extra time and energy is not wasted transferring data entries that will not be used.

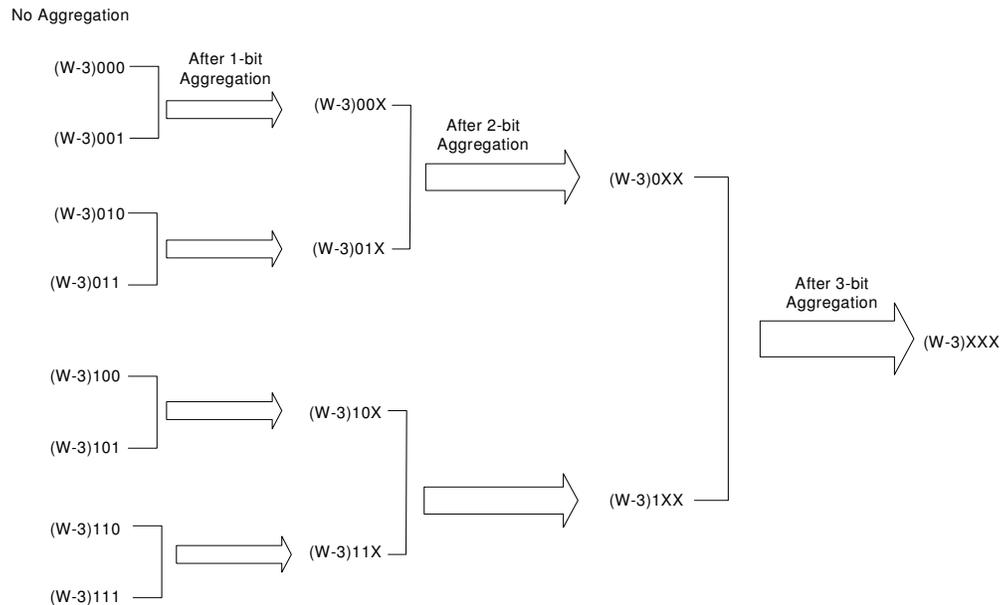


Figure 2: Sequential TCAM Based Tag Aggregation Process

Figure 2 gives an overview of the Dynamic Aggregation process carried out by the TCAM Enhanced Cache described in the next chapter. The figure shown here is based on aggregation of the three LSBs, which requires the use of three TCAM cells. In the figure, W represents the total number of tag bits, $(W-3)$ is the tag bits that are not considered for aggregation, and X represents a bit that has been compressed such that it can be interpreted as either a logic value of 0 or 1. In order to keep aggregation hardware overhead low, a sequential tag aggregation process has been adopted. With this 3-bit implementation, a maximum of eight entries can be merged into a single entry designated by three X values stored in the three least significant bits. For 3-bit aggregation, at most three rounds of sequential compression are performed depending on how many of the eight entries are in the tag array. Each of the values connected with a square bracket in the figure constitute an aggregation group. If both entries of an aggregation group are present in the tag array, then

the two entries will be removed and replaced with a single entry having the appropriate number of bits set as X . In order to avoid any noticeable compression latency, dynamic aggregation is only performed during a cache miss, so only one member of an aggregation group can be present in the tag array at a time otherwise the aggregated entry will be present.

CHAPTER III

TCAM ENHANCED CACHE ARCHITECTURE

TCAM cells are ideally suited for an architecture supporting dynamic tag aggregation. In addition to the standard 0 and 1 logic states, TCAM cells have a “don’t care” state, which is typically represented with an X . When a TCAM cell containing the don’t care value is searched, a match is returned regardless of whether a 0 or 1 is being requested. Based on this property, two tag entries can be aggregated by setting their least significant differing bits to be don’t care values.

The TCAM Enhanced Cache Architecture consists of a tag array that is augmented with a few TCAM cells in order to support dynamic aggregation. The enhanced cache structure can be divided into two major components – the CAM/TCAM Cache and the Dynamic Aggregator Module (DAM). Details of each component are given in the following subsections.

A. CAM/TCAM CACHE

In general, only the W most significant bits of a tag address value are stored in the cache’s tag array. TCAM cells are used to store the N least significant bits of the total W bits. The rest of the tag bits are stored using CAM cells. The extent of dynamic aggregation depends on the number of TCAM cells in a single tag array entry. More precisely, a tag entry can map up to a maximum of 2^N cache lines. All the blocks that are mapped by a single tag entry are referred to as a *super-block*, and its tag entry is referred to as a *super-block tag*. Therefore, the number of cache lines in a super-block might be 1, 2, 4, 8, 16.... 2^N depending on the amount of compression achieved for the tag entry. The

CAM/TCAM Cache maintains a single valid and dirty bit for the super-block. The number of cache lines fetched from main-memory on a cache miss is one; however, the number of cache lines that get replaced might be 1, 2, 4, 8, 16 ... 2^N depending on the super-block size.

Rather than sequentially storing all the cache lines belonging to a super-block, which would require extra overhead to manage the data array, a set of 2^N data banks is used with each storing a cache line for every tag array entry. Figure 3 illustrates the CAM/TCAM design with N , the number of TCAM cells, equal to three. The data bank selector is configured such that Data Bank 0 contains the cache line for a tag value ending in 000, and Data Bank 7 contains the cache line for a tag ending in 111. Based on this design, the appropriate data bank can be activated while the tag array is being searched so that when a matching tag is found only one row is selected from a single data bank.

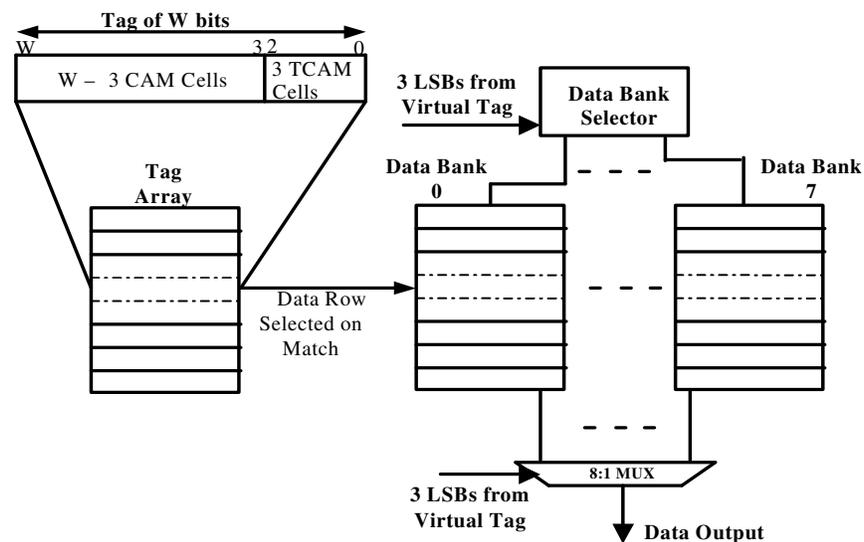


Figure 3: CAM/TCAM Cache with 3 TCAM Cells

Since new tags are only brought into the cache when a miss occurs, tag compression is only initiated when the cache does not contain the requested data. The steps involved in a CAM/TCAM Cache look up are as follows:

- 1) W most significant bits of the requested address are searched in the tag array.
- 2) Bits (N to 0) are passed as input to the Data Bank Selector and output Mux module.
- 3) If there is a match in the tag array, the match line goes high, thus selecting an entire row in the active Data Bank. The appropriate data value is then driven to the cache output.
- 4) If the requested address is not found in the tag array, the cache requests the cache line containing the data from main memory, and while the data is being fetched, the missed address tag is examined by the DAM for aggregation with entries already stored in the cache's tag array.

B. DYNAMIC AGGREGATOR MODULE

As previously stated, dynamic aggregation is only performed during a cache miss. On each miss, the tag address that missed is stored in a register, and the requested data is fetched from main memory. While waiting for the data to be retrieved, the DAM is able to use the stored tag address to search the tag array and perform tag aggregation if it is possible. As long as the dynamic aggregation process is completed before the data is returned from main memory, the compression latency will remain hidden. Figure 4 depicts the interconnections among the DAM and CAM/TCAM Cache hardware blocks. Although

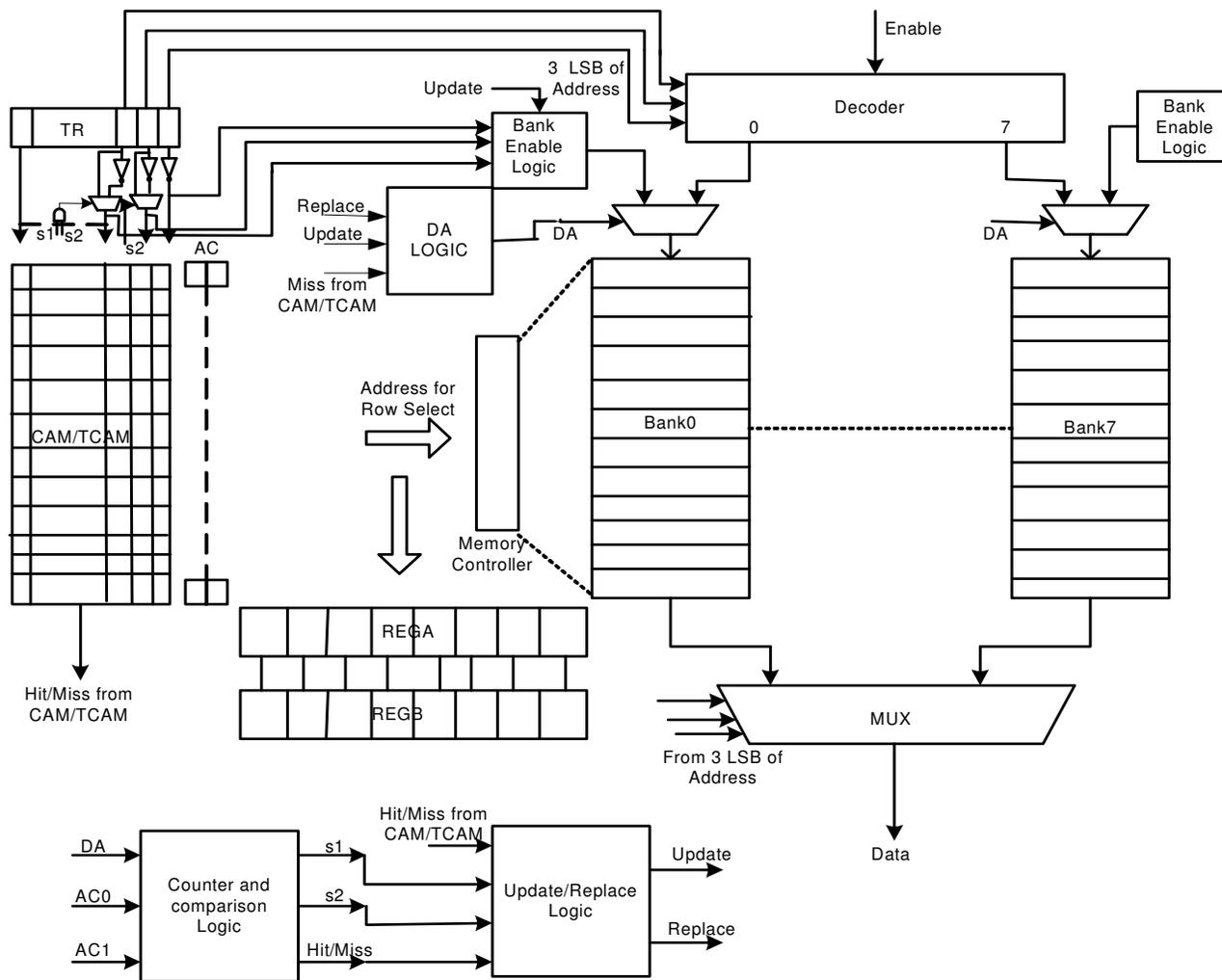


Figure 4: CAM/TCAM Cache with DAM Hardware

the DAM can be implemented to support N TCAM cells, it is only illustrated for 3-bit aggregation.

The DAM consists of seven distinct hardware units. The Temporary Register (TR) stores the tag address when a miss occurs. Each data bank has its own “Bank Enable Logic” unit and an associated buffer (not shown in figure) capable of storing a single data block entry. The “Bank Enable Logic” unit generates an enable signal for the corresponding bank during aggregation. The “Dynamic Aggregator (DA) Logic” unit generates the DA signal, which initiates the aggregation process when a miss occurs. Each tag array entry has an associated 2-bit Aggregation Counter (AC) unit that stores the current number of bits set as don’t care. During a tag array search, Register A (REGA) holds the current Row Address of the matching tag array entry and Register B (REGB) contains the Row Address value previously held by REGA. The “Counter and Comparison Logic” unit maintains state information regarding the aggregation process. The “Update/Replace Logic” unit generates Update and Replace signals for the Memory Management Unit (MMU) to assist storing the data retrieved from main memory as a result of a miss.

To perform 3-bit dynamic aggregation, variable bit length searches must be supported by the CAM/TCAM tag array. To enable searching of $(W-1)$, $(W-2)$, and $(W-3)$ bit lengths, additional hardware is added to the TR so that the Counter and Comparison Logic unit can control the values being searched for in the TCAM cells. For example, consider a miss that occurs for the tag address $(W-3)000$. Since this tag can aggregate only with $(W-3)001$, the LSB of the tag address stored in the TR is inverted, so the tag array is actually searched for $(W-3)001$. If a match occurs, 1-bit aggregation is possible, so the tag entry is updated to

($W-3$)00 X . Next, the tag array is searched for ($W-3$)01 X to see if 2-bit aggregation is possible. To search for ($W-3$)01 X , the LSB is again inverted, so the tag array is actually searched for ($W-3$)011. Thus a match might result if either ($W-3$)011 or ($W-3$)01 X is present in the tag array. To resolve the ambiguity of the matching entry, the Comparison Logic unit checks that the AC value of the matching entry is 01, which indicates that ($W-3$)01 X was found. Similarly for 3-bit aggregation, the tag array is searched for ($W-3$)111, which can match if either ($W-3$)111 or ($W-3$)1 XX is present in the tag array. Again, the Comparison Logic unit determines whether a hit occurs during aggregation based on the AC value of the matching entry. Since tag aggregation is initiated for every cache miss, it is only possible to have at most one entry match for each search of the tag array. For example, either one member of an aggregation group is present in the tag array or the aggregated entry is present. Therefore, the requirement of a complex priority encoder, which would have been needed to resolve multiple tag array matches, is eliminated.

On each cache miss, the “DA Logic” sets DA=1. This acts as an enable signal to all gated clocks within the aggregator module and starts the dynamic aggregation process. Next, the missed tag address is stored in the TR, and the 2-bit counter is incremented from 00 to 01. For Count=01, the CAM/TCAM array is searched for a ($W-1$) bit tag match. If it misses, dynamic aggregation is not possible, so the “Replace” signal is driven high, which instructs the MMU to replace the least recently used cache entry with the data retrieved from main memory. If a hit occurs, then 1-bit aggregation is possible. The Row Address of the matching tag entry is stored in a temporary register (REGA), and the data entry from the corresponding data bank is placed in that bank’s buffer.

After completing 1-bit aggregation, the counter is incremented to 10; the CAM/TCAM array is then searched for a $(W-2)$ bit tag match. If a miss occurs, no further aggregation is possible, so the “Update” signal is driven high. The “Update” signal causes the MMU to modify the last matched tag entry so that its LSB is set as X and $AC=01$. The data entry stored previously in a data bank buffer is loaded back into the data bank row corresponding to the tag entry that matched while performing 1-bit aggregation. If a hit occurs while searching for a $(W-2)$ bit tag match, the Row Address of the new matching tag entry is stored in REGA, and the data entries of the two corresponding data banks are stored in the banks’ buffers. The previous matched tag entry is then invalidated in the cache. Similar to 1-bit aggregation, only one match is possible when performing 2-bit aggregation. For example, consider a missed tag address value of $(W-3)000$. After 1-bit aggregation, the tag address is represented by $(W-3)00X$. When searching the tag array for a 2-bit aggregation candidate, only one of the following can be present in the tag array – either $(W-3)010$, $(W-3)011$, or $(W-3)01X$.

After completing 2-bit aggregation, the counter is incremented to 11, and the CAM/TCAM array is searched for a $(W-3)$ bit tag match. If a miss occurs, no more aggregation is possible. The Update signal is driven high, so the MMU will update the tag entry that matched when performing 2-bit aggregation. The MMU will set the tag entry’s two LSBs as X and set $AC=10$. The data bank buffer contents are stored back into the row of the data banks for the corresponding matched tag entry. If a hit occurs when searching for $(W-3)$ bits, then 3-bit aggregation is possible. The address of the matching tag entry is stored in REGA and the four corresponding data entries are stored in the data bank buffers.

At this point, there are seven data entries stored in the buffers associated with seven of the eight data banks; the empty buffer location will be filled once the missed data entry has returned from main memory. The matched tag entry from the 2-bit aggregation search is invalidated. Upon arrival of the missed data entry, Update is driven high thus forcing the MMU to update the matched tag entry with 3 don't care bits in its LSBs, the corresponding 2-bit AC is set to 11, and the data stored in the buffers is written back into the appropriate row of the data banks. This completes the aggregation process for the missed tag address value. Now the DA Logic unit sets DA=0 to deactivate the DAM until the next cache miss.

C. IMPACT ON CACHE ACCESS TIME

The TCAM Enhanced Cache provides access times similar to a conventional cache using a CAM based tag array with an equivalent number of tag entries. The access time of CAM and TCAM cells has been shown to be negligibly different [13]. Since the tag arrays contain the same number of entries, there is no extra latency for searching the CAM/TCAM tag array. While the tag array is being searched, the LSBs of the address tag are used to select the appropriate data bank containing the corresponding cache line; therefore there is no extra latency for bank selection. Since only a single data bank is activated at a time and each data bank is the same size as the conventional cache data array, there is no extra latency when retrieving data once a tag match has occurred. Based on these observations, it can be concluded that the compression scheme proposed in this paper does not increase the cache access time.

D. SET-ASSOCIATIVE IMPLEMENTATION

Set-associative caches partition the total cache space into distinct sets and decode the address bits to determine which set a desired data entry belongs. In traditional set-associative designs, the least significant bits indicate the byte offset, the next group of bits indicates the set, and the most significant bits constitute the tag value stored in the cache's tag array. Based on this approach, tag entries belonging to the same aggregation group will be located in different cache sets, and an entry that has one or more bits aggregated could conceivably be placed in any set mapped to by one of its uncompressed members. In order to simplify the aggregation process for a set-associative TCAM Enhanced Cache, the set mapping is altered so that all tag entries belonging to an aggregation group will map to the same set. Additionally, each set will have its own group of data banks in order to keep the data access time short. This modified set-associative design allows the sets to be treated as subbanks and search only a single set when accessing the cache or performing aggregation. By accessing a single set at a time, significant power savings can be achieved since only a fraction of the total CAM cells are searched on each access. Figure 5 illustrates the difference between the conventional address division and the modified set-associative mapping scheme.

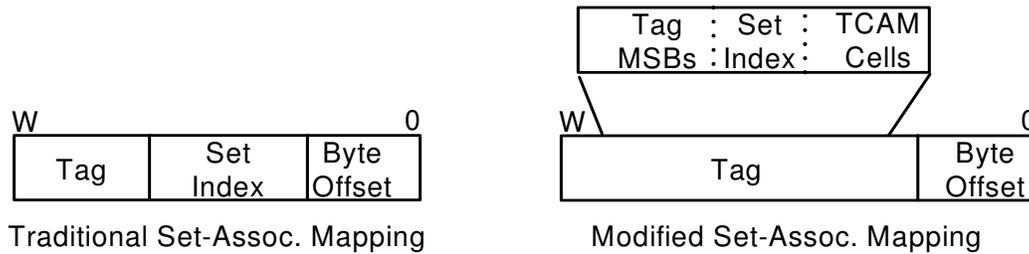


Figure 5: Set-Associative Mappings for an Address of W Bits

E. ESTIMATING ENERGY CONSUMPTION

The primary focus of modern processors is to maximize system performance while maintaining the energy consumption within a given budget. The energy consumed by a processor can be examined in two ways – the *on-chip energy consumption* and the *total energy consumption*. The on-chip energy consumption is the amount of energy consumed by transistors on the processor itself. Generally, the greater the on-chip energy consumption, the greater the heat dissipated by the processor. If too much heat is dissipated by the processor, a complex cooling scheme might be needed to keep the processor operating at acceptable temperatures. Intricate cooling schemes generally result in larger chips, which tend to make packaging more complex. The total energy consumption includes the on-chip energy consumption as well as the energy consumed by off-chip main memory accesses that occur in response to cache misses. For systems that have a limited power source, the total energy consumption gives an indication of an architecture's effect on battery life.

To evaluate the TCAM Enhanced Cache Architecture in terms of its energy efficiency, it is necessary to examine the impact it has on both the on-chip and total energy

consumption. For this analysis, it is only necessary to consider the energy consumed by accesses to the cache structures and main memory since the energy consumed by the rest of the processor would remain the same even if the cache structures were altered. The energy estimates are derived as follows:

$$E_{Aggregation} = N \times E_{Tag_search} + E_{DAM}$$

$$E_{On_chip} = Cache_{Accesses} \times (E_{Tag_search} + E_{Data_bank}) + E_{Aggregation} \times Cache_{Misses}$$

$$E_{Total} = E_{On_chip} + E_{Main_memory} \times Cache_{Misses}$$

E_{On_chip} and E_{Total} are the on-chip and total energy consumed. E_{Tag_search} is the energy consumed during a single search in the tag array, and E_{Data_bank} is the energy consumed when accessing the data from the selected row of the data bank. E_{DAM} is the worst-case energy consumption for aggregating all of the N TCAM cells in the CAM/TCAM Cache using the Dynamic Aggregator Module, so $E_{Aggregation}$ is the greatest amount of energy that could be consumed while performing tag aggregation in response to a cache miss. E_{Main_memory} is the energy consumed when accessing the main memory on each cache miss.

CHAPTER IV

EFFECTIVENESS OF DYNAMIC TAG AGGREGATION IN THE L2 DTLB

A. SIMULATION METHODOLOGY

To evaluate the effectiveness of the TCAM Enhanced Cache Architecture for high-performance processor TLBs, a subset of the SPEC CPU2000 benchmark programs [28] were executed using the SimpleScalar [29] simulator, which was extended to support 3-bit Dynamic Tag Aggregation in the L2 Data-TLB (Dynamic Aggregation DTLB).

1. Experimental Setup

The SimpleScalar simulator was configured to use the Alpha Instruction Set Architecture. In all of the tests, the processor configuration is exactly the same except for the L2 DTLB parameters, thus isolating the effects of the DTLB. Table 1 describes the three L2 DTLB configurations evaluated. These test configurations were chosen to evaluate how close the Dynamic Aggregation DTLB comes to its maximum and minimum performance levels. For example when all tag entries are fully compressed, the Dynamic Aggregation DTLB can at most perform as well as a conventional DTLB that can store the same amount of data. Similarly, the Dynamic Aggregation DTLB can at worst perform like the Equivalent Tag DTLB when no aggregation is possible. Based on CACTI [30] simulation results, the hit latency of a DTLB with 128 tag entries was determined to be 2 clock cycles and the hit latency of a DTLB with 1024 tag entries was found to be 3 cycles. Researchers working on cache compression have estimated that main memory latency for high-performance processors will grow to hundreds of clock cycles in the near future, and

they have used main memory access latencies as high as 400 clock cycles to judge the effectiveness of their scheme [23]. Based on this same line of reasoning, the miss latency of the L2 DTLB was chosen to be 400 clock cycles for all test configurations.

Table 1: L2 DTLB Test Configurations

Test Configuration	Parameters
Dynamic Aggregation (3-bit aggregation)	Tag Entries = 128 Total Data Entries = 1024 Hit Latency = 2 cycles Miss Latency = 400 cycles Replacement = LRU Page Size = 4KB
Equivalent Data Size	Tag Entries = 1024 Total Data Entries = 1024 Hit Latency = 3 cycles Miss Latency = 400 cycles Replacement = LRU Page Size = 4KB
Equivalent Tag Size	Tag Entries = 128 Total Data Entries = 128 Hit Latency = 2 cycles Miss Latency = 400 cycles Replacement = LRU Page Size = 4KB

2. Workload Selection

Ten benchmarks were chosen from the SPEC CPU2000 suite [28] to evaluate the proposed architecture – bzip, crafty, twolf, vpr, gcc, parser, and vortex are Integer applications while apsi, art, and facerec are Floating Point programs. These benchmarks were selected because they exhibited a high degree of spatial locality and a significant number of L2 DTLB misses [31]. For all the experiments, performance metrics were collected while executing the first 2 billion instructions.

B. RESULTS AND ANALYSIS

To gauge the performance of a L2 DTLB with 3-bit Dynamic Aggregation, its Miss Rate, Instructions Per Cycle (IPC), Energy Consumption, and Area are compared to those for a L2 DTLB with Equivalent Data and Tag Sizes.

1. Miss Rates

Since Tag Compression is intended to increase the effective L2 DTLB storage capacity, the Dynamic Aggregation DTLB is expected to have significantly fewer misses than the Equivalent Tag Size DTLB. Ideally, if the benchmarks exhibit a considerable amount of spatial locality, then all of the tag entries can be compressed to their maximum capacity resulting in the Dynamic Aggregation DTLB achieving miss rates comparable to the Equivalent Data Size DTLB.

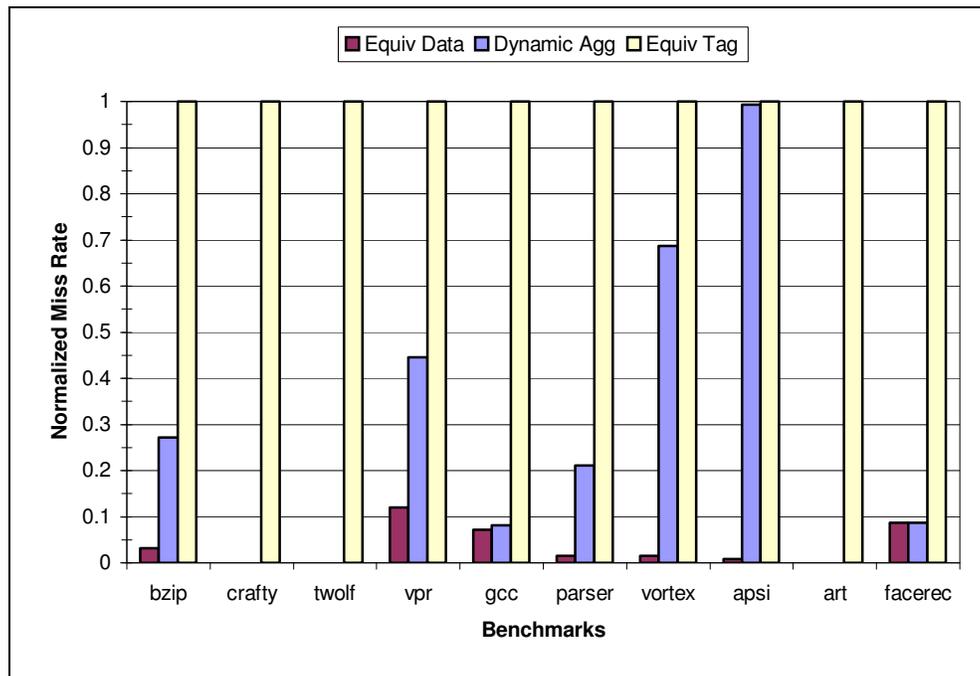


Figure 6: L2 DTLB Miss Rates Normalized to Equivalent Tag DTLB

Figure 6 presents the miss rates achieved during benchmark simulation. On average, the Dynamic Aggregation DTLB reduces the Equivalent Tag Size miss rate by 72%. Also, Dynamic Aggregation attains 76% of the Equivalent Data Size miss rate on average.

Clearly in *apsi* and *vortex*, there was not sufficient spatial locality in the L2 DTLB accesses to achieve enough compression to drastically reduce the miss rate. However, in *facerec*, *art*, *gcc*, *twolf*, and *crafty*, tag compression reaches nearly 100% thus decreasing the miss rates to those of the Equivalent Data Size DTLB.

2. System Performance

The goal of any architecture is to achieve the maximum possible performance. System Performance is typically measured in terms of Instructions Per Cycle (IPC). Since TLB misses take a significant amount of time to handle, miss rate reductions are usually reflected as an increase in overall system performance.

Figure 7 reports the IPC achieved during simulation. On average, the Dynamic Aggregation DTLB achieves 93% of the Equivalent Data IPC while the Equivalent Tag DTLB averages only 76% of the IPC. Dynamic Aggregation, therefore, provides approximately a 17% performance advantage over Equivalent Tag Sized DTLB with a 46% increase for *twolf* and a 32% increase for *facerec*.

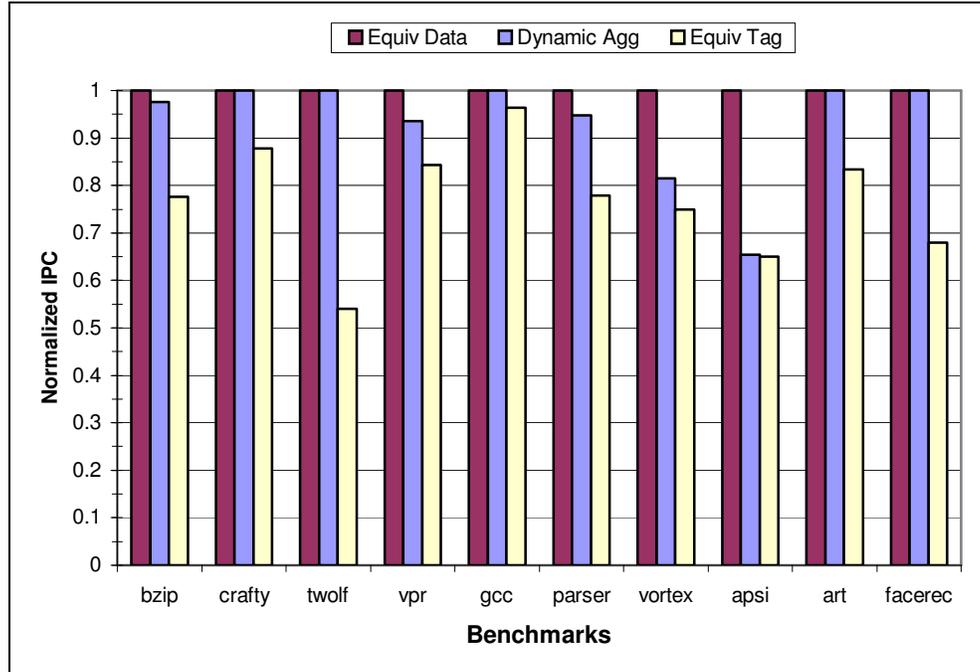


Figure 7: IPC Normalized to Equivalent Data DTLB

3. Energy Consumption

The on-chip and total energy consumptions were calculated based on the equations presented in Chapter III. E_{Tag_search} and E_{Data_bank} were determined using CACTI simulations [30] while E_{DAM} was determined using SPICE simulations [32]. E_{Main_memory} was calculated to be 18nJ per access based on a 300mW DRAM chip having a 60ns access time [33]. All the CACTI and SPICE simulations were performed based on 0.18 μ m technology.

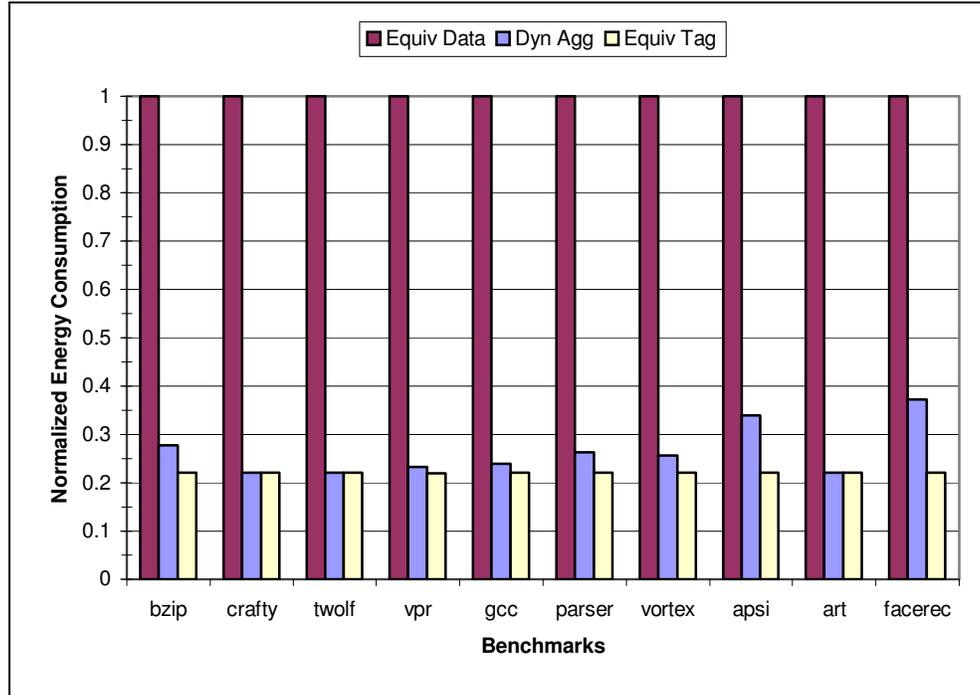


Figure 8: On-Chip Energy Consumption Normalized to Equivalent Data DTLB

Figure 8 shows that the Equivalent Data Sized DTLB consumes on average 74% more on-chip energy than either of the other two schemes while only providing an average performance gain of 7% over Dynamic Aggregation. The Dynamic Aggregation scheme provides a 17% performance increase over Equivalent Tag Sized DTLB while on average requiring only 4% more energy. Figure 9 shows that DRAM energy consumption has a significant impact on the total energy consumed, and in some cases, is able to outweigh the benefits of increased system performance. The Equivalent Data DTLB consumes on average 55% less energy than the Equivalent Tag DTLB; however, for vpr, the Equivalent Data DTLB consumes nearly 25% more total energy. The Dynamic Aggregation DTLB

saves an additional 10% over the Equivalent Data DTLB by consuming on average 65% less energy than the Equivalent Tag DTLB.

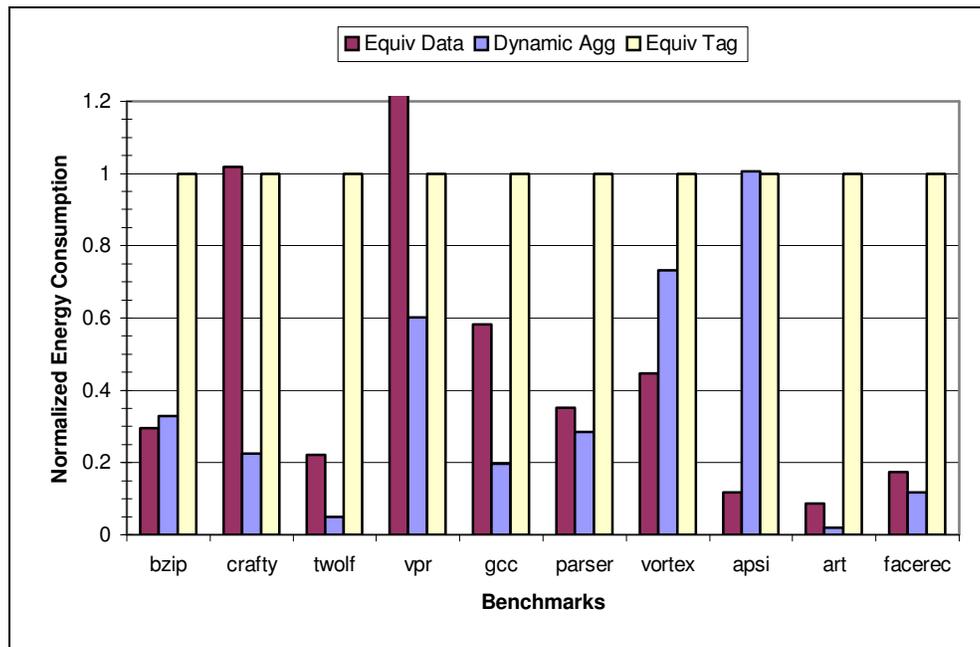


Figure 9: Total Energy Consumption Normalized to Equivalent Tag DTLB

4. Area Comparison

Since the TCAM Enhanced Cache Architecture uses compression to reduce the number of tag entries needed to access the data array, the architecture is meant to be an area efficient alternative to simply implementing a conventional TLB that can store the same amount of data. To determine the area efficiency, an area comparison was performed based on the number of transistors in the L2 DTLB and in the extra DAM hardware needed for 3-bit aggregation. For the area calculations, the following transistor relationships were used: CAM cell = 10 transistors [34], SRAM cell = 6 transistors, and TCAM cell = 17 transistors

[35]. As determined based on SPICE simulations [32], the DAM hardware needed to support 3-bit aggregation requires approximately 4616 transistors. Table 2 gives a breakdown of the number of transistors needed to implement each scheme. Based on these transistor counts, the Dynamic Aggregation TLB requires 85% less tag area than the Equivalent Data TLB, which translates to approximately 50% less total area. Also, the results clearly show that the DAM area overhead is quite small; it is approximately 1.5% of the total area used by a Dynamic Aggregation TLB storing 128 tag entries.

Table 2: Transistor Counts for L2 DTLB

L2 DTLB Configuration	Tag Array	Data Array	DAM
Dynamic Aggregation	43648	196608	4616
Equivalent Data Size	296960	196608	0

CHAPTER V

EFFECTIVENESS OF DYNAMIC TAG AGGREGATION IN THE L1 CACHES

A. SIMULATION METHODOLOGY

To evaluate the effectiveness of the TCAM Enhanced Cache Architecture for low-power embedded processor caches, a subset of the MiBench [36] embedded benchmark applications were executed using the SimpleScalar [29] simulator, which was extended to support 2-bit tag aggregation in the level 1 instruction and data caches.

1. Experimental Setup

The SimpleScalar was simulator configured to use the ARM Instruction Set Architecture. In all of the tests, the processor core configuration was based on Intel's XScale microarchitecture, and the only processor parameters that varied among the test configurations were the L1 instruction and data cache parameters thus isolating the effects of dynamic aggregation. Table 3 describes the cache configurations evaluated. These test configurations were chosen to evaluate how close a processor using Dynamic Aggregation caches for both the L1 instruction and L1 data caches comes to its maximum and minimum performance levels. For example when all tag entries are fully compressed, the Dynamic Aggregation Caches can at most perform as well as conventional set-associative caches that can store the same amount of data per set. Similarly, the Dynamic Aggregation Caches can at worst perform like the Equivalent Associativity Caches when no aggregation is possible.

Table 3: L1 Cache Test Configurations

Test Configuration	Parameters	
Dynamic Aggregation (2-bit aggregation)	Sets = 8 Line Size = 32 Hit Lat. = 1 cycle Repl. Policy = LRU	Assoc. = 16 Data Size = 16KB Miss Lat. = 32 cycles
Equivalent Data Size (has same number of data entries per set as the Dynamic Aggregation Cache)	Sets = 8 Line Size = 32 Hit Lat. = 1 cycle Repl. Policy = LRU	Assoc. = 64 Data Size = 16KB Miss Lat. = 32 cycles
Equivalent Associativity (has same number of tag entries per set as the Dynamic Aggregation Cache)	Sets = 8 Line Size = 32 Hit Lat. = 1 cycle Repl. Policy = LRU	Assoc. = 16 Data Size = 4KB Miss Lat. = 32 cycles

2. Workload Selection

Ten benchmarks were chosen from the MiBench suite [36] to evaluate the proposed architecture – dijkstra, ispell, mad, qsort, rijndael encode, tiffmedian, lame, gsm encode, ghostscript, and rsynth. These benchmarks were selected because they exhibited a high degree of spatial locality and a significant number of L1 cache misses. For all the experiments, performance metrics were collected while executing the applications’ large data sets up to the first 2 billion instructions.

B. RESULTS AND ANALYSIS

To gauge the performance of a processor using 2-bit Dynamic Aggregation Caches, its Miss Rates, Instructions Per Cycle (IPC), Energy Consumption, and Area are compared to a processor using conventional L1 caches having Equivalent Data capacity and a processor using conventional L1 caches having Equivalent Associativity.

1. Miss Rates

Since tag compression is intended to increase the effective storage capacity of a cache having the same associativity, the Dynamic Aggregation Caches are expected to have significantly fewer misses than the Equivalent Associativity Caches. Ideally, if the benchmarks exhibit a considerable amount of spatial locality, then all of the tag entries can be compressed to their maximum capacity resulting in the Dynamic Aggregation Caches achieving miss rates comparable to the Equivalent Data Size Caches.

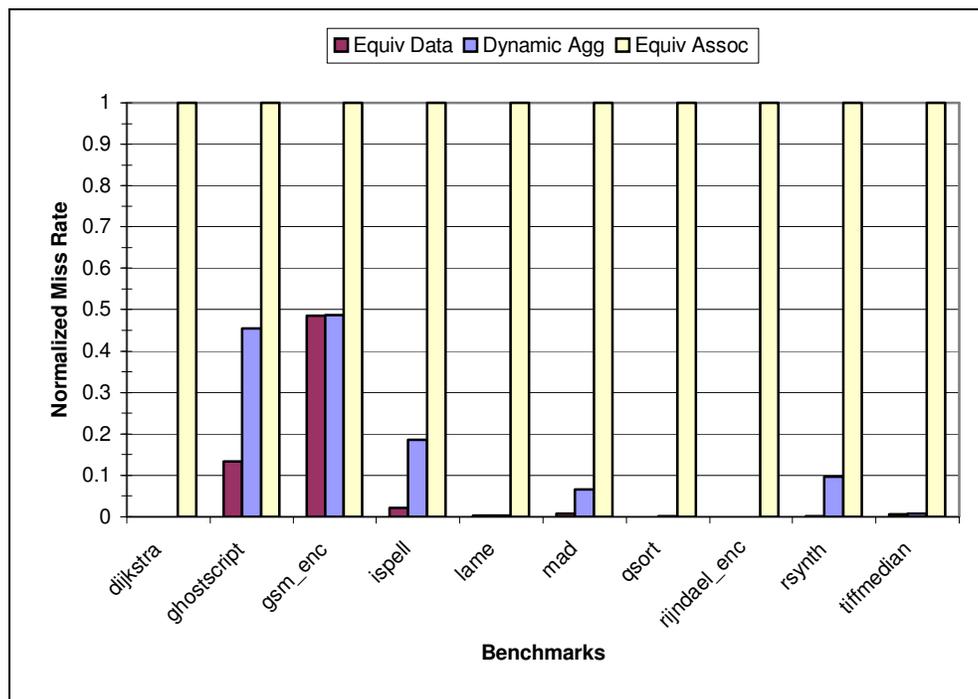


Figure 10: Instruction Cache Miss Rates Normalized to Equivalent Associativity Cache

Figure 10 and Figure 11 present the L1 instruction and data cache miss rates achieved during benchmark simulation. On average, the Dynamic Aggregation Cache reduces the Equivalent Associativity Cache instruction miss rate by 87% and the data miss rate by 54%.

Also, Dynamic Aggregation attains 92% of the average instruction miss rate and 79% of the average data miss rate achieved by the Equivalent Data Cache. Only ghostscript, ispell, and gsm_enc resulted in the Dynamic Aggregation Cache having a 10% or higher increase in miss rate than the Equivalent Data Cache. For these benchmarks, there was insufficient spatial locality in the L1 cache accesses to achieve enough compression to drastically reduce the miss rate. However, in dijkstra, gsm_enc, lame, qsort, rijndael_enc, and tiffmedian, tag compression reaches nearly 100% in the instruction cache thus decreasing the miss rates to those of the Equivalent Data Cache; this is also the case of the data cache miss rates for dijkstra, lame, qsort, and rijndael_enc.

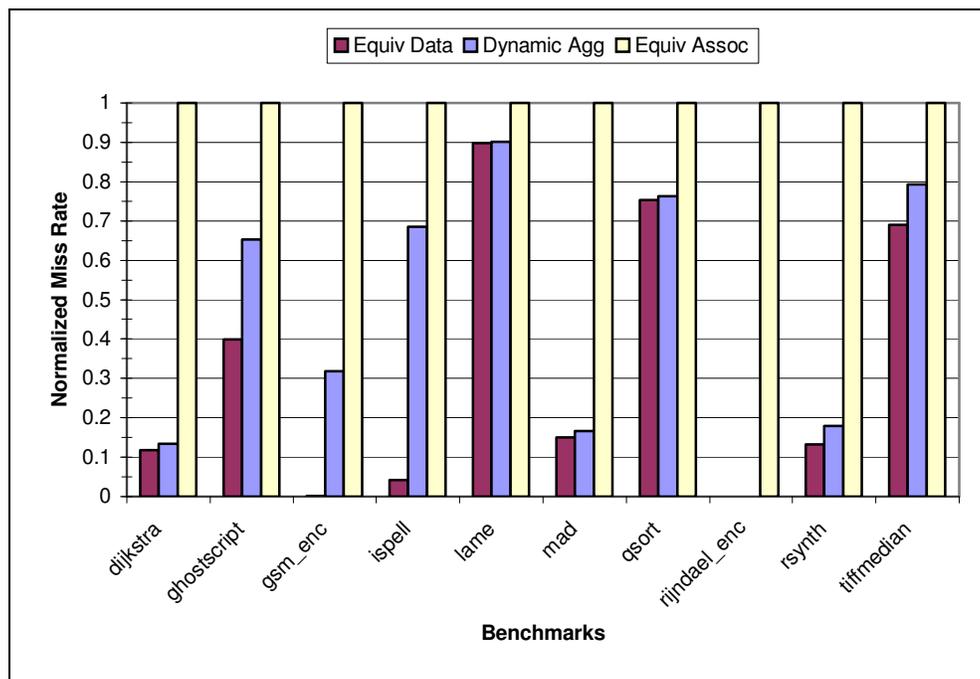


Figure 11: L1 Data Cache Miss Rates Normalized to Equivalent Associativity Cache

2. System Performance

The goal of any architecture is to achieve the maximum possible performance. System Performance is typically measured in terms of Instructions Per Cycle (IPC). Since main memory accesses take much longer than cache accesses, cache miss rate reductions are usually reflected as an increase in system performance. The Equivalent Data Cache is expected to have the lowest miss rates and therefore the highest IPC. Depending on the main memory access latency, the benchmarks that have significantly more misses for the Dynamic Aggregation Cache might still result in nearly the same system performance, so miss rate is not always a direct translation into system performance.

Figure 12 reports the IPC achieved during simulation. On average, the Dynamic Aggregation Cache achieves 99% of the Equivalent Data IPC while the Equivalent Associativity Cache averages only 80% of the IPC. Dynamic Aggregation, therefore, provides approximately a 19% performance advantage over Equivalent Associativity Caches with a 70% increase for rijndael_enc, a 36% increase for dijkstra, and a 38% increase for qsort.

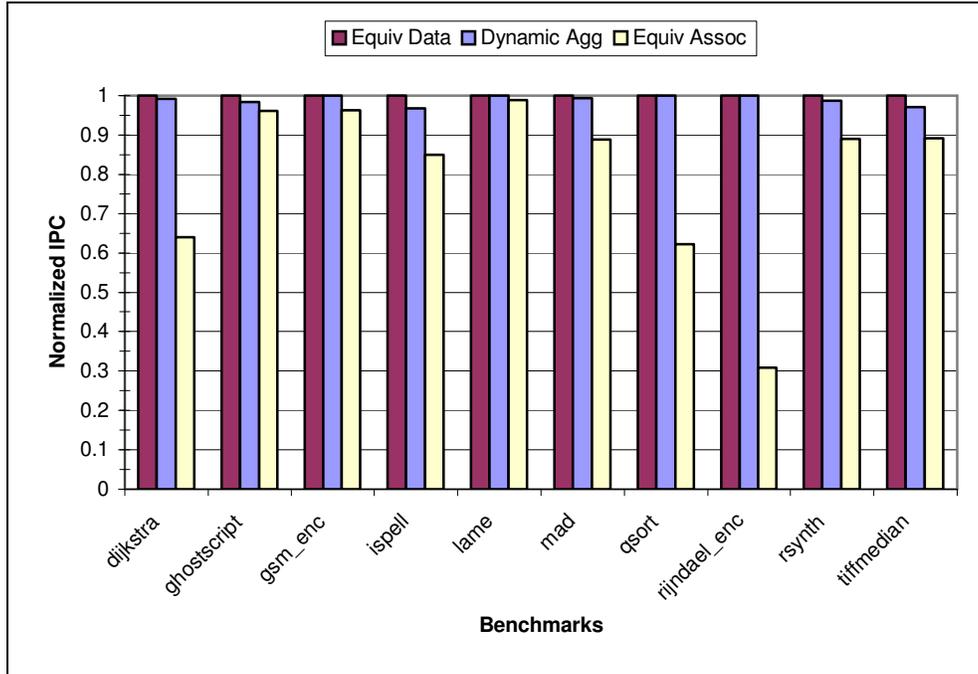


Figure 12: IPC Normalized to Equivalent Data Caches

3. Energy Consumption

The on-chip and total energy consumptions were calculated based on the equations presented in Chapter III. E_{Tag_search} and E_{Data_bank} were determined using CACTI simulations [30] while E_{DAM} was determined using SPICE simulations [32]. E_{Main_memory} was calculated to be 18nJ per access based on a 300mW DRAM chip having a 60ns access time [33]. All the CACTI and SPICE simulations were performed based on 0.18 μ m technology.

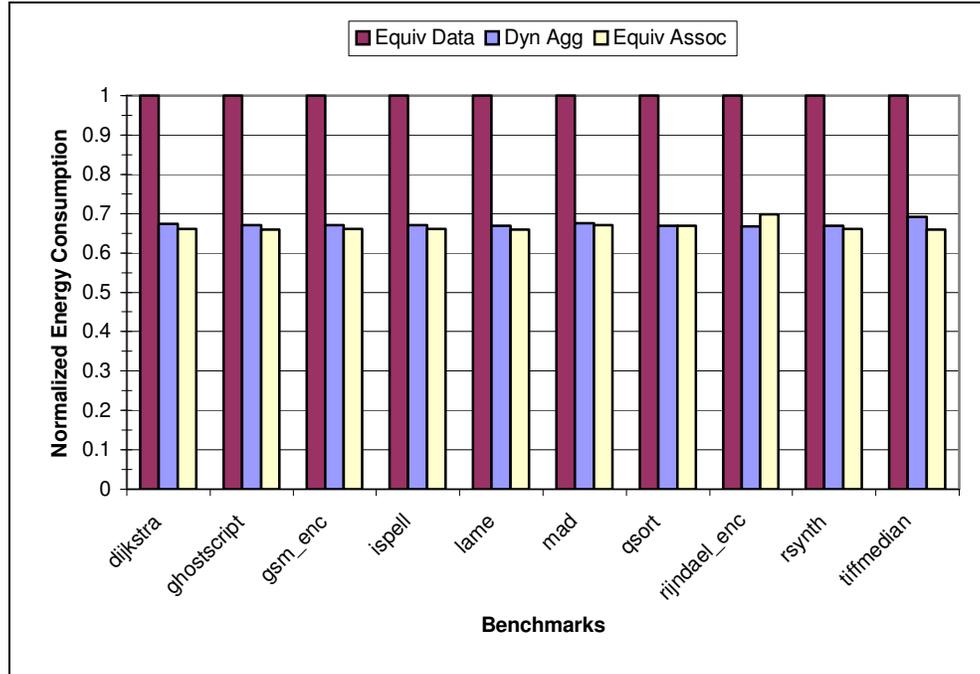


Figure 13: On-Chip Energy Consumption Normalized to Equivalent Data Caches

Figure 13 shows that the Equivalent Data Caches consume on average 33% more on-chip energy than either of the other two schemes while only providing an average performance gain of 1% over the Dynamic Aggregation Caches. The Dynamic Aggregation Caches provide a 19% performance increase over the Equivalent Associativity Caches while on average requiring only 1% more on-chip energy consumption. Figure 14 shows that DRAM energy consumption has a significant impact on the total energy consumed, and in some cases, is able to outweigh the benefits of increased performance. The Equivalent Data Cache consumes on average 10% less energy than the Equivalent Associativity Caches; however, for ghostscript, gsm_enc, and lame, the Equivalent Data Caches consume significantly more total energy. The Dynamic Aggregation Caches

consume 35% less total energy than the Equivalent Associativity Caches saving 25% more energy than the Equivalent Data Caches.

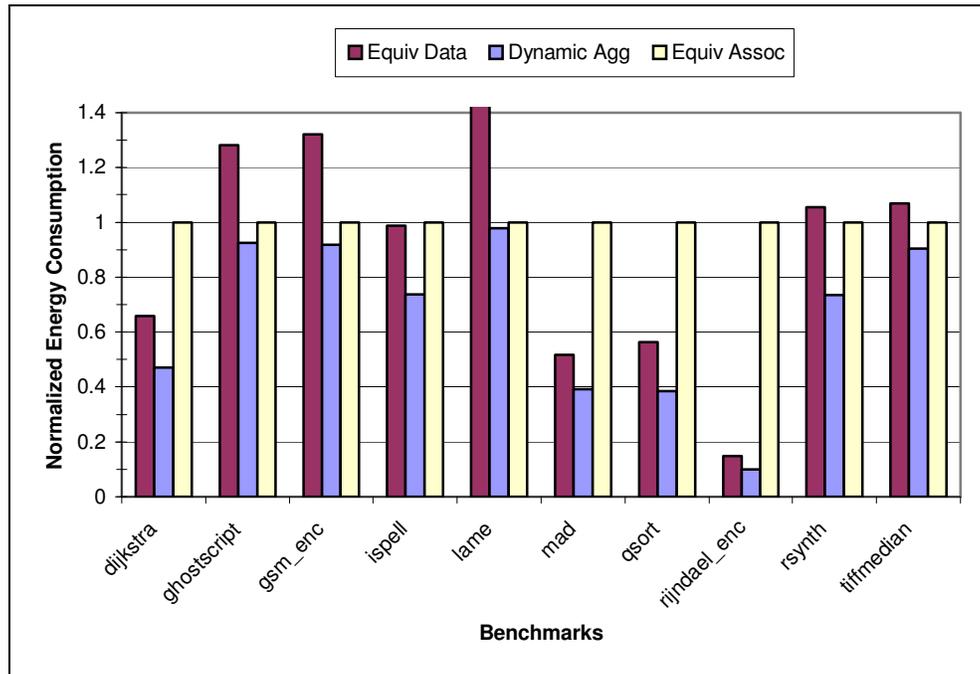


Figure 14: Total Energy Consumption Normalized to Equivalent Associativity Caches

4. Area Comparison

Since the TCAM Enhanced Cache Architecture uses compression to reduce the number of tag entries needed to access the data array, the architecture is meant to be an area efficient alternative to simply implementing a conventional cache that can store the same amount of data. To determine the area efficiency, an area comparison was performed based on the number of transistors in the L1 Cache and in the extra DAM hardware needed for 2-bit aggregation. For the area calculations, the following transistor relationships were used: CAM cell = 10 transistors [34], SRAM cell = 6 transistors, and TCAM cell = 17 transistors

[35]. As determined based on SPICE simulations [32], the DAM hardware needed to support 2-bit aggregation requires approximately 3362 transistors. Table 4 gives a breakdown of the number of transistors needed to implement each scheme. Based on these transistor counts, a 16 KB Dynamic Aggregation Cache requires 73% less tag area than an Equivalent Data Cache, which translates to approximately 10% less total area. Also, the results clearly show that the DAM area overhead is quite small; it is approximately 0.4% of the total area used by a 16 KB Dynamic Aggregation Cache.

Table 4: Transistor Counts for a 16 KB Cache

L1 Cache Configuration	Tag Array	Data Array	DAM
Dynamic Aggregation	32512	786432	3362
Equivalent Data Size	122880	786432	0

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

In this thesis, a TCAM Enhanced Cache Architecture is proposed as an energy and area efficient alternative to conventional cache designs that utilize CAM based tag arrays. Although 3-bit dynamic aggregation is described in this thesis, tag compression can be extended to N bits per tag entry depending upon the amount of spatial locality exhibited by the applications. Moreover, the proposed tag aggregation process does not affect the cache access time and there is no latency incurred for decompression, which is a major drawback of most other compression schemes. To evaluate the effectiveness of the proposed architecture's compression scheme, two case studies were performed – the L2 Data-TLB (DTLB) of a high-performance processor and the L1 instruction and data caches of a low-power embedded processor. These case studies were chosen to illustrate the wide scope of applications that could benefit from the proposed design.

The L2 DTLB case study showed that the performance of compute-intensive applications could benefit significantly from the proposed architecture. For the study, dynamic tag aggregation was performed only in the L2 DTLB. A maximum performance increase of 46% was achieved by the TCAM Enhanced Architecture as compared to the conventional Equivalent Tag Sized TLB, while consuming an average of 65% less energy. Although the Equivalent Data Sized TLB was able to achieve lower miss rates than the Dynamic Aggregation DTLB, the average performance improvement was only 7%; however, this additional performance costs an average of 74% more on-chip energy and 50% more area. The performance to energy and area consumption ratio clearly indicates

that the TCAM Enhanced Architecture is a more attractive solution to improving system performance rather than simply increasing the size of the TLB.

The L1 instruction and data cache case study illustrated that the performance of embedded applications could also benefit from the TCAM Enhanced Architecture. In this study, a 16 KB TCAM Enhanced Cache supporting Dynamic Tag Aggregation achieved 99% of the performance of caches that have Equivalent Data sizes, while consuming 33% less on-chip energy, 25% less total energy, and 10% less area. Clearly, caches that employ the Dynamic Aggregation technique proposed in this thesis are more power and area efficient than conventional caches having the same size data array.

The TCAM Enhanced Architecture described in this paper provides a basis for several future research directions. One such research direction is to investigate the benefit of incorporating a prefetch scheme to further reduce miss rates and improve system performance. The proposed architecture seems to be well suited for a sequential prefetching scheme that dynamically changes the cache line size thus varying the number of consecutive bytes retrieved in response to a cache miss. Another area of research is to extend the DAM to support a more general form of tag aggregation that can aggregate the low order tag bits in any order. This general form of tag aggregation would result in greater compaction of tag entries since compression could take place among any entries that only differ in their least significant bits. Also, alternative replacement schemes that take into account the tag aggregation degree of an entry could be evaluated to see if they result in better performance than the traditional Least Recently Used scheme currently employed by the TCAM Enhanced Architecture.

REFERENCES

- [1] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Third Edition, pp. 389-509, Morgan Kaufmann, San Francisco, CA, 2003.
- [2] International Technology Roadmap for Semiconductors, "ITRS 2003 Edition," 2004, <http://public.itrs.net/Files/2003ITRS/Home2003.htm>.
- [3] L. Clark, E. Hoffman, J. Miller, M. Biyani, L. Luyun, et al., "An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1599-1608, Nov. 2001.
- [4] J. Montanaro, R. Witek, K. Anne, A. Black, E. Cooper, et al., "A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, pp. 1703-1714, Nov. 1996.
- [5] S. Santhanam, A. Baum, D. Bertucci, M. Braganza, K. Broch, et al., "A Low-Cost, 300-MHz, RISC CPU with Attached Media Processor," *IEEE J. Solid-State Circuits*, vol. 33, no. 11, pp. 1829-1839, Nov. 1998.
- [6] M. Zhang and K. Asanovic, "Highly-Associative Caches for Low-Power Processors," in *Kool Chips Workshop, 33rd Int'l Symp. Microarchitecture*, 2000.
- [7] T. Anderson, H. Levy, B. Bershad, and E. Lazowska, "The Interaction of Architecture and Operating System Design," in *Proc. 4th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 108-119, 1991.
- [8] K. Bala, M. Kaashoek, and W. Weihl, "Software Prefetching and Caching for Translation Lookaside Buffers," in *Proc. 1st Symp. Operating Systems Design and Implementation*, pp. 243-253, 1994.

- [9] J. Chen, A. Borg, and N. Jouppi, "A Simulation Based Study of TLB Performance," in *Proc. 19th Int'l Symp. Computer Architecture*, pp. 114-123, 1992.
- [10] E. Witchel, S. Larsen, C. Aanian, and K. Asanovic, "Direct Addressed Caches for Reduced Power Consumption," in *Proc. 34th Int'l Symp. Microarchitecture*, pp. 124-133, 2001.
- [11] A. Kumar, "Increasing TLB Reach Using TCAM Cells," MS Thesis, Department of Computer Science, Texas A&M University, Fall 2004.
- [12] J. P. Wade and C. Sodini, "A Ternary Content Addressable Search Engine," *IEEE J. Solid-State Circuits*, vol. 24, no. 4, pp. 1003-1013, Aug. 1989.
- [13] I. Arsovski, T. Chandler, and A. Sheikholeslami, "A Ternary Content-Addressable Memory (TCAM) Based on 4T Static Storage and Including a Current-Race Sensing Scheme," *IEEE J. Solid-State Circuits*, vol. 38, no. 1, pp. 155-158, Jan. 2003.
- [14] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, Transparent Operating System Support for Superpages," in *Proc. 5th Symp. Operating Systems Design and Implementation*, pp. 89-104, 2002.
- [15] T. Romer, W. Ohlrich, A. Karlin, and B. Bershad, "Reducing TLB and Memory Overhead Using Online Superpage Promotion," in *Proc. 22nd Int'l Symp. Computer Architecture*, pp. 176-187, 1995.
- [16] M. Talluri and M. Hill, "Surpassing the TLB Performance of Superpages with Less Operating System Support," in *Proc. 6th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 171-182, 1994.

- [17] Z. Fang, L. Zhang, J. Carter, W. Hsieh, and S. McKee, "Reevaluating Online Superpage Promotion with Hardware Support," in *Proc. 7th Int'l Symp. High-Performance Computer Architecture*, pp. 63-72, 2001.
- [18] C. Park, J. Chung, B. Seong, Y. Roh, and D. Park, "Boosting Superpage Utilization with the Shadow Memory and the Partial-Subblock TLB," in *Proc. 14th Int'l Conf. Supercomputing*, pp. 187-195, 2000.
- [19] M. Swanson, L. Stoller, and J. Carter, "Increasing TLB Reach Using Superpages Backed by Shadow Memory," in *Proc. 25th Int'l Symp. Computer Architecture*, pp. 204-213, 1998.
- [20] J. Park and G. Ahn, "A Software-controlled Prefetching Mechanism for Software-managed TLBs," *Microprocessing and Microprogramming*, vol. 41, no. 2, pp. 121-136, May 1995.
- [21] A. Saulsbury, F. Dahlgren, and P. Stenstrom, "Recency-Based TLB Preloading," in *Proc. 27th Int'l Symp. Computer Architecture*, pp. 117-127, 2000.
- [22] G. Kandiraju and A. Sivasubramaniam, "Going the Distance for TLB Prefetching: An Application Driven Study," in *Proc. 29th Int'l Symp. Computer Architecture*, pp. 195-206, 2001.
- [23] A. R. Alameldeen and D. A. Wood, "Adaptive Cache Compression for High-Performance Processors," in *Proc. 31st Int'l Symp. Computer Architecture*, pp. 212-223, 2004.
- [24] J. Yang, Y. Zhang, and R. Gupta, "Frequent Value Compression in Data Caches," in *Proc. 33rd Int'l Symp. Microarchitecture*, pp. 258-265, 2000.

- [25] J. Lee, W. Hong, and S. Kim, "Adaptive Methods to Minimize Decompression Overhead for Compressed On-chip Cache," *Int'l J. Computers and Applications*, vol. 25, no. 2, pp. 98-105, Jan. 2003.
- [26] P. Petrov, A. Orailoglu, "Tag Compression for Low-Power in Dynamically Customizable Embedded Processors," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1031-1047, July 2004.
- [27] H. Wang, T. Sun, and Q. Yang, "CAT – Caching Address Tags: A Technique for Reducing Area Cost of On-Chip Caches," in *Proc. 22nd Int'l Symp. Computer Architecture*, pp. 381-390, 1995.
- [28] Standard Performance Evaluation Corporation, "CPU2000 Benchmarks," 2004, <http://www.spec.org/benchmarks.html>.
- [29] D. Burger and T. Austin, "The SimpleScalar Toolset, Version 2.0," Technical Report #1342, Computer Sciences Dept., Univ. of Wisconsin-Madison, June 1997.
- [30] P. Shivakumar and N. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Power Model," DEC WRL Research 2001/2, Aug. 2001.
- [31] G. Kandiraju and A. Sivasubramaniam, "Characterizing the d -TLB Behavior of SPEC CPU2000 Benchmarks," in *Proc. of the 2002 ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems*, pp. 129-139, 2002.
- [32] T. Quarles, A. Newton, D. Pederson, and A. Sangiovanni-Vincentelli, "SPICE 3 Version 3F5 User's Manual," Department of EECS, University of California, Berkeley, Mar. 1994.

- [33] N. Kirubanandan, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "Memory Energy Characterization and Optimization for the SPEC2000 Benchmarks," in *Proc. 4th IEEE Workshop on Workload Characterization*, pp. 193-201, 2001.
- [34] A. Efthymiou and J. Garside, "An Adaptive Serial-Parallel CAM Architecture for Low-Power Cache Blocks," in *Proc. of the 2002 Int'l Symp. Low Power Electronics and Design*, pp. 136-141, 2002.
- [35] B. Gamache, Z. Pfeffer, and S. Khatri, "A Fast Ternary CAM Design for IP Networking Applications," in *Proc. 12th Int'l Conf. Computer Communications and Networks*, pp. 434-439, 2003.
- [36] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *Proc. 4th IEEE Workshop on Workload Characterization*, pp. 3-14, 2001.

VITA

Jason Mathew Surprise was born on December 6, 1979 in Little Rock, Arkansas. In December of 2002, he received his Bachelor of Science degree in Computer Engineering from Texas A&M University, and he received his Master of Science degree in Computer Science from Texas A&M University in the spring of 2005. His research was focused on cache architectures that use compression to improve energy efficiency. His main interests are in computer architecture and real-time embedded systems. He can be contacted through email at jason.surprise@gmail.com. He can also be reached by mail at 280 West Renner Road #4423, Richardson, Texas 75080.

The typist for this thesis was Jason Mathew Surprise.