INFORMATION PATTERN AWARE DESIGN STRATEGIES FOR NANOMETER-SCALE ADDRESS BUSES

by

JIANGJIANG LIU
August 9, 2004

A dissertation submitted to the
Faculty of the Graduate School of
University at Buffalo, The State University of New York
in partial fulfillment of the requirements for the
degree of

Doctor of Philosophy

Department of Computer Science & Engineering

ACKNOWLEDGEMENTS

I express my heartfelt gratitude to my advisor Dr. Nihar Mahapatra for his invaluable advice and support throughout this research. His graduate course on computer architecture piqued my interest in the area. His thoughtful criticism and inspiring guidance not only helped shape this work, it also benefited me immensely in terms of both intellectual and personal growth during my doctoral studies.

Many thanks go to my other thesis committee members, Dr. Chunming Qiao and Dr. Shambhu Upadhyaya, for their valuable time, helpful advice, and feedback on my research work, and encouragement on numerous occasions. My thanks also go to Dr. Hongjiang Song for serving as the outside reader and for his invaluable comments on my thesis. I also thank Dr. Xin He for his help and support during my graduate studies.

I express my appreciation to the *Computer Science and Engineering* department staff, especially, Jodi Reiner, Joann Glinski, Matthew Stock, William Wallace, Margaret Evans, and Lynne Terrana for being always there to patiently help me over the years.

Acknowledgement is due to: the *National Science Foundation*, which partly funded this research; the *Center for Computational Research* at the *University at Buffalo* for access to their high-performance computers on which many of the simulations for this research were performed; *Sun Microsystems, Inc.* and *SimpleScalar LLC* for providing free use of their *Shade* and *SimpleScalar* simulators, respectively; Sergey Lyubskiy for helping me with the use of *Shade*; and Niki Thornock of *Brigham Young University* for helping me with the use of the *BYU traces*.

I sincerely acknowledge help, both big and small, received from my research group mate and good friend, Krishnan Sundaresan. Research is more fun with his help and support. The innumerable discussions we had were greatly stimulating and enabled deeper and faster progress. Also, I thank him for proof reading a good portion of my dissertation.

I thank all former and current members of the Computer Architecture Lab for helping me over the past five years. Special thanks go to Shanker Nagesh and Srinivas Dangeti. I also thank all my friends who made my stay at Buffalo pleasant and memorable. Most of all, I wish to thank my best friend Jian Chen for always being there.

Lastly and most importantly, I thank my parents and my brother for their love and constant support in my educational endeavors, and, in the most special manner, my husband for his unwavering support and understanding which made my efforts easier.

To Daoying

ABSTRACT

The growing disparity in processor and memory performance has forced designers to allocate an increasing fraction of die area to *communication* (I/O buffers, pads, pins, on- and off-chip buses) and *storage* (registers, caches, main memory) *components* of the memory system to enable low-latency and high-bandwidth access to large amounts of *information* (addresses, instructions, and data). Consequently, the memory system has become critical to system performance, power consumption, and cost.

In this dissertation, we consider three types of redundancies related to information communicated and stored in the memory system, with the main focus being on information communicated on nanometer-scale address buses. They are *temporal redundancy*, *information redundancy*, and *energy redundancy*. To take advantage of these redundancies, we analyze and design *information pattern aware* strategies to exploit various patterns in information communicated and stored in a multi-level memory hierarchy to derive gains in performance, energy efficiency, and cost. Our main contributions are as follows. (1) A comprehensive limit study on the benefits of address, instruction, and data compression at all levels of the memory system considering a wide variety of factors. (2) A technique called *hardware-only compression (HOC)*, in which narrow bus widths are used for underutilized buses to reduce cost, novel encoding schemes are employed to reduce power consumption, and concatenation and other methods are applied to mitigate performance penalty. (3) A detailed analysis of the performance, energy, and cost trade-offs possible with two cache-based dynamic address compression schemes. (4) A highly energy- and performance-efficient dynamic ad-

dress compression methodology for nanometer-scale address buses. Many of the principles underlying this methodology are also applicable to instruction and data bus compression.

All our analysis and design has been performed in the context of real-world benchmark suites such as SPEC CPU2000 and using execution-driven simulators like Shade and SimpleScalar. Our analysis shows that ample opportunities exist for applying compression throughout the memory system. Further, we show that our address compression methods can simultaneously provide significant improvements in energy efficiency, cost, and latency compared to an uncompressed bus.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Performance, power consumption, and cost are arguably the three most important parameters that drive computer system design today, although their relative importance varies across systems. Thus, while performance is most important in high-end multiprocessors and servers, performance/cost drives the desktop market, and cost and power play a more significant role in embedded and wireless systems.

All computer systems have three main subsystems: the *computation system* or the processor core, the *memory system*, and the *I/O system*. The memory system has two main types of components: *storage components* (including registers, one or more levels of caches, main memory) for storing information (primarily instructions and data) and *communication components* (comprising I/O buffers, I/O pads, and pins on the processor and memory chips, and on- and off-chip control, address, instruction, and data buses) for communicating in-

formation (primarily addresses, instructions, and data) between the computation system and storage components and between the storage components themselves.

A combination of dramatic technological and architectural advancements has resulted in an exponential trend for computation system performance enhancement. This, coupled with slower speed improvements of on- and off-chip interconnect, caches, and DRAM, has contributed to a growing computation-memory system performance gap [26]. To address this problem, the fraction of the processor chip devoted to storage (registers, caches) and communication components (I/O buffers and pads, on-chip buses) has increased and so also has the number and size of off-chip storage (off-chip caches, main memory) and communication (pins, off-chip buses) components [26]. Moreover, in nanometer regime (drawn feature sizes < 100 nm), interconnect size scales relatively poorly compared to logic size, and not only do individual wire capacitances contribute to power consumption, but more so do interwire capacitances between adjacent on-chip bus lines due to tighter spacing between lines [58]. Consequently, increasingly more fraction of the system power consumption and cost is due to the memory system compared to the computation system [69]. Thus, the memory system is becoming an increasing bottleneck as designers strive towards higher performance, cost-effective, and power-efficient system designs.

1.2 Our Overall Approach and Contributions

In this dissertation, we consider the following three types of redundancies related to information communicated and stored in the memory system, with the main focus being on

information communicated on nanometer-scale address buses. *Temporal redundancy* refers to the fact that there are time periods when memory system components carry no or non-performance-critical information (e.g., idle buses and invalid, stale, or "dead" blocks in caches). *Information redundancy* means redundancy in the number of bits used to represent information, which causes more resources (e.g., bus lines or memory cells) to be engaged than necessary—information compression techniques address this. Finally, *energy redundancy* implies expending more than the required energy to communicate or store information; encoding schemes attempt to minimize this redundancy via energy-efficient information representations.

To take advantage of these redundancies, we analyze and design *information pattern aware* strategies to exploit various patterns in information communicated and stored in a multi-level memory hierarchy to derive gains in performance, energy efficiency, and cost. Our general approach consists of two phases. First, we analyze trace information off-line to determine information patterns and instances prevalent in different parts of the memory system. In light of this discovery, we next design hardware that, during run-time, statically exploits these frequent information instances and/or dynamically exploits information instances in accordance with predetermined frequent patterns. In our analysis, we consider address, instruction, and data information, all types of communication (on- and off-chip buses and associated circuitry) and storage (registers, caches, main memory, TLB, and page table) components at different memory levels, and various target system (embedded, desktop, server) and application (e.g., DSP, multimedia, integer-intensive, scientific) scenarios.

Our main contributions are as follows. (1) A comprehensive limit study on the benefits of address, instruction, and data compression at all levels of the memory system considering a wide variety of factors. (2) A technique called *hardware-only compression (HOC)*, in which narrow bus widths are used for underutilized buses to reduce cost, novel encoding schemes are employed to reduce power consumption, and concatenation and other methods are applied to mitigate performance penalty. (3) A detailed analysis of the performance, energy, and cost trade-offs possible with two cache-based dynamic address compression schemes. (4) A highly energy- and performance-efficient dynamic address compression methodology for nanometer-scale address buses. Many of the principles underlying this methodology are also applicable to instruction and data bus compression.

All our analysis and design has been performed in the context of real-world benchmark suites such as SPEC CPU2000 and using execution-driven simulators like Shade and SimpleScalar. Our analysis shows that ample opportunities exist for applying compression throughout the memory system. Further, we show that our address compression methods can simultaneously provide significant improvements in energy efficiency, cost, and latency compared to an uncompressed bus.

1.3 Dissertation Outline

The remainder of the dissertation is organized as follows. In the next chapter, we comprehensively analyze the redundancy in the information (addresses, instructions, and data) stored and exchanged between the processor and the memory system and evaluate the poten-

tial of compression in improving performance, power consumption, and cost of the memory system. We then present our work on nanometer-scale address bus compression to improve cost, power consumption, and performance by exploiting temporal, information, and energy redundancies in the information carried. In Chapter 3, we describe a technique for exploiting temporal and energy redundancies in buses called *hardware-only compression (HOC)*, in which narrow bus widths are used for underutilized buses to reduce cost. To minimize the power overhead of HOC, we propose various techniques to ensure energy-efficient transmission of uncompressed information on narrow buses. In addition, we apply different degrees of wire spacing by taking advantage of the extra area available from HOC to reduce coupling capacitance and thereby reduce wire delay and on-chip energy.

In the next two chapters, we describe how we exploit information and energy redundancies of information transmitted on memory system buses for performance, power, and cost improvements. Dynamic address compression schemes that exploit address locality can help reduce both address bus energy and cost simultaneously with only a small performance penalty. In Chapter 4, we investigate two such schemes and determine their optimal parameters that result in the highest area/cost reductions and least performance penalty for various address buses (both on- and off-chip) in current systems. We present results on how address compression schemes perform when applied to on-chip or off-chip buses in modern superscalar processors. In particular, we explore the performance, energy, and cost benefits of address compression, the effect of techniques like bus pipelining, and the effect of technology scaling on energy-efficiency of compressed address buses. Next, in Chapter 5, we present various

techniques that can be used with existing compression schemes for buses to ensure high energy-efficiency for compressed information transmission and propose a highly energy- and performance-efficient dynamic address compression methodology for nanometer-scale address buses, partial-match compression. To improve the hit-rate and reduce miss penalty of the compression cache used in the previous schemes, we propose *partial-matching* of the tag portion stored in the compression cache with the higher order portion of the address and present performance and energy optimized designs. Finally, Chapter 6 concludes this dissertation and also discusses future research.

Chapter 2

A Limit Study on the Benefits of Memory System Compression

A *compressed memory system* (CMS) architecture is a computer system architecture that employs compression in one or more parts of the memory system. In this chapter, we consider the advantages of CMS architectures in terms of improvements that can be obtained in performance (improvements in bandwidth and latency of communication components and improvement in capacity of storage components), power consumption, and cost. We consider all the three primary types of information, namely, addresses, instructions, and data, and all important storage and communication components at all levels of the memory system hierarchy, where such information is stored or communicated. For addresses, we consider the tag fields of instruction and data caches and instruction and data address buses. For instructions, we consider the data fields of instruction caches, main memory executable code, and

instruction buses. For data, we consider integer and floating-point register files, data fields of data caches, and data buses. For our study, we consider a memory hierarchy with split instruction and data caches at the first level, a unified cache at the second level, and a main memory. We consider both demultiplexed and multiplexed buses.

The chapter is organized as follows. Section 2.1 discusses the advantages of a CMS architecture and its feasibility. Then we review related work in Section 2.2 and discuss the relationship of our work to previous research in Section 2.3. Sections 2.4, 2.5, and 2.6 describe the analysis methods and simulation environment used in our study. Sections 2.7 - 2.18 present the results of our analysis. Finally, we conclude in Section 2.19.

2.1 The Case for Compressed Memory System Architectures

Here we first briefly explain where opportunities for compression lie in the memory system in Sec. 2.1.1. Then, in Sec. 2.1.2, we explain the benefits to be gained by applying compression. Next, in Sec. 2.1.3, we discuss the feasibility of applying compression and some challenges to be overcome. Finally, we briefly describe a useful way to classify compressed memory system architectures.

2.1.1 Opportunities for compression

Compression of some source information consisting of a sequence of symbols is possible when those symbols occur with non-uniform frequencies or likelihoods either in the source

as a whole or in any given portion thereof. This allows for the encoding of the more frequent or likely symbols with shorter code words compared to the less frequent or likely symbols, resulting in an overall compression of the source. The three primary types of information that are stored and communicated by the storage and communication components of the memory system, respectively, are addresses, instructions, and data. All three of these inherently possess significant amounts of redundancy as we explain next.

Address redundancy

Addresses are of two types: instruction addresses and data addresses. Both exhibit spatial and temporal locality, meaning that the next instruction or data address to be issued by the processor is not random, but likely spatially and/or temporally close to recently issued addresses. Instruction addresses issued by the processor to the L1 cache are typically sequential, except when branches or jumps occur, and even when this happens, the target addresses are not typically very far away from the last address. That is the reason why many instruction sets provide branch and jump instructions that specify the target address relative to the previous address. The addresses issued by L1 cache to L2 cache correspond to misses in the former and are more unpredictable compared to those issued by the processor to L1. Similarly, addresses issued by higher levels (away from the processor) of the memory system become increasingly unpredictable and hence more information rich. Still, these addresses do exhibit temporal and spatial locality, although to lesser extents. Data addresses issued by the processor are also known to exhibit temporal and spatial locality because of scanning of data arrays in loops, although to a lesser extent than instruction addresses. Like instruction

addresses, redundancies are expected to decrease at higher levels of the memory hierarchy.

As far as storage components are concerned, address information is primarily stored in the tag fields of caches, translation-lookaside buffer (TLB), and page tables (and some registers, such as the program counter and the memory address register, but this is not much). Since tag fields store a portion of the address (a portion of the instruction address in the case of instruction caches and a portion of the data address in the case of data caches), they are expected to exhibit redundancy as discussed above for addresses. Specifically, the tag fields correspond to blocks that have been recently accessed and as such they should be temporally and spatially close. Note that since the tag field is normally derived from the high order portion of the address, it is expected to possess a higher amount of redundancy than whole addresses, since the high order end of the address is where more redundancy lies due to the spatial proximity of addresses issued. Similarly, the TLB and page tables which store address information (virtual and physical page numbers) will have redundancies.

Instruction redundancy

Since instructions fetched correspond to instruction addresses issued by the processor, instructions exhibit the same temporal and spatial locality as instruction addresses. Further, not all instructions, instruction sequences, opcodes, register operands, and immediate constants are present equally frequently. Repetitions of instruction sequences, opcodes, registers, and immediate constants, and correlation between opcodes and registers and between opcodes and immediate constants can be exploited. The reasons for the presence of such redundancies are that all programs have certain basic characteristics, e.g., they have procedures and

procedure calls, they have branches every few instructions (typically every six instructions), they use loops and if-then-else clauses, etc. Moreover, compilers used to generate object code do so based on a set of templates, which naturally leads to redundancies. As discussed for addresses earlier, instruction traffic at higher levels of the memory hierarchy are likely to exhibit less temporal and spatial locality. However, since at higher levels, the instruction traffic consists of larger blocks, more redundancy is present within blocks. Similarly, in storage components, there is redundancy in the instructions stored in main memory and instruction caches.

Data redundancy

Data fetched by the processor also exhibits temporal and spatial locality, although to a lesser extent than instructions. However, there is extra redundancy present in the values of data communicated by data buses and stored in registers, data caches, and main memory. For any given type of data (character, integer, floating-point, etc.), not all values are equally likely. For instance, many programs do not tend to use the entire range of integer values possible, but rather the values used tend to be concentrated around certain values, especially, zero. For such small magnitude two's complement numbers, most high order bits of the data word are likely to be either all zero (positive) or all one (negative) due to sign extension.

2.1.2 Benefits of compression

Depending upon the state of the technology at the time of implementation and application requirements, it may not be possible to use compression to advantage in all areas of the

processor system, although substantial direct or indirect improvements can be expected in most areas of the system. As an example, using compression in on-chip or off-chip buses can have multiple ramifications. The effective bandwidth of the system will increase as more number of bits can be transmitted using the same number of bus lines. If the emphasis is on reducing power, it may be possible to reduce the number of bus lines while maintaining the same effective bandwidth, and this would result in power savings because fewer bits need to be transmitted and because significant amount of power is consumed in the metal lines of the chip. Similarly, a decrease in the number of bus lines will reduce the die area and hence cost could go down significantly because cost varies as the fourth or higher power of die area. Application of compression in other areas like caches, registers, and main memory have obvious benefits like increasing the effective storage capacity using the same number of transistors or lowering power consumption and cost by using smaller number of transistors that provide the same effective storage capacity.

Compression can also be used possibly to improve cache latency by, for example, storing a portion of the information in cache in compressed form. Using the same number of transistors, this modified cache will have more effective capacity and hence less effective miss rate than a regular fully uncompressed cache. The latency of the uncompressed portion of this modified cache will be comparable or better (due to its smaller size) relative to the regular cache. Also, the miss rate of the former will be only slightly worse than the latter for larger cache sizes. This is because, for larger caches, miss rate reduces very slowly as cache size increases. The latency of the compressed portion of the cache will be more than the regular

cache, but it will be less than that of the next higher level of the memory hierarchy. As a result, if there is a miss in the uncompressed portion of the cache, the compressed portion can be checked and if the required information is present, a slower access to the next higher level of the memory hierarchy can be avoided.

2.1.3 Feasibility and challenges

As a downside, any implementation of compression in the memory system will have overheads in extra logic, latency, and power consumption due to the compression/decompression logic. However, since the size, speed, and power consumption of logic (which will be used to do compression/decompression) scale better than those of interconnect (which will be used to communicate the information), these overheads will continue to decrease over time. Also, the (area, latency, or power) overheads that can be tolerated for compression/decompression vary from one part of the memory system to another and from application to application. For example, more compression/decompression latency overhead can be tolerated at higher levels of cache and main memory than at lower levels. Similarly, less latency overhead can be tolerated in higher performance systems than in non-performance-critical systems. Depending upon the state of the technology, the location in the memory system where compression is to be applied, and the application system requirements, the compression scheme can be more aggressive (better compression, but more compression/decompression overheads) or less aggressive (moderate compression, but less compression/decompression overheads), i.e., the compression scheme, and hence its overheads, can be suitably regulated. Accurate estimation of overheads of compression and decompression is possible only with respect to

a specific compression scheme and architecture, which is not the focus of this paper, but of our future work. Therefore, we concentrate in this paper on the limits to which compression can be potentially exploited.

2.1.4 CMS architectures and degree of specialization

In general, a compression scheme is designed to compress some new raw information based upon symbol statistics or frequencies drawn from some known or typical data set. Depending upon how specialized this data set is, five important classes of CMS architectures, from the most specialized to the least specialized, can be identified as described below. Note that in all cases, symbol statistics are drawn from the same type of information (address, instruction, data) as the type of information being compressed.

- *Block-specific architecture:* In this case, symbol statistics used to compress a block of information (e.g., a block in any cache or main memory or a word on a bus) are drawn from the same block. Such a compression scheme utilizes the most specialized information for compression, but it is likely to have the most complexity.

- *Memory-component-specific architecture:* When in a CMS architecture symbol statistics are drawn from the typical data set of a memory component and are used to compress each block of that component, it is referred to as memory-component-specific. For example, symbol statistics may be drawn from all the instruction addresses typically transmitted over the L1-L2 instruction address bus and then used to compress each instruction address transmitted over that bus.

- *Application-program-specific architecture:* In this case, symbol statistics used for compression of information in a memory component are drawn from the typical data sets found in a given application program in all memory components that store or communicate information of the same type.

- *Application-class-specific architecture:* In contrast to the previous case, here symbol statistics are drawn from application programs that belong to the same class (e.g., integer-computation-intensive applications or floating-point-computation-intensive applications), rather than from one particular application program.

- *General architecture:* In this case, symbol statistics used for compressing information in a memory component are drawn from a broad range of applications meant to be executed on a system and from all memory components that store and communicate the same type of information. Here the compression scheme utilizes the most general type of statistical information and is expected to provide some reasonable compression across a range of applications.

It is possible to use different degrees of specialized statistical information to perform compression in different parts of the memory system. Also, the compression scheme can be static or dynamic, i.e., the statistical information used for compression can be predetermined and fixed or it may change dynamically. A compression scheme is effective only if it is adapted to the characteristics of the source information it seeks to compress. That is why we have chosen to study the effect of varying degrees of specialization on the effectiveness of a CMS

architecture.

2.2 Related Work

Previous work in memory system compression has been done both in analyzing compressibility and in the development of specific compression schemes for the memory system. These include schemes for address compression and extension of these schemes to instruction and data compression, program code compression and compressed instruction set design for embedded systems, and main memory and cache compression. Related work in traffic optimization for low power using bus encoding has also been reported. We briefly review previous research in these areas next.

2.2.1 Previous analysis

In previous analytical research focusing on finding the potential for compression, separate studies by Hammerstorm and Davidson [24] and Becker et al. [4] used entropy models to evaluate the compressibility of addresses in microprocessors. Wang and Quong analyzed the potential of instruction compression [66]. They evaluated the effect of instruction compression on the average memory access time for various types of memory systems. Later, compressibility of program code in different architectures on various operating systems was investigated by Kozuch and Wolfe [38]. The potential of main memory compression was studied by Kjelso et al. [37]. We presented a brief analytical study of compression focusing on overall benefits for the memory system in [48] and a broader study in [50]. Apart from analytical studies of compressibility of memory system components, specific compres-

sion schemes have also been proposed for various memory system components. We briefly review them next.

2.2.2 Address, instruction, and data compression

Park and Farrens presented a *dynamic base register caching* (DBRC) scheme for compressing off-chip, processor-memory addresses in [52]. In this scheme, the original address is split into a higher order and a lower order component and the former is stored in cache of base registers and the index to the base-register cache is transmitted on the bus along with the uncompressed lower order part of the original address. They found that by using a 16-bit bus for a 32-bit microprocessor and the DBRC scheme resulted in only a miss rate of 2% and most of the time memory addresses could be transmitted using a 16-bit bus thus achieving almost a 50% reduction in the number of pins. Citron and Rudolph proposed a similar scheme, called BUS-EXPANDER (BE), for address, instruction and data traffic and bus compression [11]. They reported hit rates of up to 95% for their compression caches [11]. Both these schemes focused on reducing costs and improving pin bandwidth for off-chip accesses. Recently, the effectiveness of BE-like schemes to reduce the switching activity (power consumption) in off-chip data buses has also been studied [3]. Also recently, Kant and Iyer have analyzed the benefits of using dynamic-cache based compressed address and data transfer mechanisms for server interconnects by exploiting the spatial and temporal locality of addresses and data [30].

2.2.3 Code memory compression

Code memory compression schemes involve compressing the text segment of an executable program to reduce code size (decreases memory requirements) and thus save power and cost (larger memories consume more power and cost). Code memory compression schemes can be divided into three. The first category of schemes, called *code compaction* schemes use compiler optimizations during embedded code generation to minimize sizes of parts of code (like procedures and subroutines) that are used frequently. These are purely software techniques and require no hardware support during run-time. Various code compaction schemes have been reported in literature [22, 19, 35, 15, 17]. A second category of schemes, called *code compression*, refers to techniques that minimize code size of the executable and require decompression to be done before the compressed code can be executed. Among popular code compression schemes are compressed code RISC processor (CCRP) [70], call-dictionary compression [47], software-managed dictionary compression [42], semi-adaptive Markov compression (SAMC) and semi-adaptive dictionary compression (SADC) [44, 43], and IBM's CodePack for PowerPC cores [32, 23]. Code compression has also been proposed for VLIW architectures [14, 71] and have been recently adopted in commercial VLIW processors [27]. Simple instruction encoding schemes have also been proposed for low-cost, low-energy embedded processors [74, 5, 29]. The third category of code memory compression schemes are *compressed instruction sets* that are supported in popular RISC cores like ARM and MIPS [1, 36].

2.2.4 Cache and main memory compression

Memory is an important resource for both embedded and general purpose processors. IBM's Memory eXpansion Technology (MXT) [64] enables the microprocessor to interface with compressed memory (C-RAM) [21] and provides fast hardware compression and decompression to enable access to the memory without significant increase in latency. Selective cache compression techniques [41], frequent value data caches [72], and dynamic zero compression in data caches [65] have also been proposed and evaluated for performance and power improvements.

2.2.5 Bus encoding

Bus encoding is an area of research that has major implications in low power design of microprocessor systems. Encoding, although closely related to compression, is directed at minimizing unwanted signal transitions in the information stream to reduce bus switching energies during transfer rather than compressing the information itself. Various bus encoding schemes for off-chip address buses like Gray code [62], bus-invert code [60], asymptotic-zero (T0) code [6], and working-zone code [51] have been proposed and some of them [39] have been applied to data buses too. Most of these schemes involve the use of a redundant line that indicates if the current value on the bus is an encoded value or not. Some modified address bus encoding schemes that do not require any redundant lines have been suggested in [2]. More recently, bus encoding schemes have been proposed for on-chip buses taking into account the effect of inter-wire capacitances that are especially important in deep submicron designs [58, 25]. Apart from energy reductions, encoding schemes that reduce bus delay and

inter-wire cross talk have also been proposed.

2.3 Relationship of Our Work to Previous Research

To our knowledge, our comprehensive analysis of the potential of compression when applied to all parts of the memory system in the context of real-world benchmark programs and using extensive simulations is the first of its kind. The purpose of this chapter is not to present specific compression schemes, but to estimate the extent of compression possible in various memory system components. Towards this end, we employ existing compression tools and analysis methods (such as SAMC, Gzip, Markov models) to estimate the extent of compression possible and estimate the improvements in performance, power consumption, and cost improvements that can be obtained. We present results for all parts of the memory system using realistic timing, power, and area models (CACTI 3.0 [56] and SimplePower [76]). We also present results related to: (1) the compressibility of original, XOR, and off-set instruction and data address traces, (2) the effect of compression on cache access time, power consumption, and area, (3) the relationship between compression ratio and bit fields and bit-field groupings, (4) the effect of application class, degree of specialization, encoding and multiplexing, analysis tool, static vs. adaptive compression, multithreading, and (5) the relationship between information content, compression ratio, and power consumption, among other things.

2.4 Analysis Methodology

We analyze the potential for compression of a particular trace by measuring two parameters described below. First, *compression ratio*, *R*, for any compression scheme is defined as the ratio of the size of compressed information to the size of the raw uncompressed information. We use various entropy measures and some available compression schemes to estimate the information content or compression ratio possible for our traces. Second, *transition ratio*, T, for the compressed information is defined as the ratio of the number of transitions that occur when the compressed information is transmitted on the bus to the number of transitions that occur when the original uncompressed information is transmitted on the same bus.

2.4.1 Compression ratios from entropy calculations

The entropy of a source denotes the average number of bits required to encode each symbol present in the source. Thus, the lower the entropy value, the more compressible the source. Entropy values can be computed for a source based upon various models (zero information, zeroth order Markov, first order Markov, etc.). Compression ratios based on these models provide a theoretical lower-bound for a particular trace. We describe these entropy models and how we computed compression ratios from entropy values next.

Zero information entropy: Given a source with symbol set $s_1, s_2, \ldots, s_N$, the compressibility of a symbol in zero information entropy is determined by its presence or absence in the trace, irrespective of the number of times the symbol occurs in the trace. Thus, if there are M unique symbols that actually occur in a trace out of N total unique symbols that could occur, where $M \leq N$, the zero information entropy for that trace is $\log_2 M$, i.e., every one of

the M symbols that actually occurs is represented by a unique $\log_2 M$ bit pattern.

Zeroth order Markov entropy: Given that the source data has symbol set $s_1, s_2, \ldots, s_N$ and each symbol s_i occurs with probability $p(s_i)$, entropy for the symbol is $-\log_2 p(s_i)$. The zeroth order Markov entropy of the source data is given by the following relation: $H_0 = -\Sigma_{\forall i}[p(s_i) \cdot \log(p(s_i))]$. Whereas zero information entropy reflects only the occurrence/non-occurrence of symbols, zeroth order Markov entropy reflects in addition the frequency of occurrence of symbols.

First order Markov entropy: In first order Markov entropy, we consider the occurrence of a symbol s_i, the probability $p(s_i)$ of that symbol's occurrence, and the probability $p(s_j|s_i)$ that the symbol is preceded by another symbol s_j. The first order Markov entropy of a source is given by: $H_1 = -\Sigma_{\forall i}\left[p(s_i) \cdot \Sigma_{\forall j}[p(s_j|s_i) \cdot \log(p(s_j|s_i))]\right]$. This means that in a sequence of symbols if the current symbol is s_j and the next symbol is s_i, this next symbol s_i can be represented using $-\log_2 p(s_j|s_i)$ bits.

The symbols that we consider while measuring the entropy of any trace (address, instruction, data) correspond to aligned words in the trace, i.e., 32-bit words for addresses and instructions and 64-bit words for data. In our compression analysis study, we use only the low-order 32 bits of the actual 64-bit address in order to keep simulation times reasonable. Doing so results in a pessimistic estimate of the actual address compression potential since the high order address bits have large amounts of redundancy due to the spatial locality characteristics of addresses. Using the entropy values measured, the corresponding compression ratio can be computed by taking the ratio of entropy times the number of symbols (words)

to the number of symbols (words) times the size of a symbol (32 for addresses and instructions and 64 for data) in the original raw trace. Thus, for example, the *average zeroth order Markov compression ratio* over n benchmarks is:

$$R_{H_0} = \frac{\sum_{i=1}^{n} H_0 \text{ of trace}_i}{n \times \text{ Original wordsize}}.$$

R_H and R_{H_1} are defined similarly.

2.4.2 Compression ratios from practical schemes

Some specific schemes to compress address, instruction, and data have also been proposed recently. We used some of these schemes to measure compression ratios to obtain an estimate of efficiency obtainable with practical schemes.

Instruction and data block compression scheme: Semi-adaptive Markov compression (SAMC), a compression algorithm based on arithmetic coding combined with a precalculated Markov model was proposed by Lekatasas and Wolf for code compression [45]. We used the SAMC executable obtained from the authors to compress instruction and data blocks with the following parameters: block size equal to L1 or L2 cache block size depending on the level where the algorithm is applied, Markov model of depth 32 and width 256, and bits-per-probability of 4. The *average SAMC compression ratio* over n benchmark traces was calculated as follows:

$$R_{SAMC} = \frac{\sum_{i=1}^{n} \text{Size of compressed instruction or data trace}_i}{\sum_{i=1}^{n} \text{Size of original trace}_i}.$$

A point to note is that the SAMC algorithm is a block-based compression algorithm and hence average compression ratio for an individual block of that size is reported as the output.

Address compression scheme: Two techniques (dynamic base register caching and bus-expander) have been proposed to compress addresses that are transmitted on buses [52, 11]. Both schemes use a small fully associative cache at the sending end for compressing addresses and decompress them using registers at the receiving end. In our analysis, we use the bus-expander scheme to compress address streams. The *average address compression ratio* over *n* benchmark traces is defined as follows:

$$R_{Addr} = \frac{\sum_{i=1}^{n} \text{Size of compressed address trace}_i}{\sum_{i=1}^{n} \text{Size of original trace}_i}.$$

Data compression scheme: Gzip is a widely used GNU utility for compression in UNIX systems. It uses Lempel-Ziv (LZ77) dictionary compression algorithm which replaces strings of characters with single codes. Gzip does not do any analysis of the information source. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters. Since Gzip uses an algorithm based on bytes, good compression ratio is achieved on text files. We used Gzip on address, instruction, and data streams to provide an idea of compression achieved using a widely used text compression utility. The *average Gzip compression ratio* over *n* benchmark traces is defined as follows:

$$R_{Gzip} = \frac{\sum_{i=1}^{n} \text{Size of compressed trace}_i}{\sum_{i=1}^{n} \text{Size of original trace}_i}.$$

2.4.3 Transition ratio

For CMOS technology, power consumption on a bus line is directly related to the switching activity on it as bits are transmitted one after another over it. We use a methodology

similar to the one used in SimplePower [73] to calculate the switching activity of a given bus when information is transmitted across it. They calculate the average probability of a transition in each bit of the bus and find the total average probability across all bits, which is a measure of the per-input switching activity of the bus in bits [76]. Thus, the ratio of bus power consumption for two traces using the SimplePower model is equal to the ratio of the number of transitions for those two traces. We define *average transition ratio* over *n* benchmarks for compressed traces as follows:

$$T_C = \frac{\sum_{i=1}^{n} \text{No. of transitions in compressed trace}_i}{\sum_{i=1}^{n} \text{No. of transitions in original trace}_i}.$$

2.5 Simulation Environment

Our target system has a memory hierarchy consisting of 32 integer and 32 floating-point registers, split instruction and data caches at the first level, a unified cache at the second level, and a paged main memory. The first level caches are write-through, 16KB each, 4-way set associative, and have a block size of 32 bytes. The second level cache is write-back, 256KB, 4-way set associative, and has a block size of 64 bytes. For this target memory system configuration, we used a modified version of the *cachesim5* cache analyzer in SHADE5 [12] running on SPARC-V9 platform to collect the real-time traffic (addresses, instructions, and data) for benchmark programs. *Cachesim5* simulates cache operation by using address information and hence can be easily modified to collect address bus traces. But we also needed to collect instruction and data block traces for our analysis. To facilitate this, we augmented *cachesim5* by creating an interface to map addresses to the appropriate location

in memory where the instruction and the data blocks are located. This way, we were able to collect the actual address, instruction, and data traffic between processor, caches, and memory for our analysis.

We used benchmarks from the SPEC CPU2000 suite [16]. To capture the characteristics of both integer and floating-point programs, we chose eight integer and seven floating-point benchmarks randomly out of 26 in the suite. For some experiments, especially when studying the effect of workloads, we additionally used five benchmarks from the MediaBench suite [40]. A summary of our complete benchmark set is shown in Table 2.1. We used the -O2 optimization flag, which does basic local and global optimization to compile these benchmarks. All executables were statically-linked, in which the procedures and libraries are linked with the main program during compilation itself. We ran the benchmark programs using reference input sets provided with the SPEC2000 suite and to limit the execution times of our simulations we used a methodology similar to the one described by Skadron, et al. [57]. Their research shows that accurate simulation results can be obtained by avoiding unrepresentative behavior at the beginning of a benchmark programs' execution and by using a single, short simulation window of 50 million instructions. In our simulations, we simulate (but do not collect results for) instructions before the representative segment (warm up window) and use a sampling window of 50 million instructions to collect our results. The sizes of the warmup windows are also different for different SPEC programs [57]. These are also summarized in Table 2.1. For MediaBench programs, we used input sets provided on the MediaBench website and collected results for complete execution.

Benchmark	Representative Application	Warmup window	Inputs
SPEC INT [16]			
gcc	GNU C/C++ compiler	221M	200.s
gzip	Text/file compression utility	2576M	input.source
vortex	Object-oriented database	2451M	bendian1.raw
parser	Word processing	500M	ref.in
crafty	Chess game playing program	500M	crafty.in
twolf	VLSI place and route	500M	ref
mcf	Combinatorial optimization	500M	inp.in
vpr	FPGA circuit placement and routing	500M	arch.in, net.in
SPEC FP [16]			
applu	Parabolic-elliptic partial differential equation solver	500M	applu.in
swim	Shallow water modeling	500M	swim.in
wupwise	Physics/quantum chromodynamics	500M	wupwise.in
lucas	Number theory/ primality testing	500M	lucas2.in
art	Image recognition with neural network	500M	c756hel.in, a10.img, hc.img
ammp	Computational chemistry	500M	ammp.in
equake	Earthquake simulation	500M	equake.in
MediaBench [40]			
jpeg	JPEG image compression and decompression	testimg.jpg, testimg.ppm	
adpcm	Family of speech compression and decompression algorithms	clinton.pcm, clinton.adpcm	
gsm	European GSM 06.10 provisional standard for full-rate speech transcoding	clinton.pcm, clinton.pcm.gsm	
ghostscript	An interpreter for the postscript language and portable document format (PDF) files	tiger.ps	
rasta	Program for bandpass filtering additive noise and spectral distortion in speech recognition systems	map_weights.dat (mapping coefficient file for 8KHz 15 critical bands) and speech file in SPHERE format.	

Table 2.1: **Summary of Benchmark Set and Input Files Used for Our Simulations.** SPEC CPU2000 (INT and FP) benchmarks were used in all experiments. MediaBench programs were used in an experiment studying the effect of different workloads on compression. All input files for SPEC CPU2000 programs are available with the benchmark suite. Input files for MediaBench programs are available from the MediaBench website.

For communication components, we performed experiments on traces of address, instruction, and data traffic between the processor and memory for all three levels: processor-L1 cache, L1 cache-L2 cache and L2 cache-main memory for each benchmark and calculated the zero information, zeroth, and first order Markov entropies, and SAMC compression ratio in each case and, in some cases, we also calculated the Gzip compression ratio. We investigated the compression potential of storage components other than registers by calculating zero information, zeroth order Markov, and first order Markov entropy values, and R_{SAMC} and R_{Gzip}. For main memory, we calculated these values for the text segment of the statically-linked executable code. For registers, we performed only zeroth order Markov analysis. The reason we did not do a first order Markov analysis for registers is because a compression scheme that exploits first-order behavior will need to represent the current value in a register in a manner that depends upon the previous value. Since a register has only one word, storing the previous and current values, even in compressed form, is unlikely to yield much compression. Moreover, if register compression is attempted, the compression scheme needs to be simple enough not to affect access latency by more than a little.

To keep the number of simulations reasonable and at the same time be able to study a number of parameter variations, we consider certain default settings as follows. We consider the default architecture to be memory-component specific as described earlier in Sec. 2.1.4. Also, in the default case, for our communication component analysis experiments, we consider demultiplexed buses, in which case there are separate buses for instruction address, data address, instruction, and data. In some cases, we consider a multiplexed bus, with one

'address' bus carrying both instruction and data addresses and one 'data' bus carrying both instructions and data. Also, the default level for which we report most of our results is between L1 and L2 caches. The default word size considered as a symbol size in Markov entropy calculations is 32 bits for address and instruction, 64 bits for data, and 20 bits for tag field (See Sec. 2.4.1 for an explanation regarding why we use 32-bit instead of the actual 64-bit address). For entropy analysis, in most cases, first order Markov provides the best results and the performance of zeroth order Markov is also better than zero information. We present these two entropy results in most of our plots. In the experiments that we describe next, we summarize results in plots by averaging over all 15 (8 INT and 7 FP) benchmarks or by showing averages for INT, FP, and MediaBench programs separately for specific components. We calculate the average compression ratios as mentioned earlier in Sec. 2.4.2.

2.6 Trace Collection

For communication components, traces were collected by writing each new value transmitted on a bus (connected between two storage components or between the storage component and the processor) and its corresponding timestamp into a file. Thus, we assume that bus lines are held at previously transmitted values when the bus is idle.

For storage components, the following methodology was adopted to collect dynamic traces and to ensure that the analysis done reflects average compressibility of the component. In instruction caches, a block may be loaded into and be replaced from a cache multiple times during the sampling window of the simulation. A load and the next replacement of a

block correspond to a time period during which it is resident in the cache known as its cache residence time. Since the time instant of a load that occurs before the sampling window and that of a replacement that occurs after the sampling window are not known, we ignore these time periods to avoid errors and consider only load-replacements that occur during the sampling window. In a data cache, a data block in cache during the sampling window can take on one or more values because of writes to it. Therefore, for data caches, we consider the all data block values (instead of data blocks) that are acquired and replaced during the sampling window.

During the simulation, we keep a record of the block address and CRT of each block that is loaded and replaced during the sampling window. After simulation, we list the blocks in non-increasing order of CRTs and sum the CRTs of all blocks to get the total CRT (TCRT). Then, starting from the first block, we select blocks in the list in order until the total residency time of selected blocks becomes equal to 80% TCRT. Then we write in random order the actual contents of these selected blocks a number of times in proportion to each block's CRT into a file to obtain the trace for our experiment. We do the same for 90% TRCT. We use a random order to write the blocks to avoid any optimistic first order compression ratios that may be obtained if the blocks were written in order of their sorted residency times.

For 80% and 90% TCRT traces, we consider only blocks up to 80% or 90% TCRT and scale down the number of occurrences of such blocks (and hence the number of times they need to be written to the trace file) by that of the last block selected. This simplifies the creation of the trace file compared to a 100% TCRT trace because in the latter the total number

of times each of the blocks needs to be written into the trace will be extremely large for some blocks. Compression analysis for a cache is then done by analyzing its corresponding trace file. The above discussion applies to the data field of instruction and data caches. From the addresses and CRTs of the blocks available for the instruction caches, we are able to create similar trace files for analyzing tag field compression.

Adopting a similar methodology as above for register compression analysis, we considered the residency times of only those values that are loaded and replaced in a register during the simulation window. Note that by considering the residency times of blocks as above, both in the case of cache and register, the trace file we created reflects the average contents of the cache/register. Hence the compression ratios obtained would be those expected from a compression scheme that chooses encodings based on average symbol statistics, rather than one where the choice changes dynamically as cache/register contents change. Therefore, the compression ratios we report in our studies are, in this sense, not optimistic.

2.7 Overall Memory System Analysis

We investigated how compression ratio and power consumption vary across memory system components, namely, registers, cache, main memory, address bus, instruction bus, data bus. The compression ratio is indicative of the extent to which performance enhancement or cost savings can be realized. Figure 2.1 presents an overview of our analysis. We observe that communication components are in general more compressible than storage components (considering H_1 values which provide the best lower bound for entropy). Among storage

components, we observe that the ordering from the most to the least compressible is L1 I-cache data field, L1 I-cache tag field, main memory, and registers.

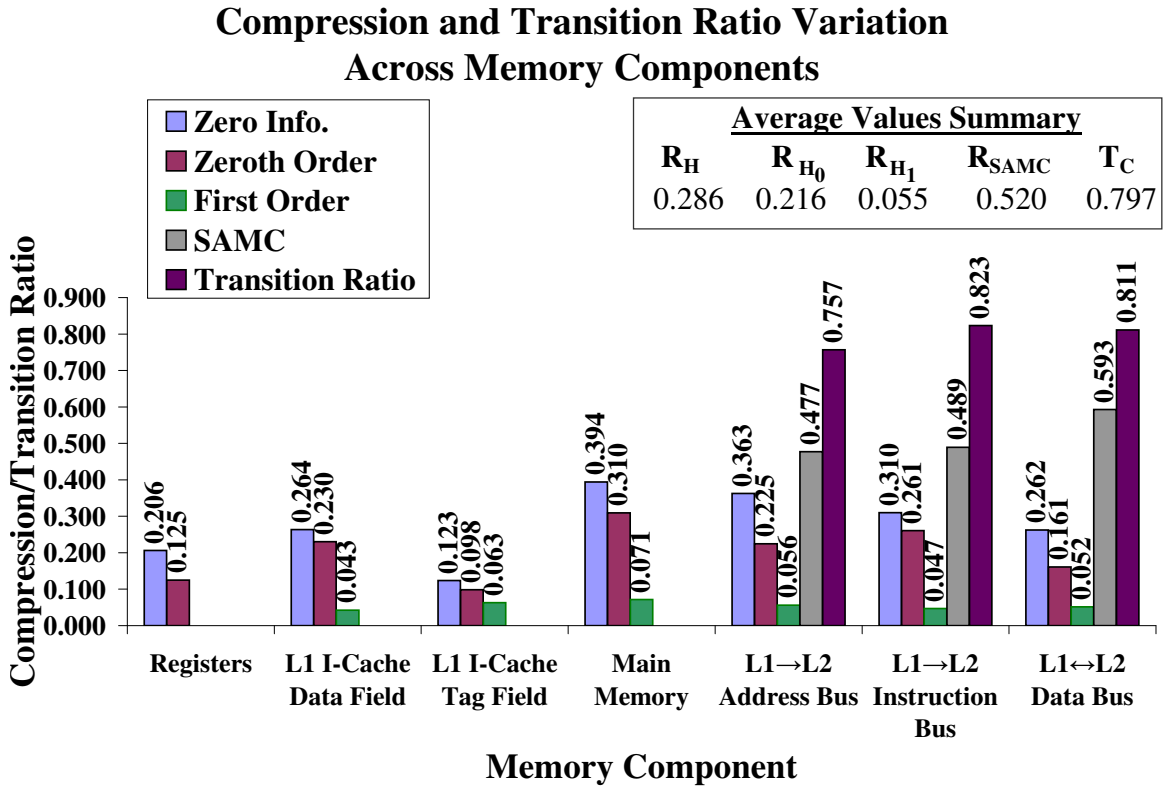


Figure 2.1: **Overall Memory System Analysis:** Compression ratio variation across memory system components. Communication components are in general more compressible than storage components when first order entropies are considered.

This is to be expected since instructions that are stored in the data fields of I-cache and tag field that corresponds to the high-order portion of the instruction address carry significantly higher amounts of redundancy than main memory or registers. Among communication components, the ordering, from the most to the least compressible (again considering H_1 values), is instruction bus, data bus, and address bus. A possible explanation for the higher redundancy in the data bus compared to address bus is that a lot of the data blocks transmitted may

contain small magnitude numbers that have lots of either 0 or 1 bits. Further, it is observed that the volume of data read traffic (data blocks sent from L2 to L1) is far greater than the write traffic (data blocks sent from L1 to L2), which means that the same blocks may appear in the data bus traffic often without any changes, and this also increases the redundancy. This also explains why data traffic shows the best compressibility in zero information and zeroth order analysis. We also observe that the ordering of the communication components in terms of power savings after compression (from most to least savings) is as follows: address bus, data bus, and instruction bus.

2.8 Register Compression Analysis

For register compression, we performed zeroth order Markov analysis over all 32 integer registers and 32 single-precision floating-point registers in our target architecture. In SPARC-V9, all integer registers are 64 bits each and the single-precision floating-point registers are 32 bits each [67]. The floating-point register file (FPRF) uses *aliasing*, i.e., some register names overlap. For example, the 32 single-precision register set, the lower half of the 32 double-precision register set, and the lower half of the 16 quad-precision register set overlay each other. Considering the total number of registers in our analysis and keeping track of all values stored in them for large samples (50 million instructions) would have been computationally intractable. Hence, we study only instructions that manipulate registers in the single-precision FPRF.

Fig. 2.2 and Fig. 2.3 show the zero-information and zeroth-order compression ratio for

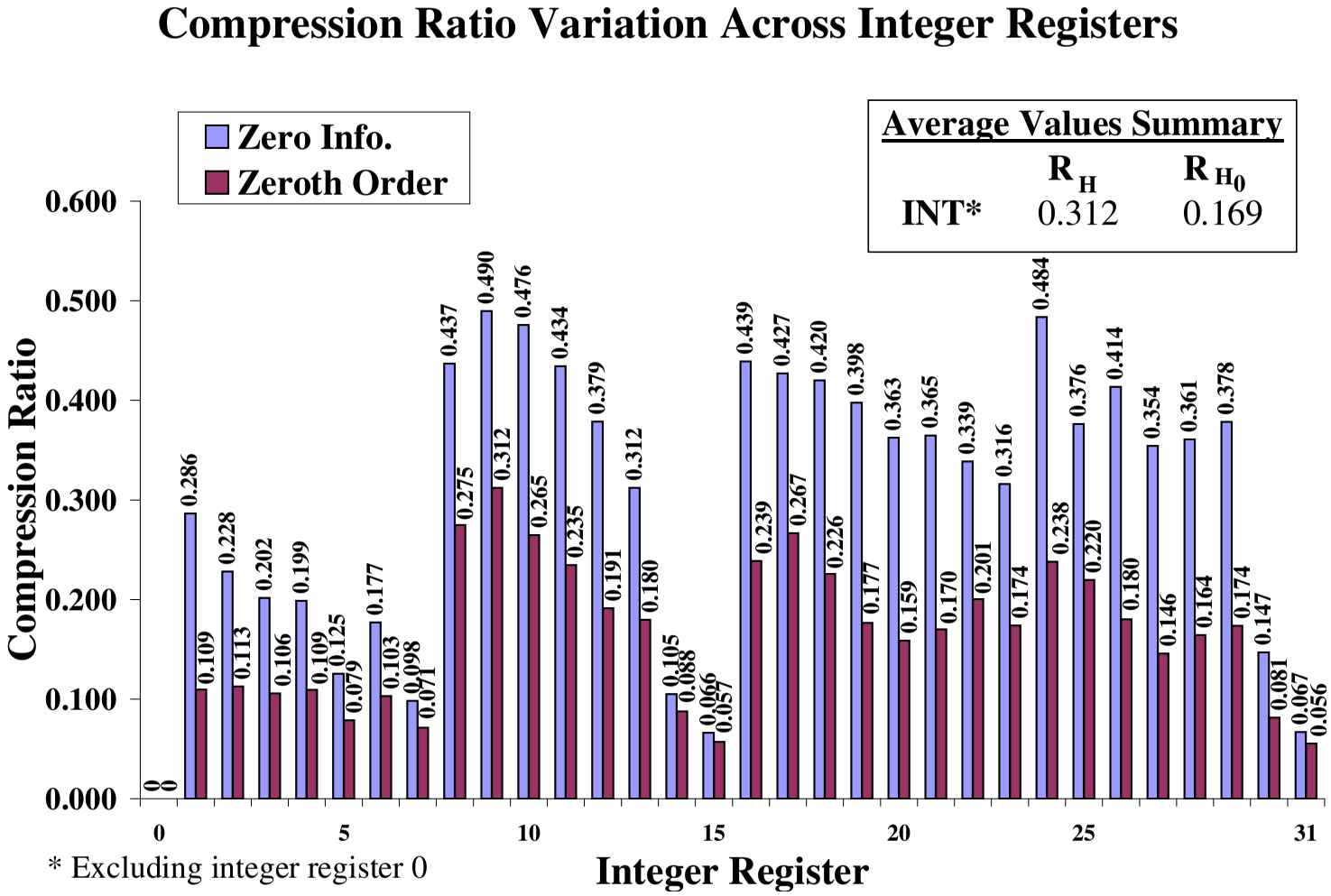


Figure 2.2: **Compression Potential of Storage Components – Register Compression Analysis:** Average register compression analysis for 32 integer registers.

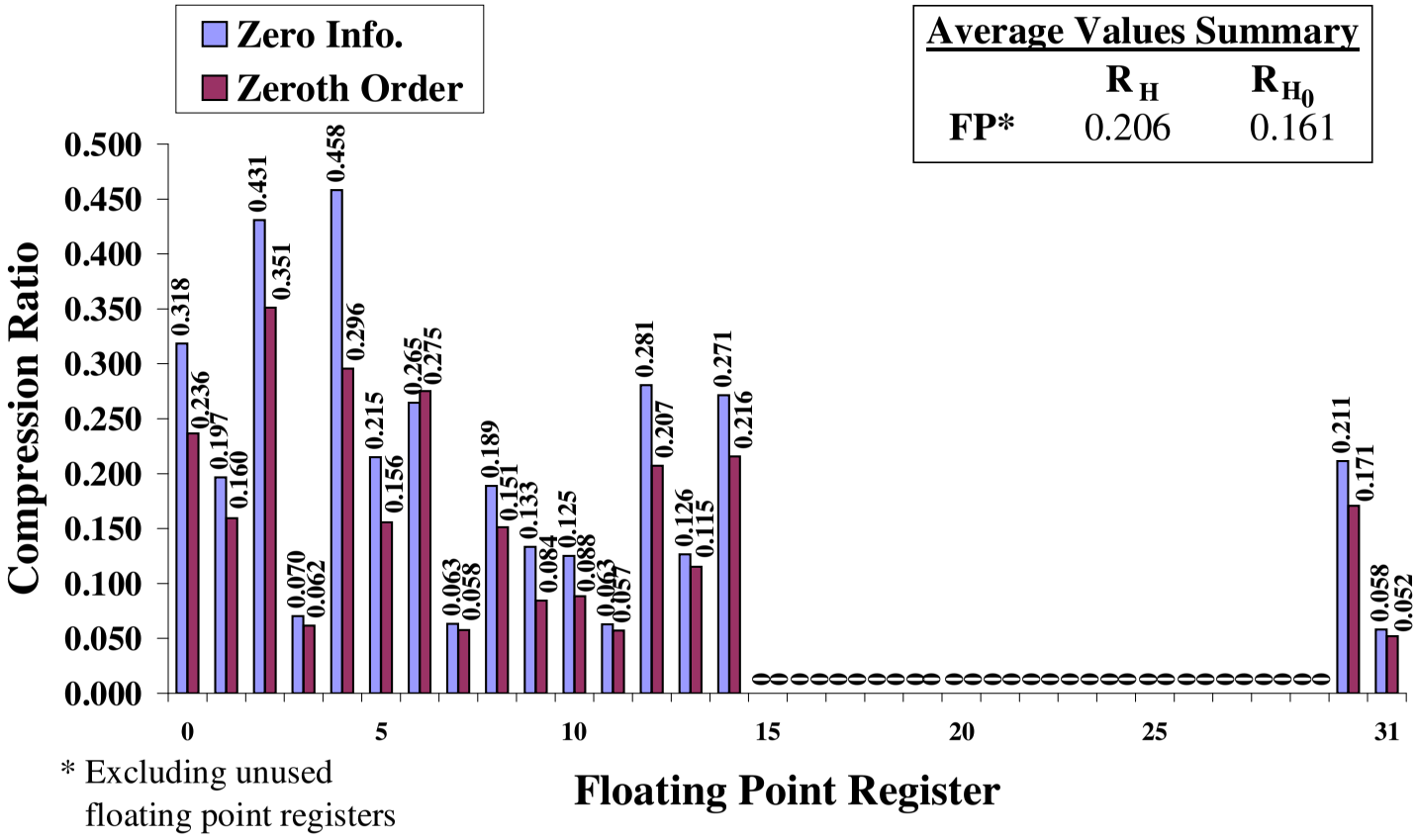


Figure 2.3: **Compression Potential of Storage Components – Register Compression Analysis:** Average register compression analysis for 32 single-precision floating-point registers.

each register in the integer and floating-point register file. Considering average values, we find that floating-point registers are more compressible than integer registers. The following observations can be made from the plots.

- *Integer register compression:* The average zeroth-order integer register compression ratio across all 32 registers, excluding register r0, is 0.169. We observe that integer registers r1-r7, r14, r15, r30, and r31 show potential for more compression than the rest. This can be attributed to the register windowing employed in the SPARC register architecture: r1-r7 correspond to the most often used 'global' set of registers which, due to their more frequent usage show higher compression potential. Also, registers r14, r15, r30, and r31 have dedicated use as stack, frame, temporary, and return-address register respectively and may hence be used more frequently than others. But, one may argue that many integer registers can potentially contain pointer values[1] (32-bit addresses of other locations where data is actually stored) that can take large values and hence may be poorly compressible. But, there is indeed a lot of redundancy present in pointers because they point to roughly a similar region in memory (since they are dynamically allocated). Hence many of their high-order bits will be the same and hence redundant.

- *Floating point register compression:* The average zeroth order floating-point register compression ratio we observe for the 32 single-precision registers in SPARC-V9 is 0.161 (excluding registers f16-f29 that were all unused). Note that, as opposed to inte-

[1] Pointers in SPARC-V9 are 32 bits. A simple C program using the sizeof(void *) functions will reveal this.

ger registers, a symbol size of 32 bits was used here to calculate entropy because only single-precision operands were considered. The substantial underutilization of the register set—13 out of 32 were not used by the benchmarks at all—can be explained by the fact that these may have been used as double or quad-precision registers.

In summary, our results show that although there is good amount of variation in compression ratio across registers, no register (INT or FP) has an average H_0 compression ratio exceeding about 0.35, which implies registers can, on average, be compressed to about one-third of their original size using a very good zeroth-order compression scheme.

2.9 Cache Compression Analysis Across Different Levels

In this section, we analyze the compressibility of L1 and L2 caches. First, we explore the potential for instruction cache and data cache compression in separate experiments. Then, we investigate the effect of change in cache parameters (cache size, block size, and associativity) on compression. Finally, we estimates the benefits of cache compression in terms of improvement in cache access times, reduction in consumption, and reduction in area.

2.9.1 Instruction and data cache compression

Fig. 2.4 shows results for compression ratios calculated using zero information, zeroth order, and first order Markov entropies for instruction and data caches. To limit the running times and memory required for this analysis, we limited the sample size used for cache trace collection to 20M instructions and 100% TCRT cache traces are collected for this study.

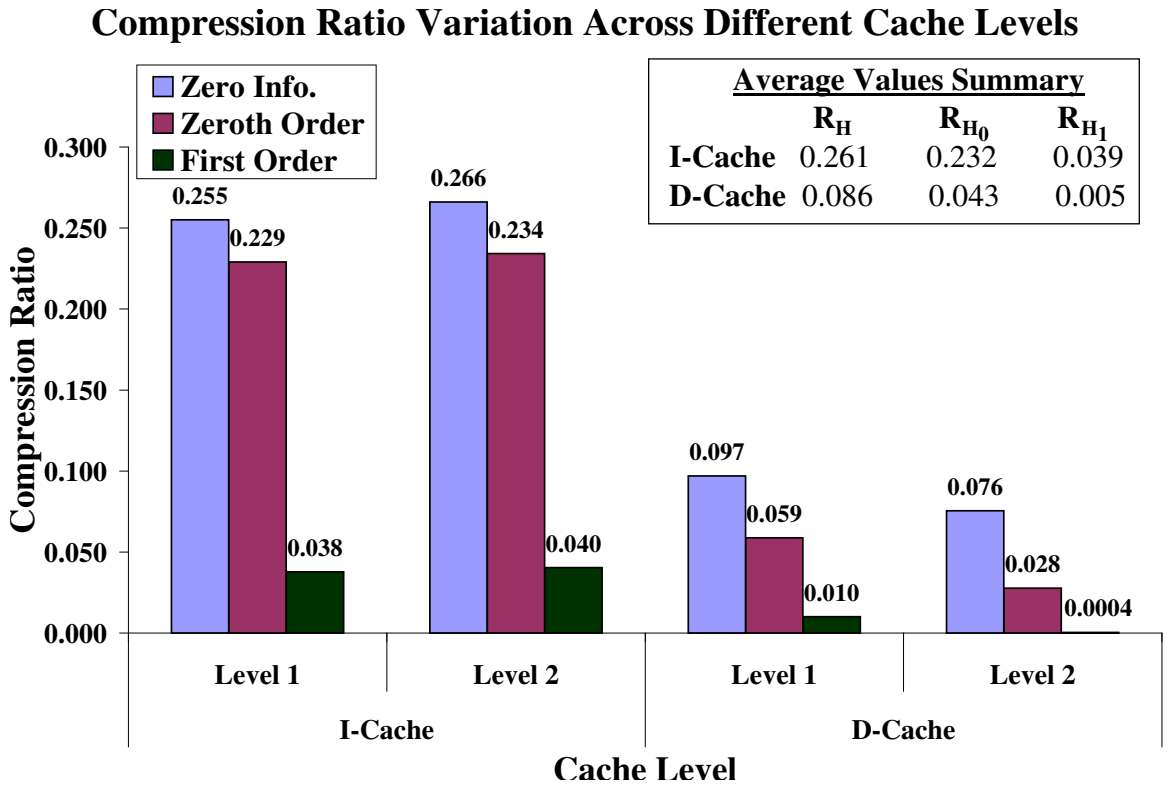


Figure 2.4: **Compression Potential of Storage Components – Cache Compression Analysis:** Average instruction and data cache compression analysis for L1 and L2 caches.

Comparing instruction and data caches, we observe that data caches are more compressible. One reason for this could be the presence of data blocks with uninitialized values (mostly zeros) that add to redundancy. Comparing between L1 and L2 caches, it would be expected that L1 cache will be more compressible, if both L1 and L2 blocks are dynamically compressed with the same scheme, due to the following reason. L1 cache contains a more frequent symbol set (of instructions or data) and the L2 cache, in addition to storing the contents of L1, also contains additional symbols (instructions or data) that are relatively infrequent. This is observed to hold in the case of instruction cache, but for data caches we observe that L2 is more compressible than L1, albeit slightly (by about 3% or less). One

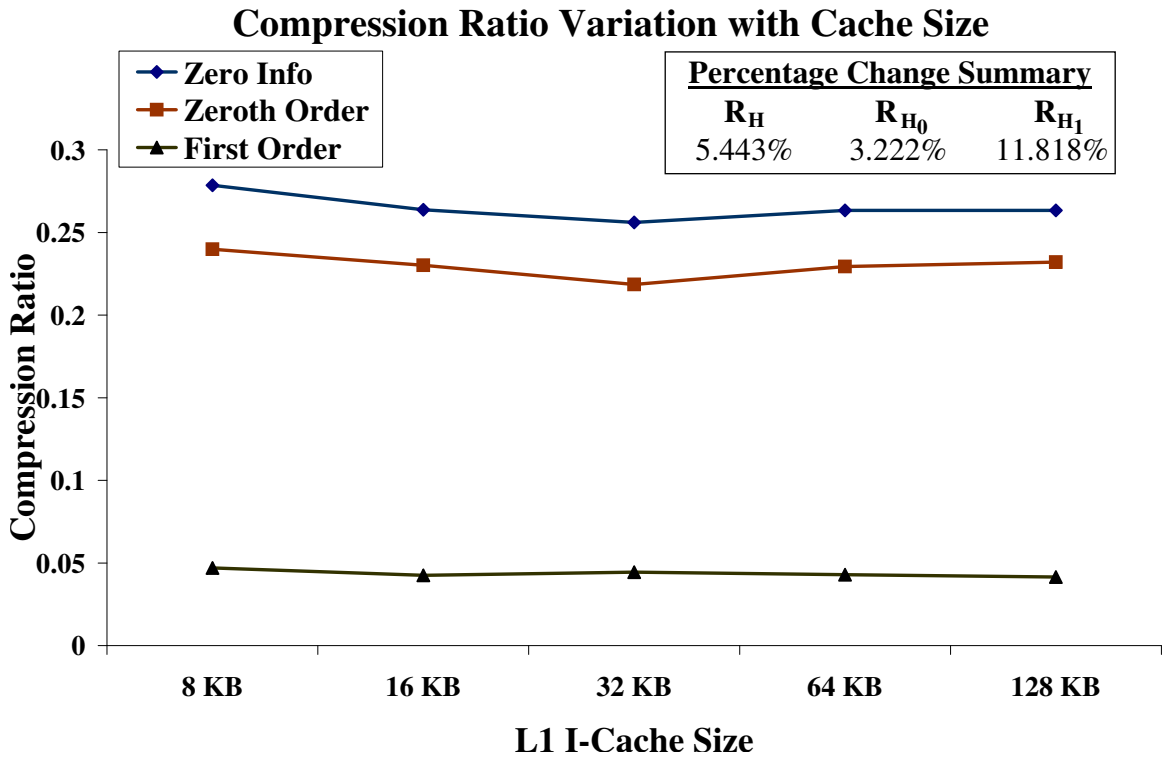


Figure 2.5: **Cache Compression and Cache Size:** With increasing cache size, compression ratio deteriorates somewhat.

possible explanation is that, since data is more dynamic in nature compared to instructions, blocks in L1 cache tend to be replaced more frequently. This tendency may have been aggravated by a small L1 data cache size (16KB). Both these factors result in a more dynamic mix of data in the L1 cache trace making it less compressible. As we will see later in Sec. 2.9.2, increasing cache size from 16KB to 32KB could have resulted in better compression for L1 D-cache. In contrast, due to the larger size of the L2 cache (256KB), data blocks tend to stay longer and thus the L2 data cache trace is more compressible. On average for instruction caches, we observed a zeroth order Markov compression ratio of about 0.23 and a first order Markov compression ratio of about 0.04. This means that, theoretically, we could reduce

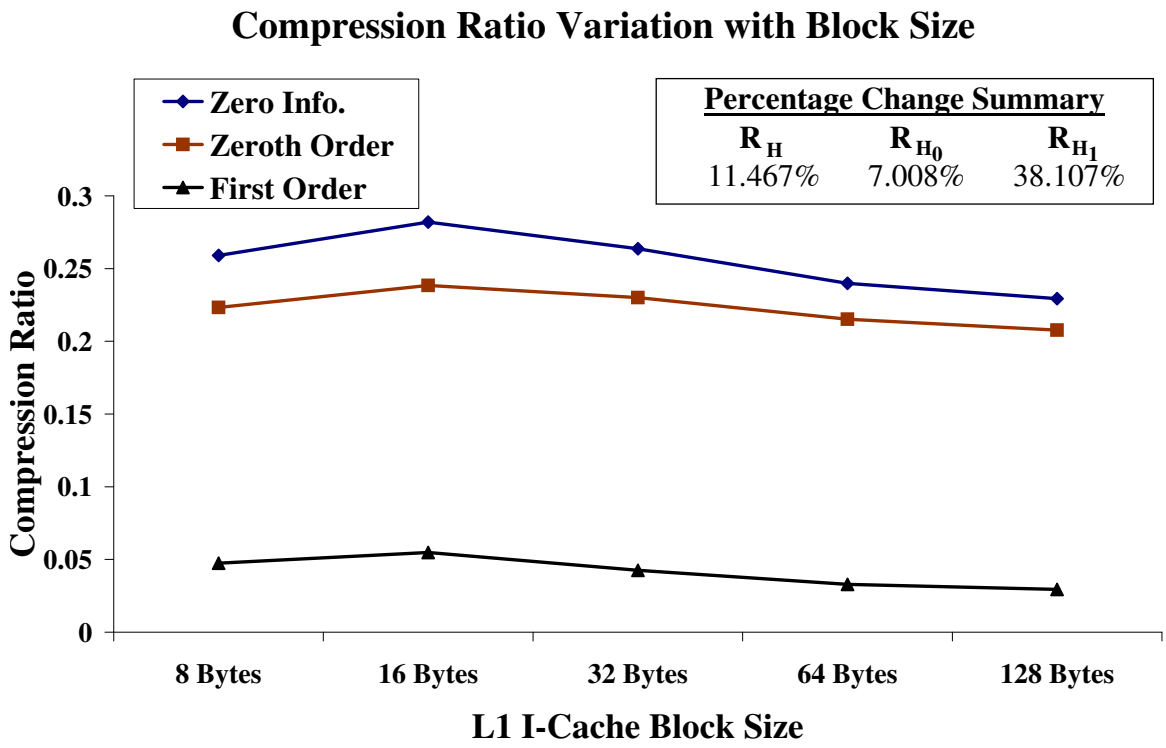


Figure 2.6: **Cache Compression and Block Size:** With increasing block size, compression ratio improves. Cache associativity has negligible impact on compression ratio.

instruction cache sizes by about 4 to 25 times by applying cache compression methods or store that much more information in the same area.

2.9.2 Compression ratio and cache parameters

We also investigated the sensitivity of cache compressibility to cache parameters, namely, cache size, block size, and degree of associativity and its relationship to access time, power consumption, and area. All experiments in this set were done on L1 instruction cache resident blocks. From Fig. 2.5, we find that the compression potential of cache first increases and then decreases with increasing cache size. For the range that we studied, cache compression potential is maximum for a 32KB cache. A larger cache has more relatively infrequently

occurring blocks than a smaller one, and that explains its lower compressibility. However, even for large caches, the compression ratio is very good.

In general, compression ratio improves when we increase block size as shown in Fig. 2.6. This is because a larger block has more spatially close instructions than a smaller one, and so, for the same cache size, increasing block size increases the number of instructions that are related to each other, and a smaller block size leads to more block boundaries where interruptions in related instructions occur. We also performed experiments to test the impact of varying cache set associativity on compression and we found that it has negligible impact on compression ratio.

Cache Type	**Comp. Method**	**Access Time**		**Total Power**		**Area**			
		(ns)	(% redn.)	(nJ)	(% redn.)	Tag (cm^2)	Tag (% redn.)	Data (cm^2)	Data (% redn.)
L1	Uncomp.	1.2790	–	1.6889	–	0.00112	–	0.01163	–
L2	Uncomp.	1.7363	–	3.0679	–	0.00514	–	0.12910	–
L1 I-cache	Zeroth-order	1.2357	03.385	1.5845	06.185	0.00063	43.75	0.00633	45.57
L1 D-cache†	Zeroth-order	0.7550	40.969	0.5767	68.854	0.00023	79.46	0.00179	84.61
L1 I-cache†	First-order	0.7550	40.969	0.5767	68.854	0.00023	79.46	0.00179	84.61
L1 D-cache†	First-order	0.7398	42.158	0.5724	66.108	0.00020	82.14	0.00160	86.24
L2	Zeroth-order	1.3011	25.065	1.8850	38.558	0.00117	77.24	0.02549	80.26
L2	First-order	1.2398	28.600	1.7281	43.672	0.00023	95.53	0.00179	98.61

Table 2.2: **Access Time, Power Consumption, and Area of Caches:** Cache parameters obtained using the CACTI 3.0 model. Entries marked with a † use a direct-mapped organization for the compressed cache.

2.9.3 Cache compression and cache access time, energy consumption, and area

To measure the effect of compression on other parameters like access time, power consumption, and area of the tag and data arrays, we used the CACTI 3.0 model [56] for a 0.18 micron SRAM cache implementation. Table 2.2 gives values of these parameters for L1 and L2 caches. Here, we compare a normal uncompressed cache with a smaller (by compression ratio) compressed cache having the same effective storage capacity. Both cache have similar parameters, such as block size and set associativity, but the compressed cache has fewer blocks (compression ratio times the number of blocks in the corresponding normal uncompressed cache). In some cases, however, the size of the compressed cache was too small (due to the compression ratio being very small) to use a set-associative mapping in CACTI 3.0. In those cases, we used a direct-mapped cache implementation. We observe that with tag and data field compression in the compressed cache, access times can be reduced by about 41%(29%) and power consumption by about 66%(44%) on the average for L1 (L2) levels w.r.t. normal uncompressed caches with the same effective capacity.

2.10 Compression and Transition Ratio Across Individual Buses

Zeroth order and first order redundancies in all buses

Figure 2.7 shows compression and transition ratio results for demultiplexed buses at all three levels. We observe that the R_H and R_{H_0} values are similar across all levels. Based on R_{H_1} values, instruction address is most compressible and data address least, except for L2-M, where data is most compressible.

Original, XOR, and offset address trace compression

Since instruction and data addresses are known to exhibit spatial redundancy to different degrees, it would be expected that the XOR of consecutive addresses will have a lot of zeros (especially at the high order bit positions) and that the offset values for consecutive addresses will have small magnitudes. Note that computing bitwise XOR of two n-bit addresses requires constant time and little hardware and offsets can be computed in $O(\log N)$ time using a carry lookahead tree adder. However, XOR traces have a power disadvantage. Every bit transition in the original trace will cause two bit transitions in the XOR trace, except when consecutive transitions occur in the original trace (not likely), in which case there will not be any transition in the XOR trace. To study the compressibility of original, XOR, and offset address traces, we evaluated their zero information, zeroth order, and first order Markov compression ratios, which are shown in Figure 2.8. Since instruction addresses occur at some very frequent offsets (typically an instruction word), the zero information and zeroth

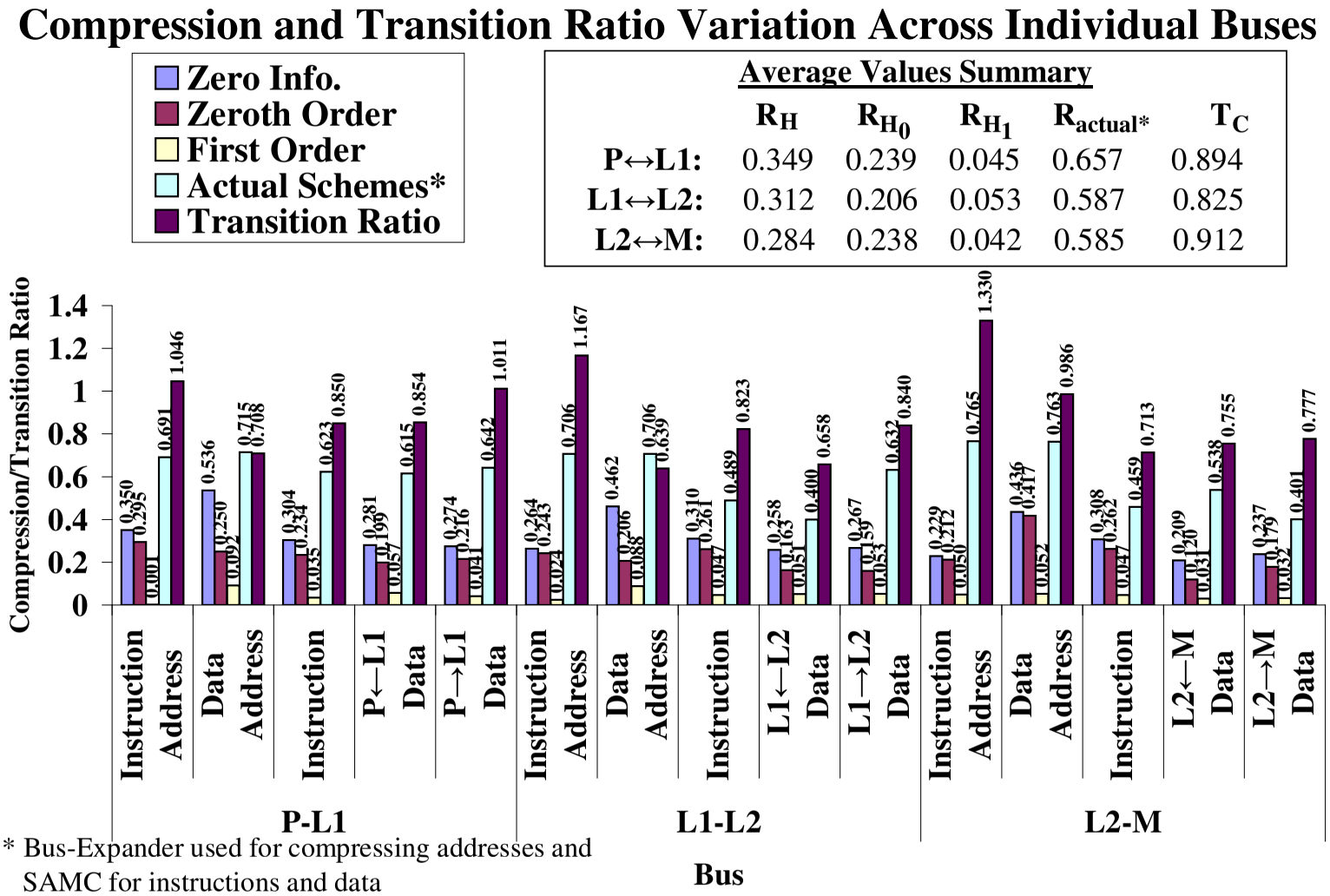


Figure 2.7: **Compression Potential of Communication Components:** Compression ratios for zeroth and first order behavior of various buses at different levels of the memory system hierarchy.

order Markov compression ratios for instruction address offset traces is the best and even the XOR trace has better compressibility than the original trace. However, when considering first-order Markov compression, the original trace provides the best compression and the offset trace the worst. This is expected since, given an offset, the next offset value can vary depending upon the instructions being executed at the time. However, given an instruction address, the next instruction address can be easily predicted. In the case of data addresses, XOR and offset traces do not necessarily give better compression ratios due to more variation in data addresses issued.

2.11 Compression Ratio and Bit Fields

In this experiment, we consider eight consecutive bit fields (from high to low order: F7, F6,. . . , F1, F0) corresponding to each nibble for instruction and data addresses. For 64-bit data, we consider four consecutive bit fields (F3, F2, F1, and F0) corresponding to each half-word (16 bits). For 20-bit I-cache tag, we consider four fields each a nibble wide. For 32-bit instructions, we consider six fields (F5, F4, F3, F2, F1, F0) of widths 2, 5, 6, 5, 9, and 5 bits respectively based on the field boundaries of the most common instruction format in SPARC-V9 architectures (J-format). In the experiments under this section, the symbol size for compression corresponds to the above-mentioned bit field sizes. We generated individual bit-field traces for data addresses and instruction addresses at the P→L1 level, instructions and data at the L1→L2 level, and tag field of L1 I-cache and then analyzed each trace by doing zeroth and first order Markov analysis. We also considered three different representations

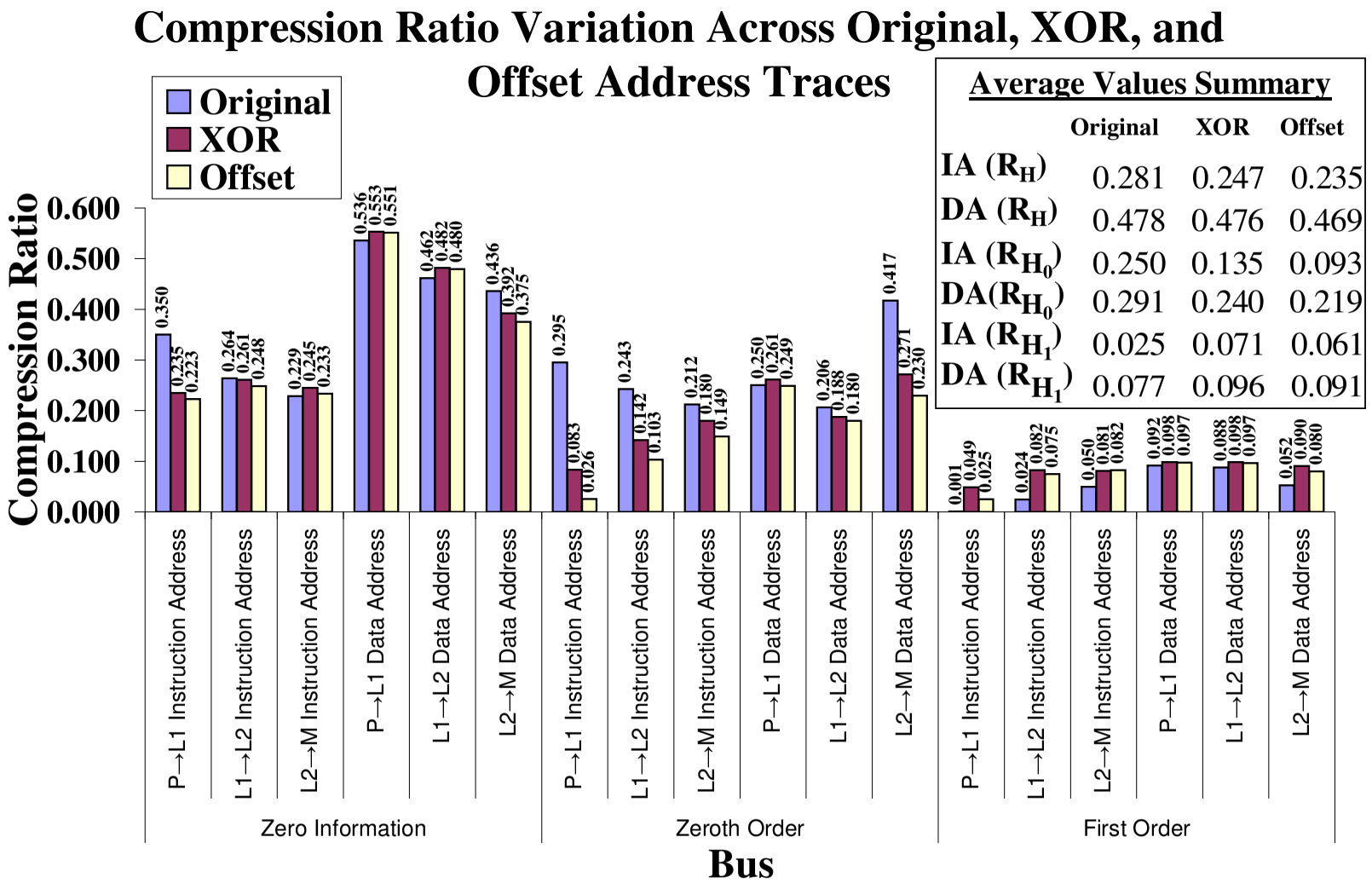


Figure 2.8: **Original, XOR, and Offset Address Trace Compression:** Compression ratios for original, XOR, and offset address traces for various address buses.

for each bit-field stream in addresses: original (raw), XOR-encoded, and offset-encoded. The motivation for studying these address representations was described earlier in Sec. 2.10.

According to the results shown in Figs. 2.9- 2.13, we observe that compression ratio varies across bit-fields and the variation differs for each type of traffic. In general, across all types of information, we observe that compressibility improves from low order to high order bit fields, except somewhat in the case of instruction bus traffic. Comparing data addresses and instruction addresses (Figs. 2.9 and 2.10), we observe the following. First, instruction addresses are more compressible than data addresses. Second, zeroth-order and first-order compression of bit-fields yield more returns in instruction addresses than in data addresses. Third, offsets and XORs of instruction addresses are more compressible with higher-order compression schemes.

2.12 Compression Ratio and Bit-Field Groupings

We also investigated how compression ratio varies depending upon grouping of bits fields for compression. We considered five bit-field groupings for addresses that are mentioned in the top right corner of Fig. 2.14: Group-1 (G1) consists of 8 nibbles with each compressed separately, Group-2 (G2) consists of a most significant byte followed by 6 nibbles, Group-3 (G3) comprises a most significant part of 12 bits followed by a byte and then two nibbles, Group-4 (G4) consists of a most significant half-word, a byte, and then a nibble, and finally Group-5 (G5) considers the whole word as a symbol. In a similar vein, the bit-field groupings that we considered for instruction, data, and cache tag fields are shown in Fig. 2.15.

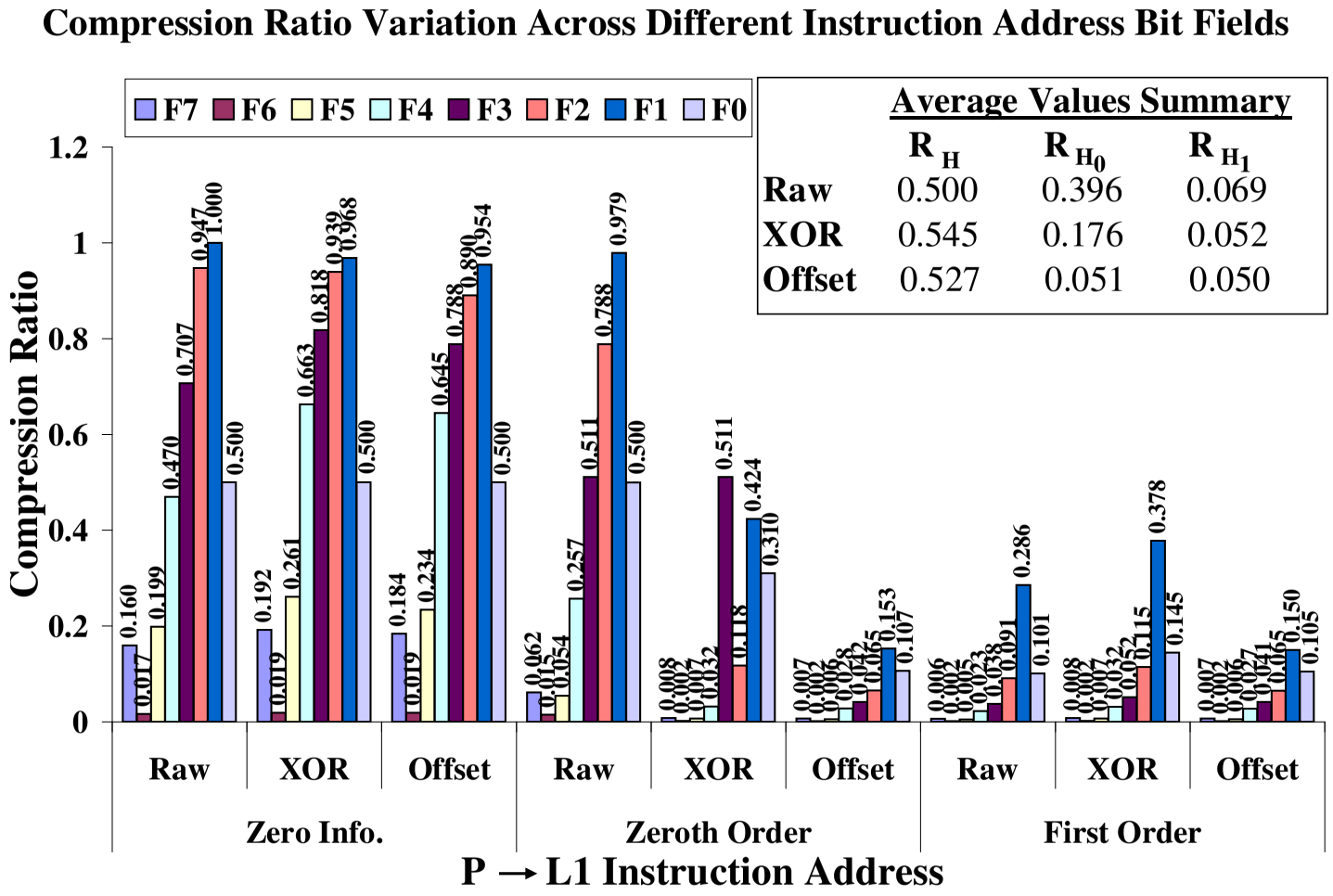


Figure 2.9: **Compression Ratio and Bit Fields and Bit-Field Groupings:** Variation of compression ratio across instruction address bit-fields – Higher order bit fields show best compression.

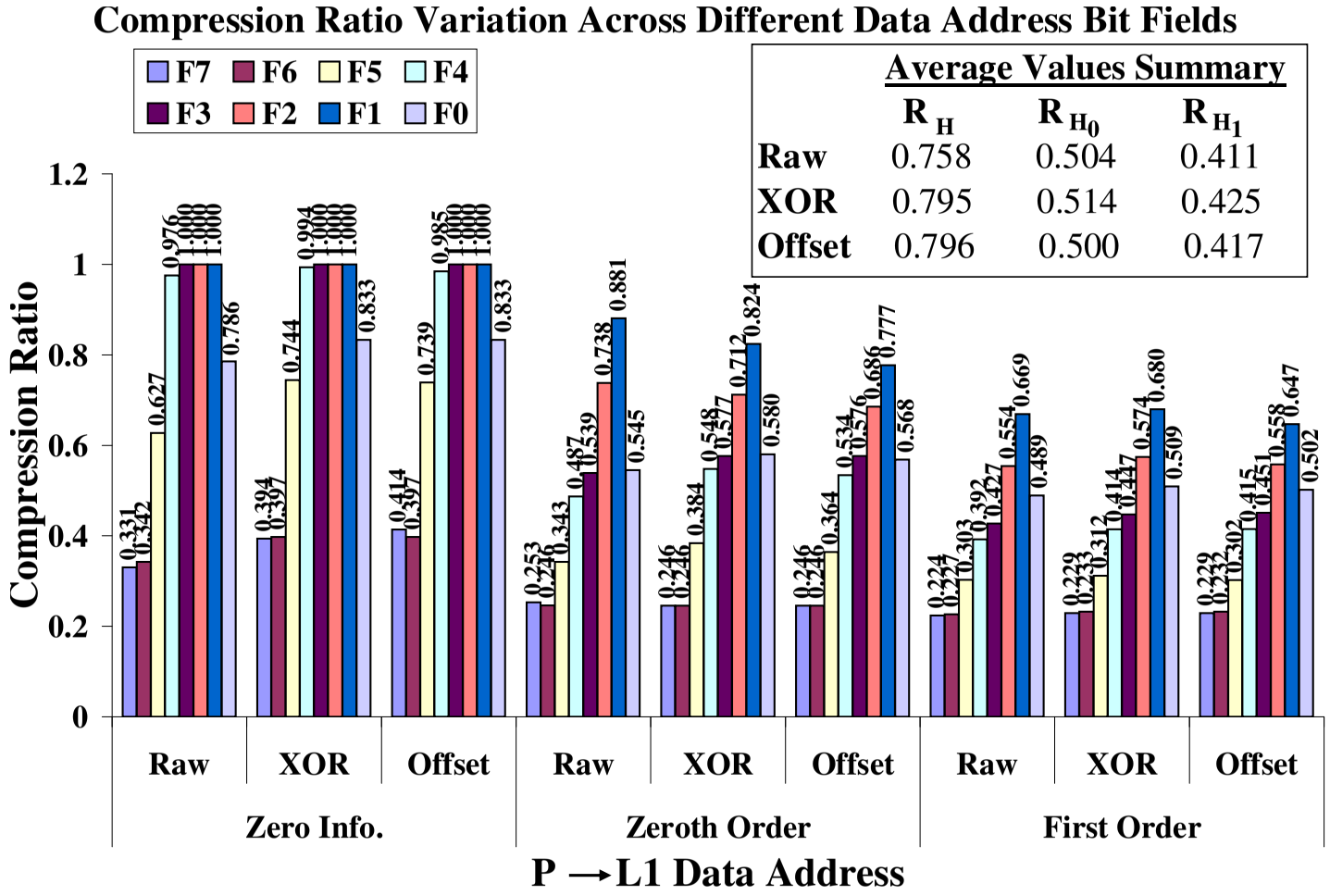


Figure 2.10: **Compression Ratio and Bit Fields and Bit-Field Groupings:** Variation of compression ratio across data address bit-fields – Higher order bit fields show best compression.

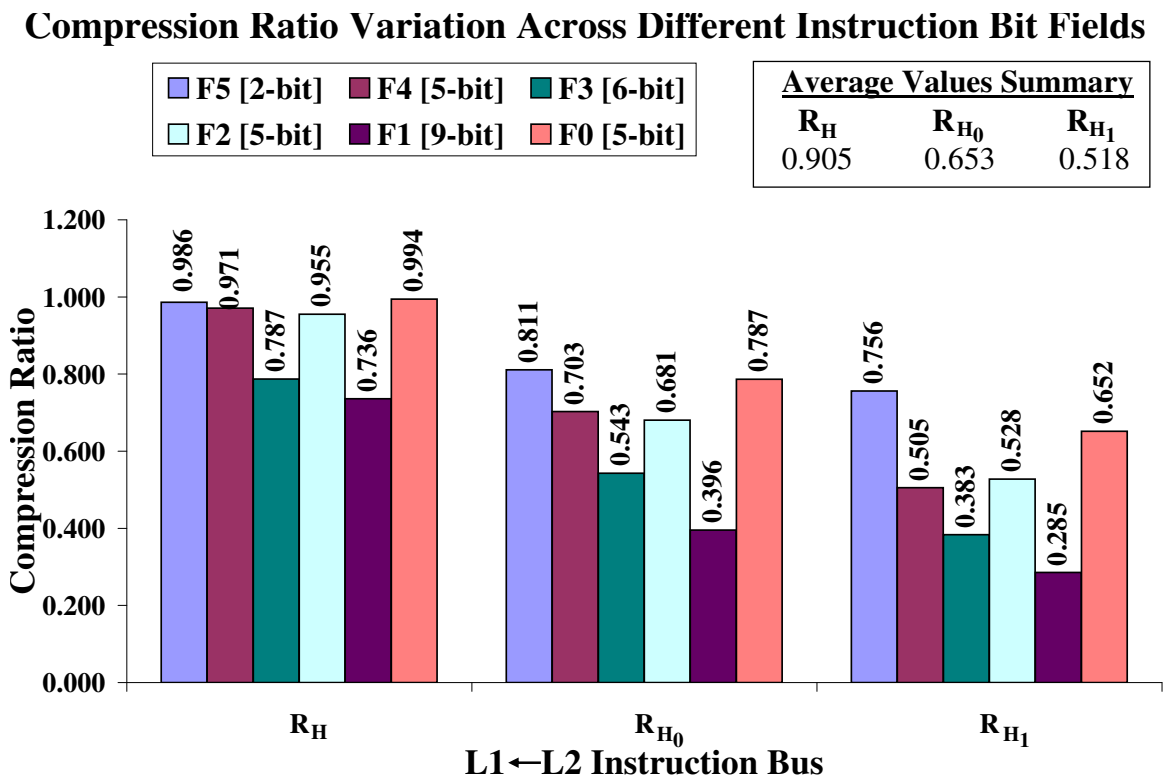


Figure 2.11: **Compression Ratio and Bit Fields and Bit-Field Groupings:** Variation of compression ratio across instruction bit-fields.

For addresses only, we considered original, XOR-encoded, and offset-encoded values for compression separately. We observe the following from the results shown in Fig. 2.14 and Fig. 2.15. In general, for any type of information the more the number of bits in the higher order field, the better overall compression ratio we get. When we consider the whole word as a symbol (G5 for addresses and G4 for others), the best compression ratio was obtained. In the case of instruction addresses, we find that XOR-encoded and offset-encoded values, in most cases, perform worse than original values for zero-info and first-order compression. However, for zeroth-order compression, these perform substantially better than original values. This is because the same XOR or offset values repeat for different combinations of

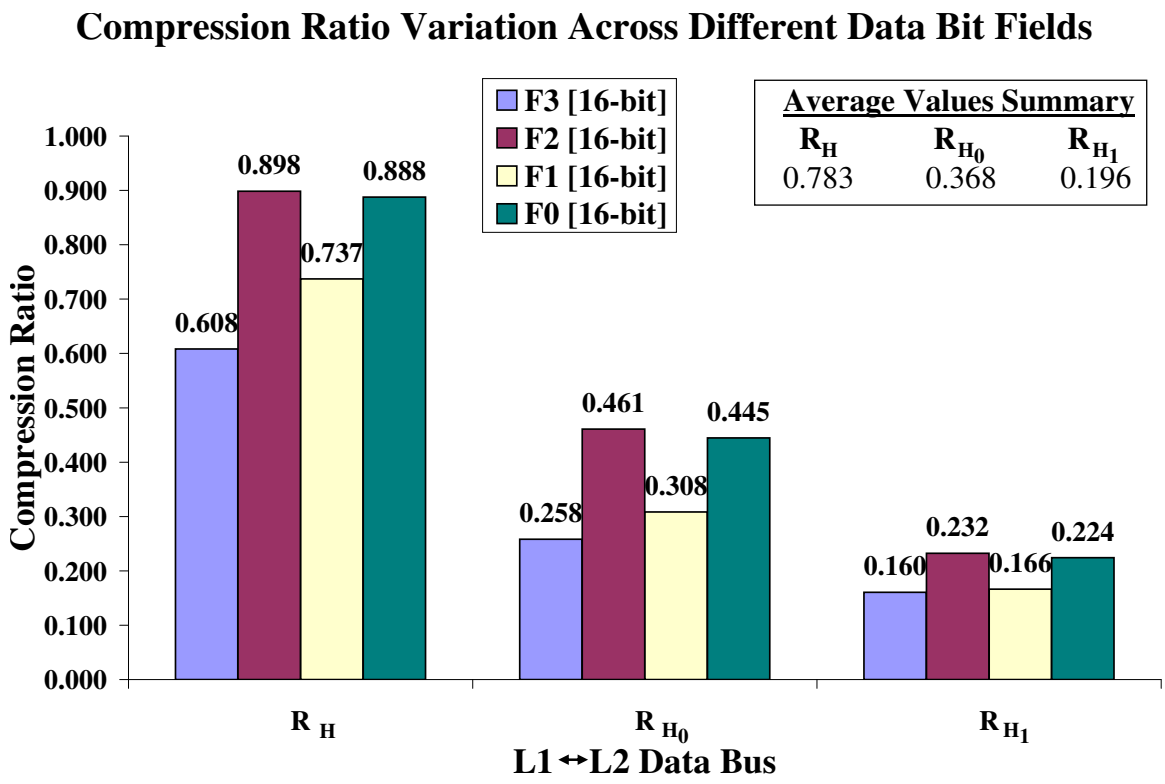


Figure 2.12: **Compression Ratio and Bit Fields and Bit-Field Groupings:** Variation of compression ratio across data bit-fields.

original addresses, thus resulting in higher zeroth-order compression.

2.13 Compression Ratio and Power Savings for Different Workloads

Results for experiments reported in previous sub-sections were averaged over all benchmarks. In this experiment, we compare the compression potential and power savings due to compression of different workloads: integer, floating-point, and embedded. The results of this experiment are shown in Fig. 2.16 and Fig. 2.17 for SPEC CPU2000 and MediaBench programs, respectively.

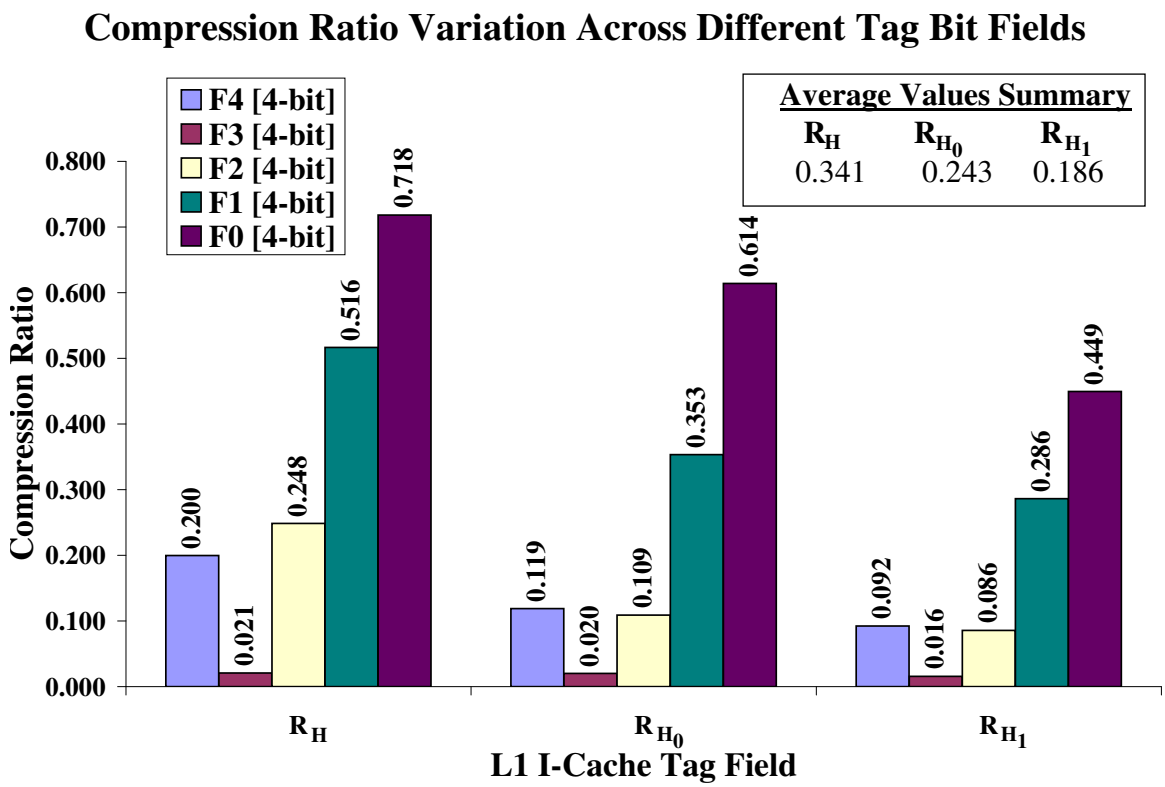


Figure 2.13: **Compression Ratio and Bit Fields and Bit-Field Groupings:** Variation of compression ratio across tag bit-fields.

The following observations can be made for desktop/workstation class workloads represented by the SPEC CPU2000 benchmark programs. As seen earlier in Sec. 2.8, for this type of workload, data in floating-point registers are more compressible than data in integer registers. For program instructions (stored in I-cache data field and main memory and transmitted on instruction bus) and addresses (in I-cache tag field and instruction address bus), we observe that the information for the FP application class is more compressible than the INT application class. We also see that the FP data sent over the data bus is more compressible than the INT data sent over the same bus. This may be because the FP data blocks sent from L2 to L1 (in the event of an L1 D-cache miss) may contain many unused FP words that are

Effect of Different Bit-Field Groupings on Compression Ratio

Average Values Summary

		R_H			R_{H_0}			R_{H_1}		
		Original	XOR	Offset	Original	XOR	Offset	Original	XOR	Offset
G1:	4|4|4|4|4|4|4|4	0.629	0.670	0.539	0.611	0.652	0.660	0.613	0.655	0.625
G2:	8|4|4|4|4|4|4	0.517	0.530	0.591	0.443	0.394	0.435	0.450	0.317	0.150
G3:	12|8|4|4	0.435	0.303	0.275	0.414	0.293	0.261	0.343	0.230	0.246
G4:	16|8|4	0.273	0.172	0.182	0.240	0.258	0.074	0.226	0.245	0.234
G5:	32	0.211	0.228	0.219	0.137	0.158	0.199	0.046	0.074	0.129

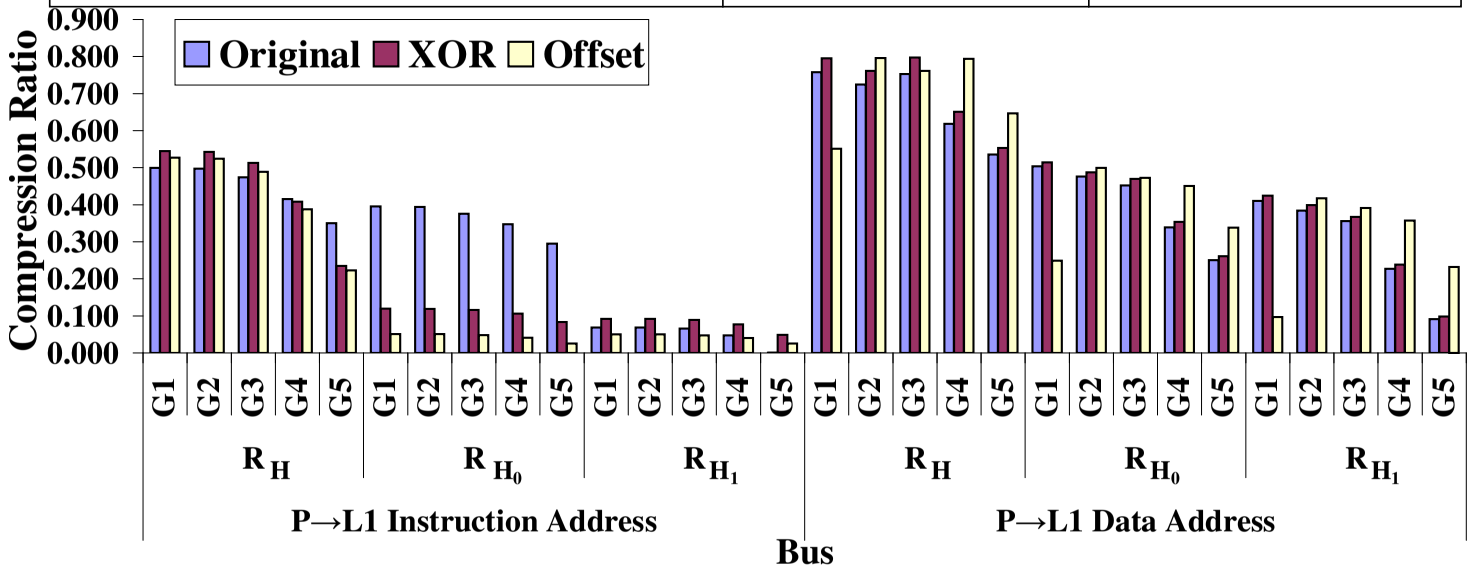


Figure 2.14: **Compression Ratio and Bit Fields and Bit-Field Groupings:** Variation of compression ratio across different instruction and data address bit-field groupings.

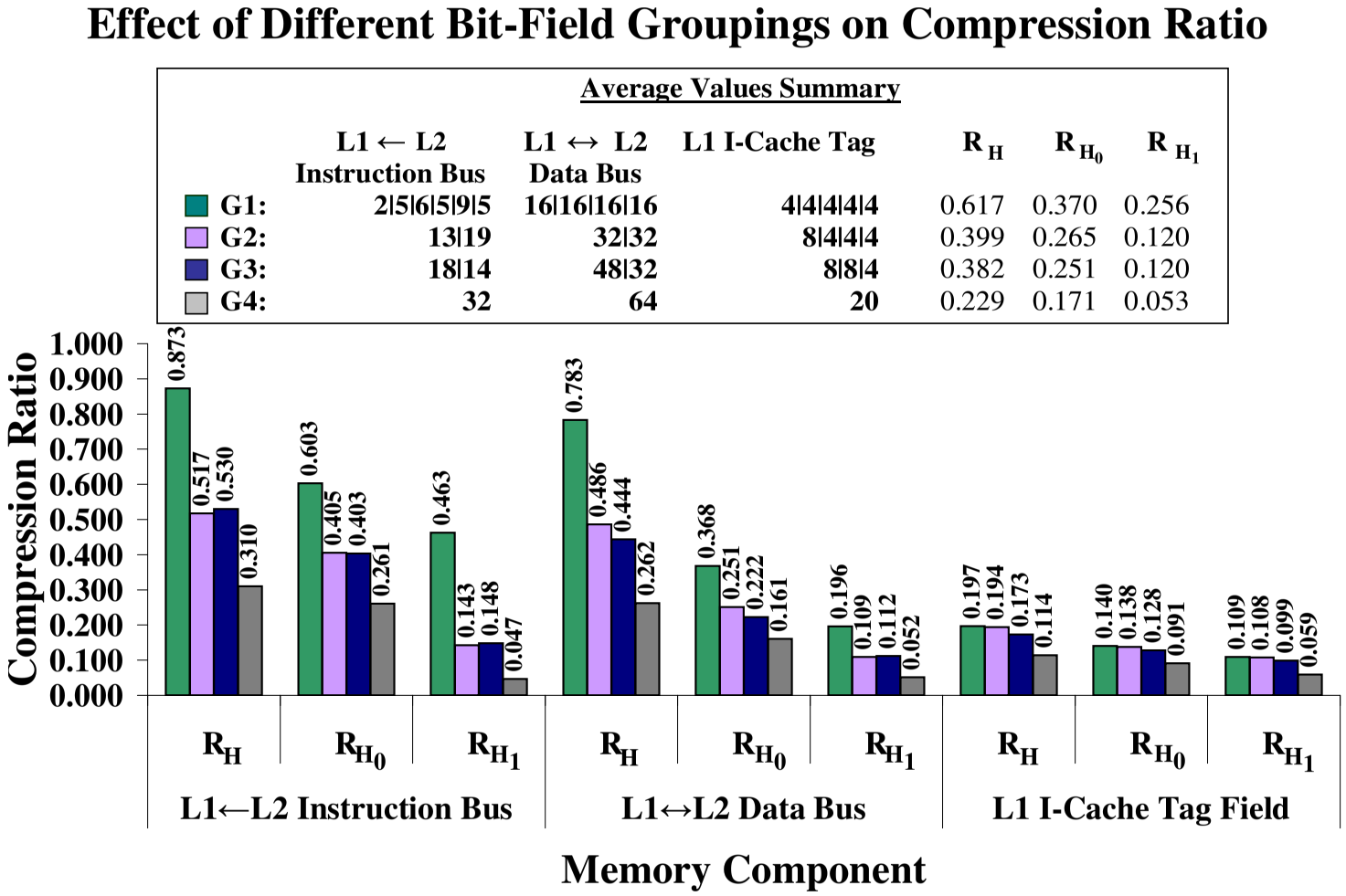


Figure 2.15: **Compression Ratio and Bit Fields and Bit-Field Groupings:** Variation of compression ratio across different instruction, data, and tag bit-field groupings.

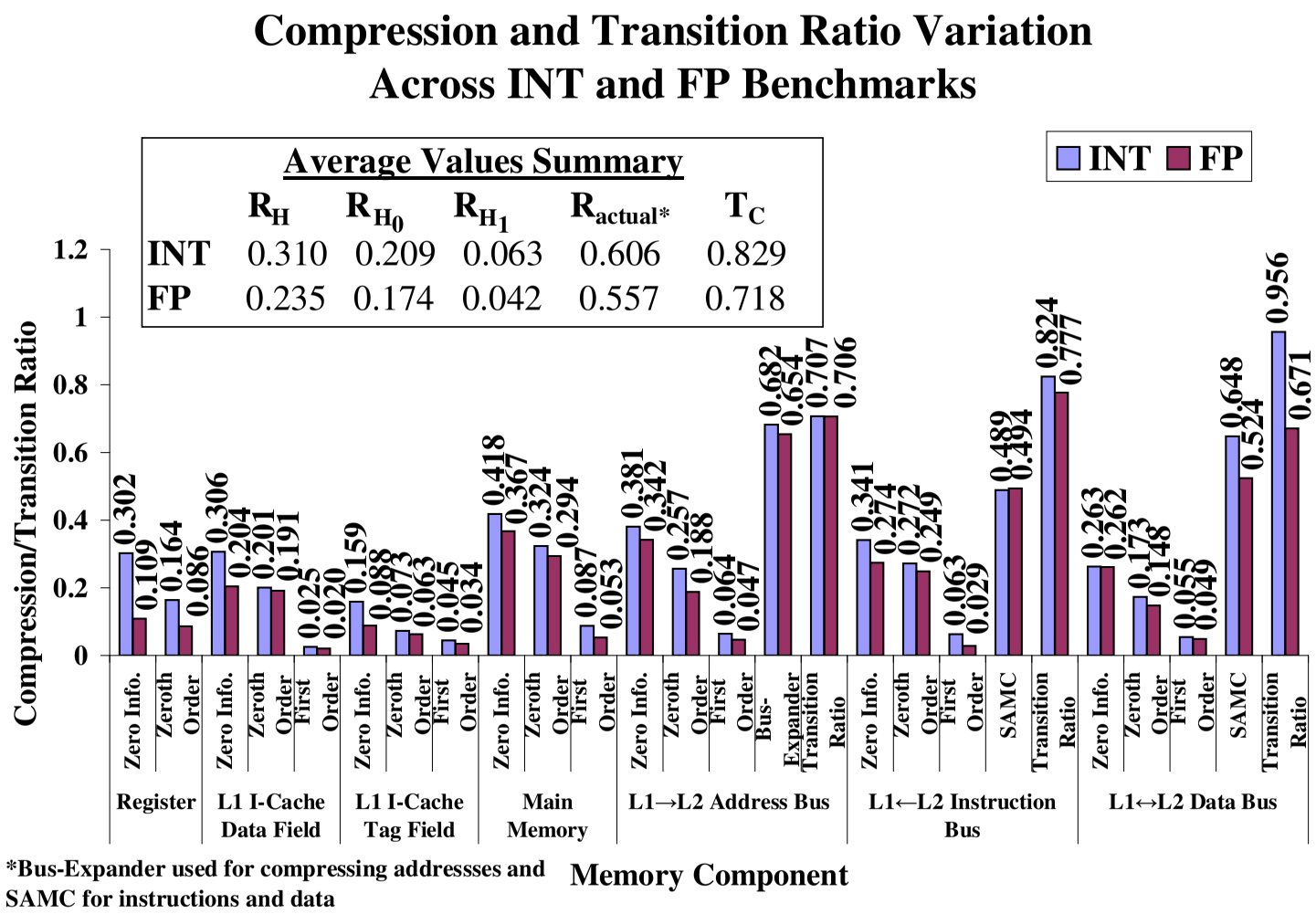


Figure 2.16: **Application Class Analysis:** Desktop/workstation class workloads (SPEC CPU2000 INT and FP programs).

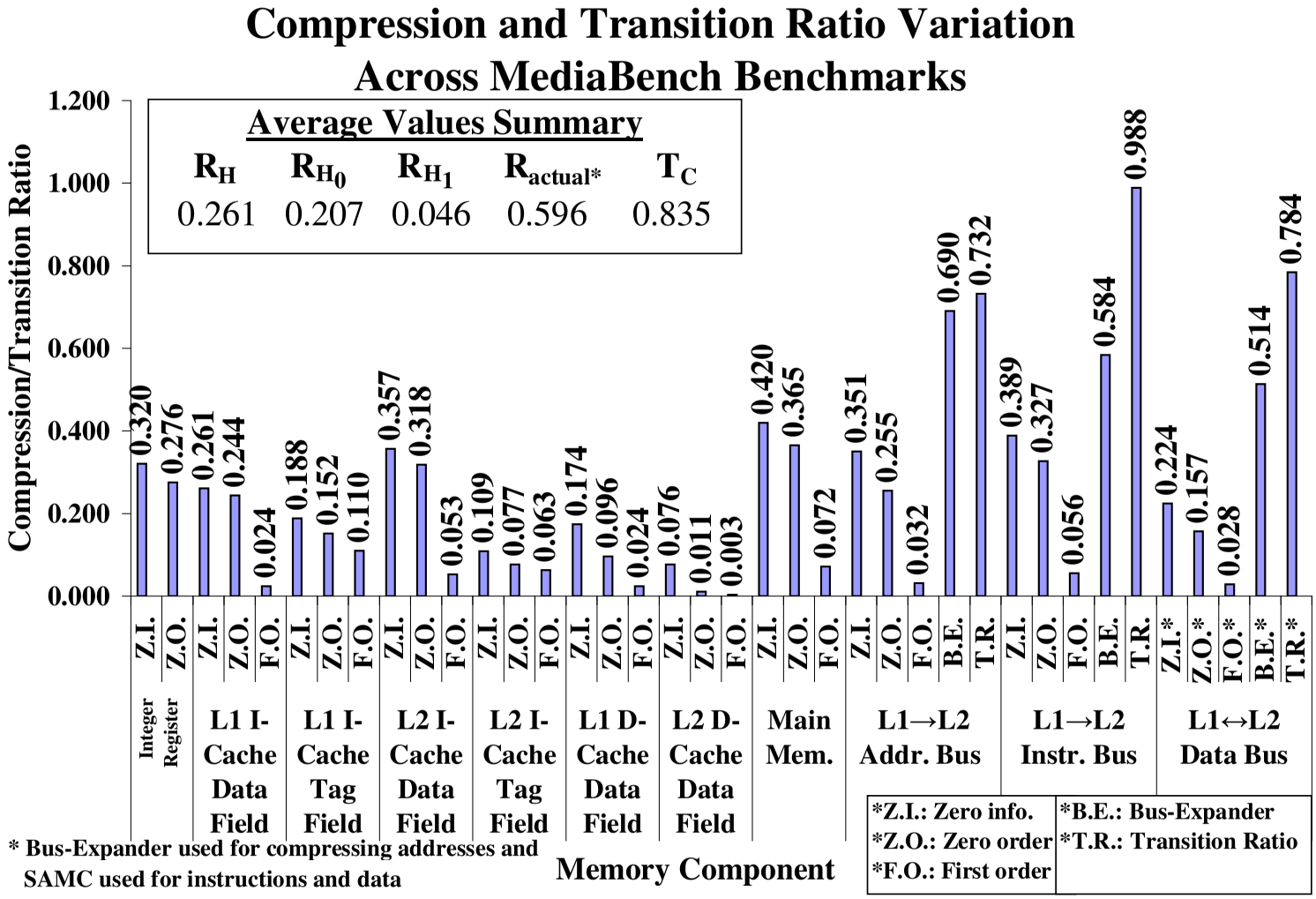


Figure 2.17: **Application Class Analysis:** Embedded workloads (MediaBench programs).

set to zero giving rise to redundancy of information. We also observe that for communication components, FP programs give better power savings than INT programs. For embedded workloads, represented by MediaBench programs, compressibilities are somewhat worse than both integer and floating-point programs.

2.14 Compression Ratio and Degree of Specialization

In this experiment, we investigate how varying degrees of specialization of the compression scheme affect compression ratio. We set up five different types of specialization as mentioned in Section 2.1.4. In the *benchmark-specific* architecture, the compression scheme is specific to each benchmark, but same for all blocks and memory components. For this, symbol statistics used for compression of any trace are determined by analyzing symbols from all memory components. In the *application-class-specific* case, symbol statistics for various components for a subset of benchmarks, the sample-benchmarks, for each application class (INT or FP) are determined and then these statistics are used to compress components for the remaining test-benchmarks in the same application class. To limit the simulation time and memory required for this study, we limited the sample size used for trace collection to 10M instructions. Here, we show separate results for INT and FP.

We observe from results in Fig. 2.18 that with the degree of specialization decreasing, the compression ratio deteriorates. But we observe that compressibility with a general compression architecture is slightly better than an application-class-specific architecture although the former is less specialized than the latter. The general case that we considered here very sim-

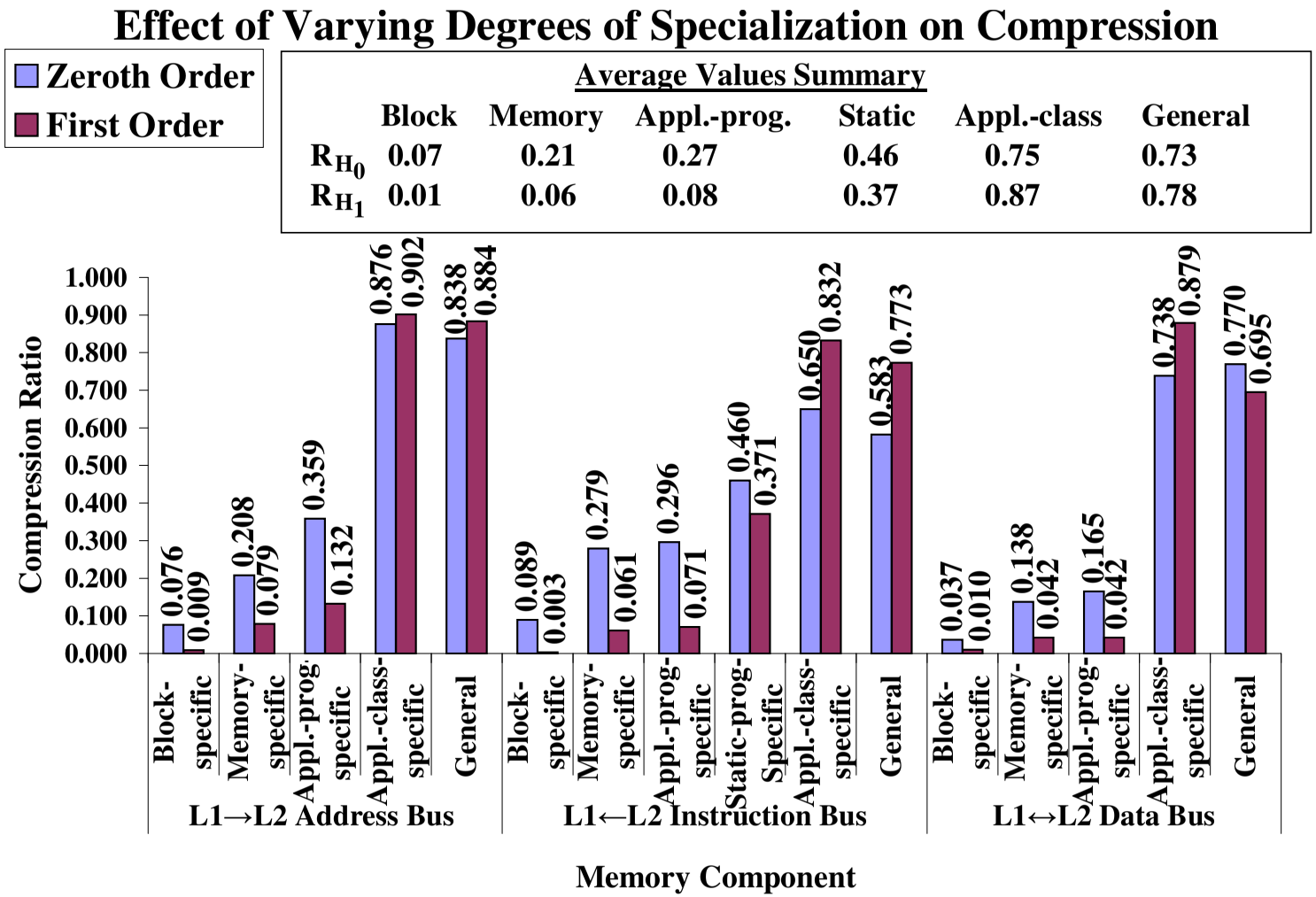


Figure 2.18: **Degree of Specialization Analysis:** Compression ratio variation with degree of specialization.

ilar to the application-specific-class and the only difference is that it draws statistics from all application classes combined. Since the number of distinct application classes considered in our analysis is only two (INT and FP classes–MediaBench programs can be considered to in the INT class), the general case does not result in worse compression than the application-specific class. For the first four cases, first order Markov performs better than zeroth order Markov. But in the application-class-specific case, it is the opposite. This is because for symbols that occur in both test benchmarks and sample benchmarks, symbols are compressed according to statistics in sample benchmarks in zeroth order Markov, but if their preceding symbols do not occur in sample benchmarks, the symbol is left uncompressed in first order Markov and this results in worse compression for first order Markov.

2.15 Power Savings Due to Compression, Encoding, and Both Combined

Some experiments above demonstrated that power saving can be achieved with compression alone. We wanted to investigate if bus encoding, compression, or both applied together decrease power consumption further. We conducted experiments for the three cases and the results are shown in Fig. 2.19 we found that by using compression and encoding together, we could achieve the best power saving. In fact, on the average, compared to the reduction in transitions due to encoding alone, compression reduces transitions further by about 9% and compression followed by encoding reduces transitions by 17%. Thus, a scheme that combines both compression and encoding, can provide excellent benefits in terms of energy

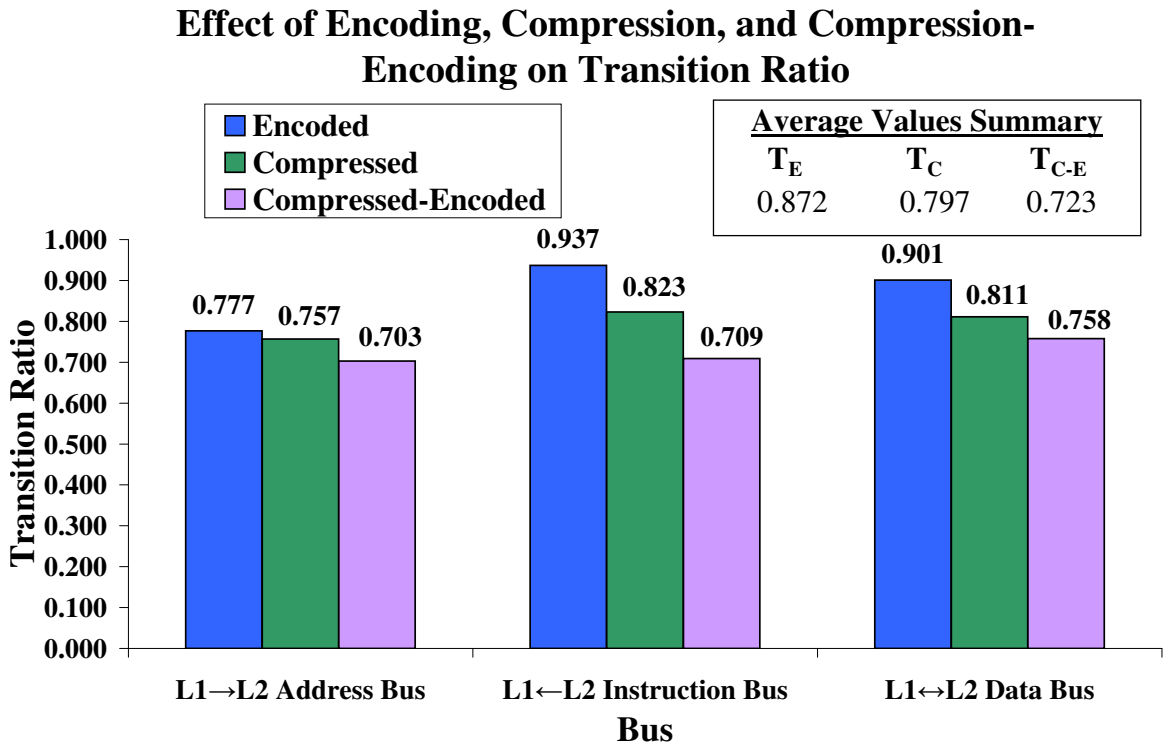


Figure 2.19: **Communication Component Analysis Considering Bus Encoding and Compression:** The extent of power saving due to encoding, compression, and compression and encoding combined. Compression followed by encoding shows best results.

efficiency.

The above result is reiterated in another experiment where we studied the effect of information content on the power consumption of a particular trace. To study how different amounts of information content affect the power savings in a trace, we grouped all traces that we used (address, instruction, and data traces) in different groups according to their first order compression ratio (information content). For example, traces with compression ratios in the range (0, 0.1] were put in one group, those in (0.1, 0.2] in another, and so on until the last group with range (0.9, 1.0]. We used first order compression ratio since it has the lowest value for all traces and hence it represents the lower bound for compression. After

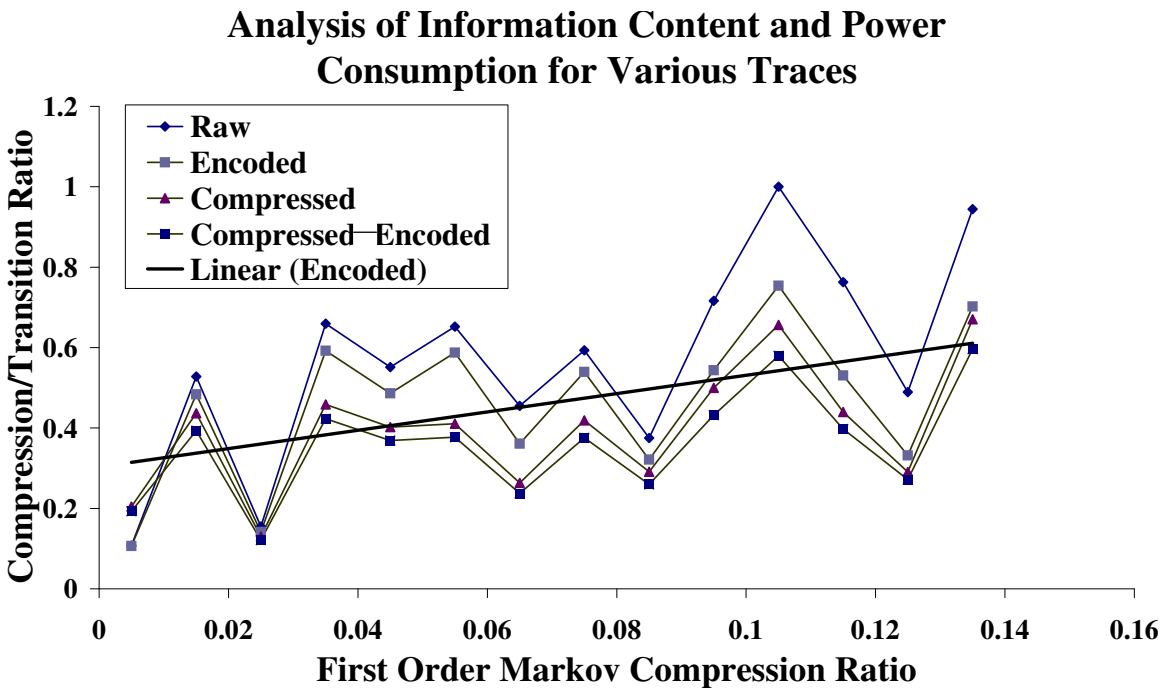


Figure 2.20: **Communication Component Analysis Considering Bus Encoding and Compression:** The effect of information content of a trace on its power consumption.

grouping the traces, we calculated the average transition ratio for each group (total number of transitions in all traces in a group divided by the number of traces in the group) for the original, compressed, encoded, and compressed+encoded versions of the traces in the group. We did this for all traces in all groups and normalized the number of transitions with respect to the trace that had the maximum number of transitions. We plotted this normalized average transitions against the mean compression ratio of a group and the result is shown in Fig. 2.20. It shows that for a given trace, the number of transitions increase with information content, although, for a given information content (compression ratio), the compressed+encoded and compressed traces cause fewer transitions.

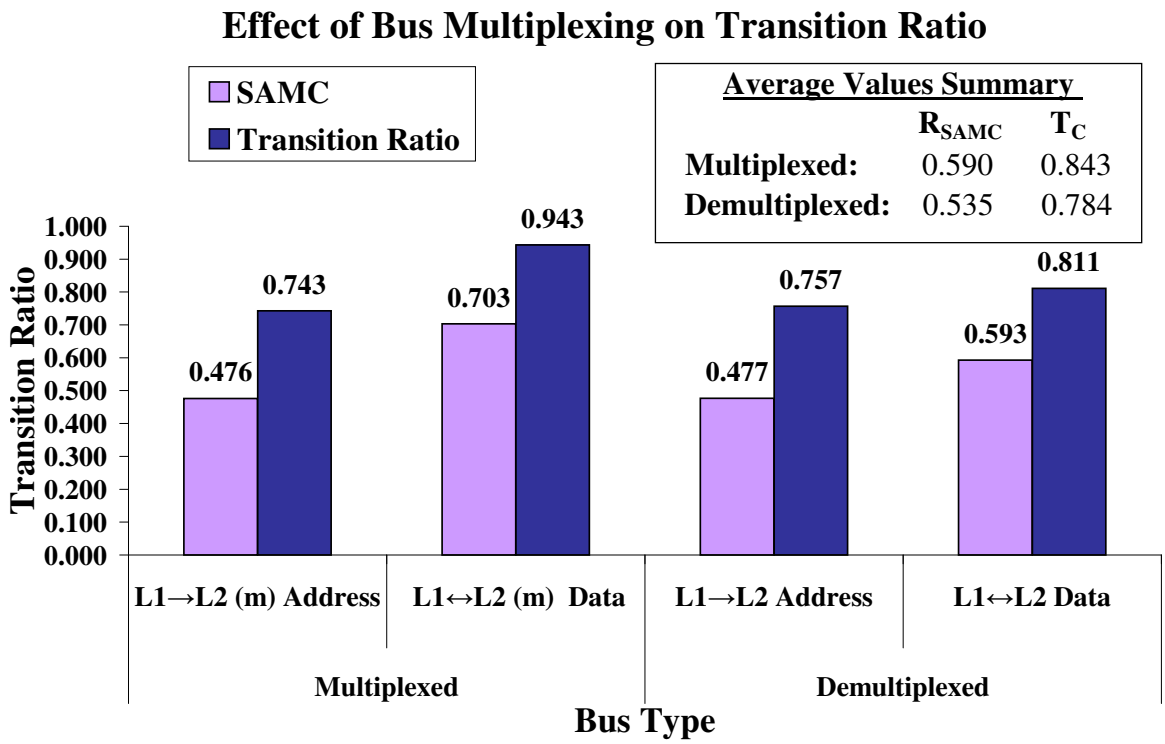


Figure 2.21: **Compression and Transition Ratio Variation with Multiplexed Traffic.**

2.16 Power Savings and Bus Multiplexing

The default bus in our experiments was the demultiplexed bus, and so we also wanted to know how multiplexing affects power consumption. As mentioned earlier, a multiplexed address bus means that both instruction and data addresses are carried on the same bus. Similarly, a multiplexed data bus means that both instructions and data are carried on the same bus. We compared multiplexed and demultiplexed address and data buses and obtained results as shown in Fig. 2.21. While multiplexing an address bus slightly improves both the address compression ratio and power savings, it degrades both in a data bus by a non-negligible amount. This shows that there is sufficient redundancy in multiplexed address streams whereas it is not true for combined data/instruction streams. For data/instruction

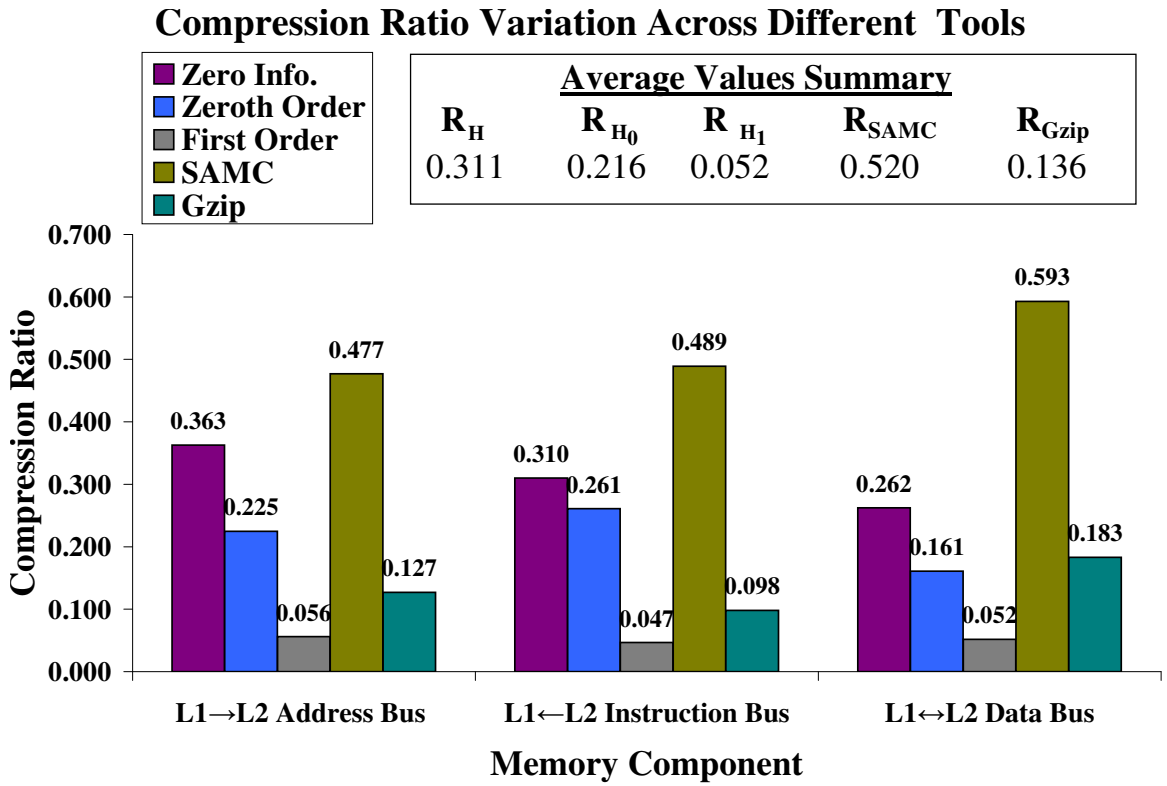


Figure 2.22: **Compression Ratio Variation Across Different Compression Measures and Tools.**

buses, the degree of specialization of the compression scheme on demultiplexed bus is higher than multiplexed bus. On demultiplexed bus compression is specific to each trace itself (instruction, data from L1 to L2, data from L2 to L1, etc.) but on the multiplexed bus, the compression scheme is used for all content on the bus, consisting of instruction and data traffic (both directions) on the bus. This also accounts for lesser compression and power savings on the demultiplexed data/instruction bus. Thus, in spite of multiplexed traffic on address buses, benefits can be obtained but the same is not true for data buses.

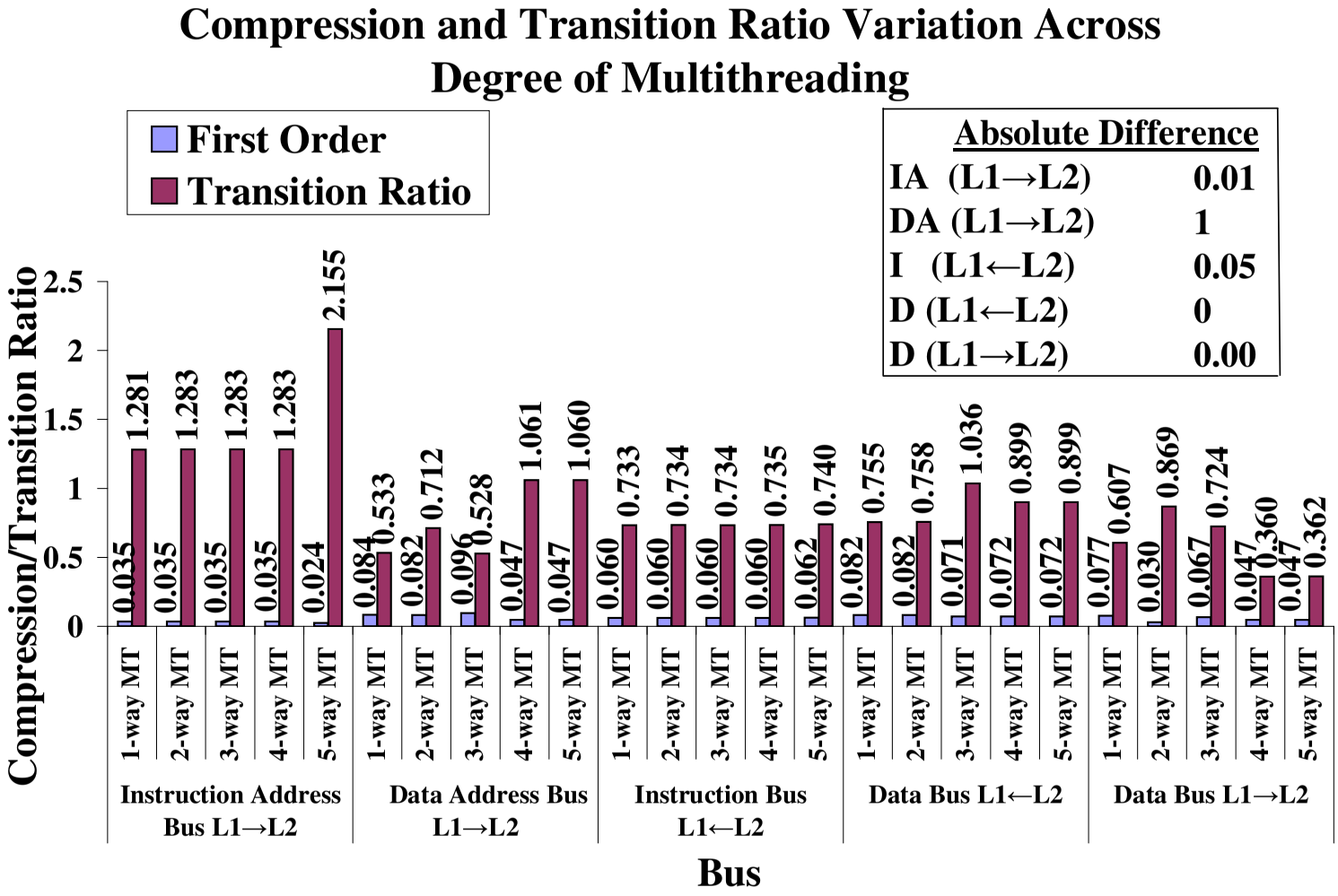


Figure 2.23: **Compression Ratio Variation with the Degree of Multithreading.**

2.17 Compression Ratio and Analysis Tool

SAMC, an arithmetic compression scheme, does not approach the entropy bound, but provides a decent compression ratio of 0.48–0.59 as shown in Fig. 2.22. Among available compression tools, SAMC performs much worse than the commonly used text compression utility *Gzip*, that uses dictionary compression methods. It is also noticeable that there is a wide gap (almost an order of magnitude) between the theoretically achievable compression bound (zeroth and first order entropies) and that achieved by existing compression techniques such as SAMC or Gzip.

2.18 Compression Ratio and Multithreaded Execution

To observe how multithreaded execution affects compression ratios of programs when a shared (address, instruction, or data) bus is used for different threads, we simulated k-way multithreading by obtaining address, instruction, or data traces from k benchmarks and creating a single trace (address, instruction, or data) by ordering them according to the timestamps, i.e., one cumulative trace is created for each of address, instruction, and data. To limit memory required for this analysis, we limited the sample size used for trace collection to 10M instructions. In the case of addresses, we analyze address offset traces because they have more redundancy in general. For any memory system component, it is expected that because of intermingling of traffic from different threads, more transitions will occur. The results shown in Fig. 2.23 suggest that this is somewhat true, although, transitions often do not increase by much when the degree of multithreading is increased from one to five.

Multithreading does not seem to have a perceptible impact on first order compression ratios.

2.19 Conclusions

In this chapter, we presented a comprehensive analysis of all three primary types of information (addresses, instructions, and data) stored and transmitted by the storage and communication components, respectively, at various levels of the memory system hierarchy. The analysis was done in terms of the compression ratio possible, which in turn reflects the amount of performance and to some extent cost improvements attainable using compression. Our analysis was done on SPEC CPU2000 integer and floating-point benchmarks. We have shown that a substantial amount of information redundancy exists in every component of the memory system, such as registers, tag and data fields of caches, main memory (storage components) and also in address, instruction, and data buses (communication components).

Some important results from our analysis are mentioned below. We observed that information stored in the memory system can be compressed to at least 39% of their original size with ideal zero-information compression schemes and to about 31% with ideal zeroth-order compression schemes. Information transmitted in the memory system through buses was found to be more compressible on the average for similar schemes. We found that by compressing tag and data fields, cache access times can be reduced by about 41%(29%) and power consumption by about 66%(44%) on the average for L1 (L2) levels w.r.t. normal uncompressed caches with the same effective capacity. Also, both tag and data areas of caches can be substantially reduced by compression. Other conclusions drawn from our

analysis are as follows: (1) Among storage components, L1 cache was slightly more compressible compared to L2 cache and cache size and block size affected compression ratios, (2) Among communication components, the level of the memory hierarchy where the component is present, different bit fields, and bit-field groupings play a part in determining the amount of compression that is possible, and (3) Compression ratio also depends on the degree of specialization of the compression scheme. We also studied the compressibility of original, XOR, and offset instruction and data address traces, the effect of application class, encoding and multiplexing, analysis tool, static vs. adaptive/dynamic compression, multithreading, and the relationship between information content, compression ratio, and power consumption.

Next, in Chapters 3-5, we present our work on nanometer-scale address bus compression to improve cost, power consumption, and performance by exploiting temporal, spatial, and energy redundancies.

Chapter 3

Hardware-Only Compression of Underutilized Address Buses

3.1 Introduction

Higher instruction issue and clock rates and larger address spaces in modern processors and systems-on-chip (SoCs) necessitate more *communication components* (address, instruction, and data buses and associated hardware like I/O buffers, pads, and pins), which contribute to increased area/cost and power consumption. On-chip buses scale relatively poorly in size compared to logic and this results in more area, which causes more individual wire capacitance and hence power consumption. Further, due to tighter spacing between higher aspect-ratio wires of buses in nanometer regime, coupling capacitance effects become pronounced, resulting in even more power consumption. Increasing the number of pins and

off-chip buses is difficult because it is limited by the surface area of the chip, whereas the amount of logic circuitry grows as the volume of the die. Also, off-chip buses have orders of magnitude more capacitance than on-chip circuit nodes and this exacerbates the power dissipation problem. Therefore, it is important to minimize area/cost and power consumption due to communication components.

This chapter focuses on the area/cost and power consumption of address buses. Previous work in reducing cost relies on using narrow buses to transmit compressed addresses [20, 11]. Bus encoding schemes, on the other hand, strive to reduce power consumption by transmitting encoded (uncompressed) addresses that cause fewer self-transitions [8] and fewer coupling-transitions [77, 34]. For narrow buses, two particularly relevant encoding schemes are the Pyramid code [9] for DRAM address buses and the BITS and ABITS codes [55]. We propose *hardware-only compression* (HOC) of underutilized address buses in which address information is transmitted on a narrow bus over multiple cycles to reduce area/cost and improve bus utilization, and possibly also lower power consumption. Due to its simplicity, HOC is expected to have lower area and power consumption overheads at the and receiving ends compared to address compression and encoding methods. We present hardware designs and new encoding methods for HOC and analyze in detail its performance, power, and cost implications through realistic execution-driven simulations.

The rest of the chapter is organized as follows. In Section 3.2, we describe in detail our HOC strategy, including its benefits and feasibility, and hardware design. In the following section, we discuss how area savings from HOC can be exploited via appropriate wire lay-

outs. In Section 3.4, we describe our simulation setup and metrics and in Sections 3.5 - 3.9 present various strategies for for improving energy efficiency of HOC and results from our simulations. Finally, Section 3.10 concludes the chapter.

3.2 Hardware-Only Compression

In this section, we describe HOC in detail, including its benefits, overheads, hardware design, and novel encoding schemes.

3.2.1 Overview

In hardware-only compression, so called because only the bus hardware, but not the information transmitted on the bus, is compressed, a narrow bus is used to transmit information over multiple cycles. HOC is applied to underutilized buses to save cost and improve bus utilization and possibly lower power consumption. Buses at different levels of the memory system are underutilized because of the following reasons. Memory referencing instructions (loads and stores) that cause data addresses to be issued from the processor constitute only about 41% in RISC processors [26]. As a result, the utilization of processor-to-level1 (P→L1) data address (DA) buses can be expected to be low. Correspondingly, the utilization of DA, instruction address (IA), data, and instruction buses to higher levels of cache or DRAM can be expected to be lower since caches filter out most of the instruction or data misses. A limited form of HOC to reduce the number of pins on DRAM is used by multiplexing row and column addresses.

3.2.2 Benefits

Some of the benefits that can be obtained directly or indirectly by using HOC are as follows. In the case of on-chip buses, HOC results in less area, and when applied to off-chip buses, the number of I/O pads and pins reduces, all of which lead to lower die and packaging costs. Due to the smaller area, capacitance and thus power consumption may reduce. Further, by using area no more than a normal, uncompressed bus, a narrow bus can: (1) use greater spacing between bus lines, which will reduce coupling capacitance and hence delay, power consumption, and cross talk; and/or (2) use wider wires to reduce resistance and hence delay.

3.2.3 Overheads

Using hardware-only compression entails performance, area, and power consumption overheads. Performance overheads can occur due to two reasons. First, since addresses occur nonuniformly over time, buffering will be required at the sending end and even then these buffers may fill up due to an address burst, necessitating pipeline stalls. Second, since addresses arrive at the receiving end later, cache/memory access is delayed and this delayed fetch may cause a dependent instruction to stall the pipeline. However, modern processors using dynamic scheduling can minimize the occurrence of such stalls by executing instructions out of order. Performance overheads can be mitigated in three ways. First, more buffering can be done at the sending end to avoid buffer-full related stalls. Second, addresses can be transmitted in w-bit groups from the high- to low-order end so that address tag and index fields are received quickly to allow early hit detection in the cache/memory at the receiving end. Third, when bus width is a non-integral fraction of the address width, *address*

concatenation can be used, i.e., during the last cycle of transmission of an address, if there are unused bus lines and if the next address is available, a part of this next address (starting from the high-order end) can be transmitted. To indicate the presence or absence of a concatenated address in this last cycle, a *concatenation bit* can be used.

There will be area and power consumption overheads due to the additional logic required to transmit and receive addresses in parts, for extra address buffers at the sending end, and for supporting address concatenation if used. But these overheads will not be much compared to the savings obtained by compressing long on-chip buses and off-chip buses. Address partitioning is not expected to take any additional cycles (although transmission of the address itself will take multiple cycles depending upon bus width). Due to the misaligned transfer of addresses in hardware-only compression, there will be some extra transitions and power consumption. To mitigate this problem, address offsets (offset w.r.t. previous address: mostly small magnitude) and address XORs (XOR w.r.t. previous address: mostly zeros) can be used. Of course, in these cases, the previous address needs to be saved at the receiving end in order to determine the new address. Finally, since logic scales better in size, speed, and power consumption than long bus wires, with technology improvements, the logic overhead to perform hardware-only compression and decompression will reduce relative to the savings in bus lines.

3.2.4 Hardware design

The compression and decompression hardwares for HOC at the sending and receiving ends are shown in Figs. 3.1 and Figs. 3.2. We propose a hardware structure shown in Figs. 3.1

with multiplexers to split the n-bit original address into $j = \lfloor \frac{n}{b} \rfloor$ partitions and place them on the narrow bus of b lines in j successive cycles. The last portion of the address is either transmitted in the $(j+1)$th cycle, with some of the bus lines unused, or concatenated with the first part of the next address. Conventional barrel or logarithmic shifters can also be used but the overheads in terms of area may be higher. We use the b-bit registers labeled $R_1, R_2, \ldots, R_{j+1}$ to store the shifted partitions at the end of each cycle. These registers (flip-flops) do not add to the overheads of our scheme since bus driving circuitry (based on static or dynamic CMOS logic) already contains flip-flops or latches for each bit-line. It can be noted from the way the hardware is organized that during each clock cycle, the contents of register R_1 are latched on to the bus and at the end of the cycle, the contents of R_2 are selected through the $2:1$ multiplexer and latched into R_1 and so on. The default hardware shown in the figure uses the concatenation mode of operation, where unused lines in the last cycle of transmission are used for the transmission of a part of the next item, if available. The normal mode of operation where concatenation is not used can be enabled by sending an appropriate signal to the control unit. The operation of the decompression hardware is analogous to the compression hardware described above. Portions of the address that arrive are first placed in the register R_{j+1} and then shifted into the next register in subsequent cycles and so on until the first portion is in register R_1. At this time, multiplexers are used to shift the address onto the outgoing lines.

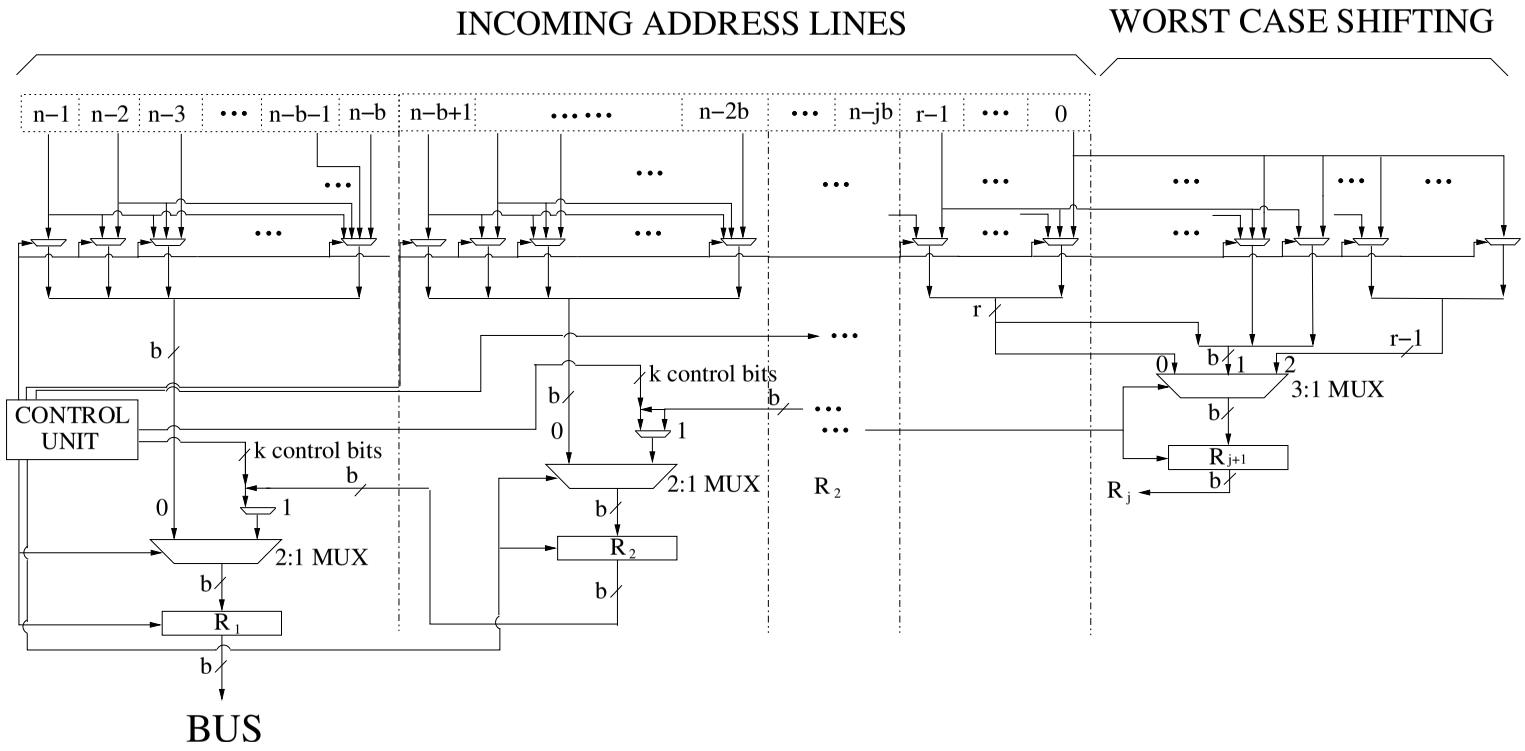


Figure 3.1: **Hardware for HOC:** Compression hardware at sending end.

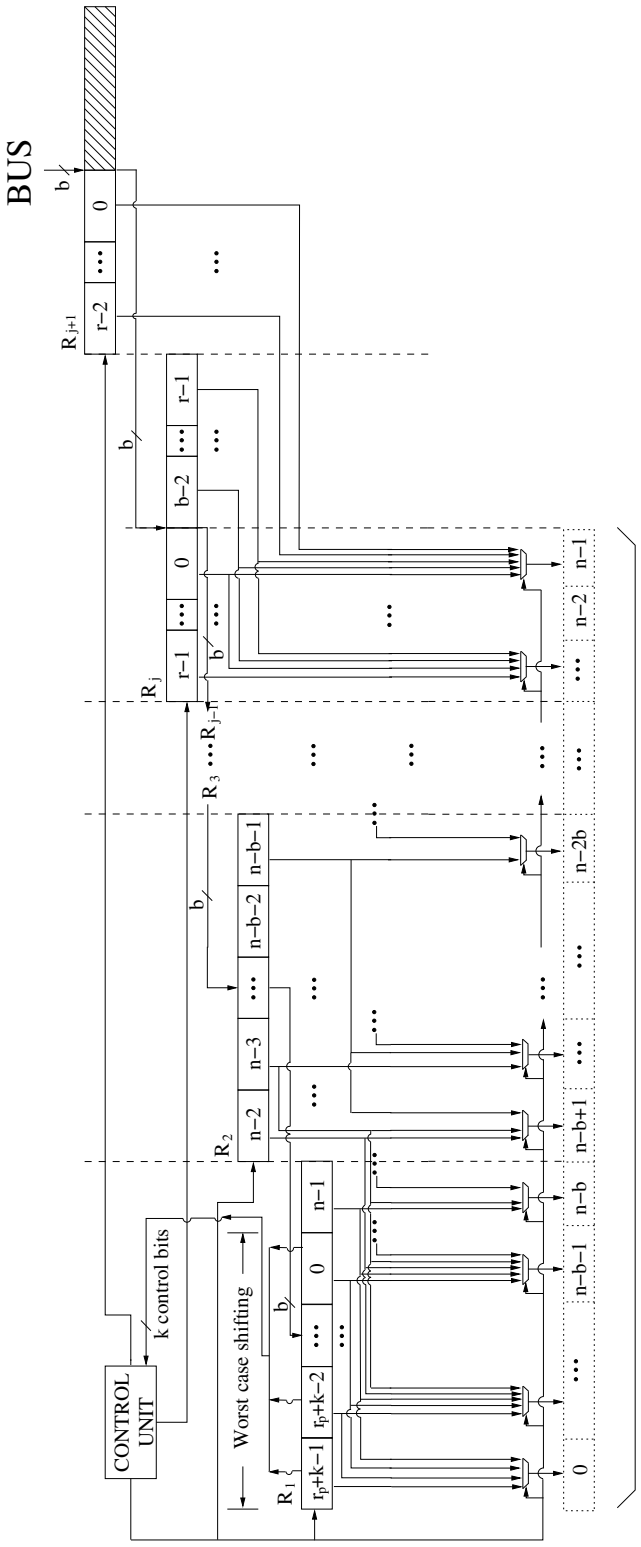


Figure 3.2: **Hardware for HOC:** Decompression hardware at receiving end.

3.3 Wire Layout Optimizations

Using compressed buses grants an extra degree of freedom while performing global wire routing for high-performance designs. Common optimizations like *net shielding* (inserting power or ground wires on both sides to protect a wire on the critical path from inter-wire couplings) and *soft spacing* (a technique that automatically maximizes spacing between tightly packed wires within given area constraints) are greatly facilitated by using compressed buses. Such optimizations go a long way in achieving signal integrity and timing closure in current nanometer designs [46]. Although these techniques have been used in the VLSI design community for a long time, our work is the first to examine their implications in the context of compressed buses.

3.3.1 Wire spacing

In this simple scheme, the wires in a compressed bus can be spaced further apart while maintaining the area footprint smaller or equal to the original bus to minimize the wire delay and coupling capacitance. Wire spacing involves no additional cost overheads and it will reduce wire delay and bus energies since inter-wire capacitances are inversely proportional to spacing.

3.4 Simulation Setup

In this section, we first describe our simulation environment. Next, we describe how we calculated various metrics like bus utilization, extra cycle penalty, and energy ratios that we used to estimate the performance, power consumption, and cost overheads of our schemes.

3.4.1 Simulation environment

Processor Core	
Clock rate	600MHz
Issue width	6 (4 integer and 2 floating point)
LSQ	32 entries each
Memory System	
P↔L1 bus	Non-pipelined; 64-bit data, 128-bit instruction, and 44-bit address lines
L1 D-cache	Virtually-indexed physically-tagged (VIPT), 64KB, 2-way set assoc., 64B block size, LRU policy, 3 cycle hit latency, write-back
L1 I-cache	Virtually-indexed virtually-tagged (VIVT), 64KB, 2-way set assoc., 64B block size, LRU policy, 1 cycle hit latency
L1 MAF	8 entries
L1↔L2 bus	Non-pipelined; 128-bit data/instruction lines and 38-bit address lines (21 bits for block index and 17 bits for tag)
L2 cache	Physically-indexed physically-tagged (PIPT), 2MB, direct-mapped, 64B block size, LRU policy, 12 CPU cycles hit latency, write-back policy, operating at 2x CPU clock cycle
L2↔M bus	Non-pipelined; 64-bit data/instruction lines and 38-bit address lines
DRAM	256MB, operating at 2x CPU clock cycle, 96 CPU cycles hit latency
Benchmarks	
CINT2000	gcc, gzip, parser, vpr, twolf, mcf, crafty
CFP2000	applu, swim, wupwise, lucas, art, ammp, equake
Sample	10 million committed instructions after skipping 500 million committed instructions initially.

Table 3.1: **Target System and Benchmarks:** Default configurations for our target system, benchmarks, and sample sizes used in our simulations. LSQ= load/store queue, MAF= miss address file. This target system is broadly based on the Alpha 21264 processor.

3.4.2 Bus utilization

In the first phase of simulation, the traces collected were used to determine the *average bus utilization* (BU) for each bus. The BU for a bus shows the extent to which the bus is utilized on the average. It also points to the amount of HOC possible in the bus. Also, by looking at the BU, it is possible to decide if a particular bus is worth compressing or not. For example, a bus with a BU close to unity is not worth compressing since the performance penalties may

be severe. For any address trace during the sampling window, if r is the number of address references for a particular bus type (P→L1 load data address (LDA), L1→L2 instruction and data address (IDA), or L2→M IDA), then the BU averaged over n benchmarks, denoted as $U_{bus\ type}$, is given by the following relation.

$$U_{\text{bus type}} = \frac{\sum_{i=1}^{n} r_{\text{bus type}}}{\sum_{i=1}^{n} \text{Simulation sample time in CPU cycles}}$$

3.4.3 Performance penalty and wire delay

The extra cycles that a benchmark program running on the modified target system (system with address compression) takes compared to its running time on the default target system (system with no compression) is reported as the performance overhead due to address compression. Since we use an event-driven simulator, our calculation of performance overhead includes any latencies due to pipeline stalls also and not just the latencies due to address transmission. The metric that we use to measure the performance overhead is the *average percentage extra cycle penalty* (ECP) which we define as follows for n benchmarks. Here, t_C is the total time (in terms of processor cycles) for execution of the sample window with address compression/decompression and $t_{original}$ is the total time for execution of the sample window of a benchmark (without HOC).

$$ECP = \frac{\sum_{i=1}^{n} (t_C - t_{original})}{\sum_{i=1}^{n} t_{original}} \times 100$$

To compute the wire delay, we use the Elmore delay formula which gives the delay of a wire routed in the global layer [53]. The metric used to evaluate the wire delay improved

when wire spacing is applied is the *wire delay ratio* (WDR). Here, wd_S is the wire delay for the compressed bus with wire spacing and $wd_{original}$ is the wire delay for the original uncompressed bus.

$$WDR = \frac{wd_S}{wd_{original}}$$

3.4.4 Bus energy model

In on-chip buses, energy is dissipated due to *self-transitions* (transitions on the capacitance between a bus line and the ground plane) and *coupling transitions* (transitions on the capacitance between adjacent bus lines) [59]. The total energy dissipated due to self-transitions can be computed using the following expression: $E_{self} \propto N_{s,edge} \cdot C_{s,edge} + N_{s,middle} \cdot C_{s,middle}$, where $N_{s,edge}$ corresponds to the total number of self-transitions occurring in the two edge wires of a bus, $C_{s,edge}$ is the self-capacitance of an edge wire, $N_{s,middle}$ is the total number of self-transitions occurring in all the non-edge wires, and $C_{s,middle}$ is the self-capacitance of a non-edge wire. Note that $C_{s,edge} > C_{s,middle}$ in current technologies due to the *fringing effect* of the isolated side-wall in each edge wire. The effect of fringing fields are non-negligible in current technologies because wire-height and hence side-wall area is more than wire-width for global and intermediate metal layers where most long buses are routed.

The total energy dissipated due to coupling transitions can be computed using the following expression: $E_{coupling} \propto (N_{charge} + N_{discharge} + 4 \cdot N_{toggle}) \cdot C_c$, where N_{charge} is the total number of *charging coupling transitions* ($00 \rightarrow 01$, $00 \rightarrow 10$, $11 \rightarrow 01$, and $11 \rightarrow 10$), $N_{discharge}$ is the total number of *discharging coupling transitions* ($01 \rightarrow 00$, $10 \rightarrow 00$, $01 \rightarrow$

11, and 10 → 11), N_{toggle} is the total number of *toggle* transitions (01 → 10 and 10 → 01), and C_c is the coupling capacitance between two adjacent lines of the bus. Note that $C_{s,edge}$, $C_{s,middle}$, and C_c are values that depend on technology and the layer of metal being considered. We used values for these parameters obtained using TSMC 0.18μ global wire dimensions and applying formulas used in Berkeley predictive technology models (BPTM) for interconnects [7].

For off-chip buses, only self-transitions need to be considered because inter-wire spacings are large; fringing effects are also negligible since wire-widths are substantially larger than wire-heights. In our simulation results, we plot *average on-chip energy ratio*, $E_{on-chip}$, and *average off-chip energy ratio*, $E_{off-chip}$, instead of absolute energies. These are obtained by summing the compressed bus energies for 14 benchmarks and dividing by the sum of original bus energies for the same set of benchmarks. The average on-chip energy ratio is defined as follows for *n* benchmarks.

$$E_{on-chip} = \frac{\sum_{i=1}^{n} e_{\text{narrow on-chip bus}}}{\sum_{i=1}^{n} e_{\text{original on-chip bus}}}$$

The average off-chip energy ratio is similarly defined.

In the following sections, we describe the experiments that we conducted to show the efficacy of our proposed schemes and discuss the results for each.

3.5 Bus Utilization and Selection of Bus Width

We calculate the average bus utilization for each of the three buses, namely, P→L1 LDA bus, L1→L2 IDA bus, and L2→M IDA bus as mentioned earlier in Sec. 3.4.2. We also calcu-

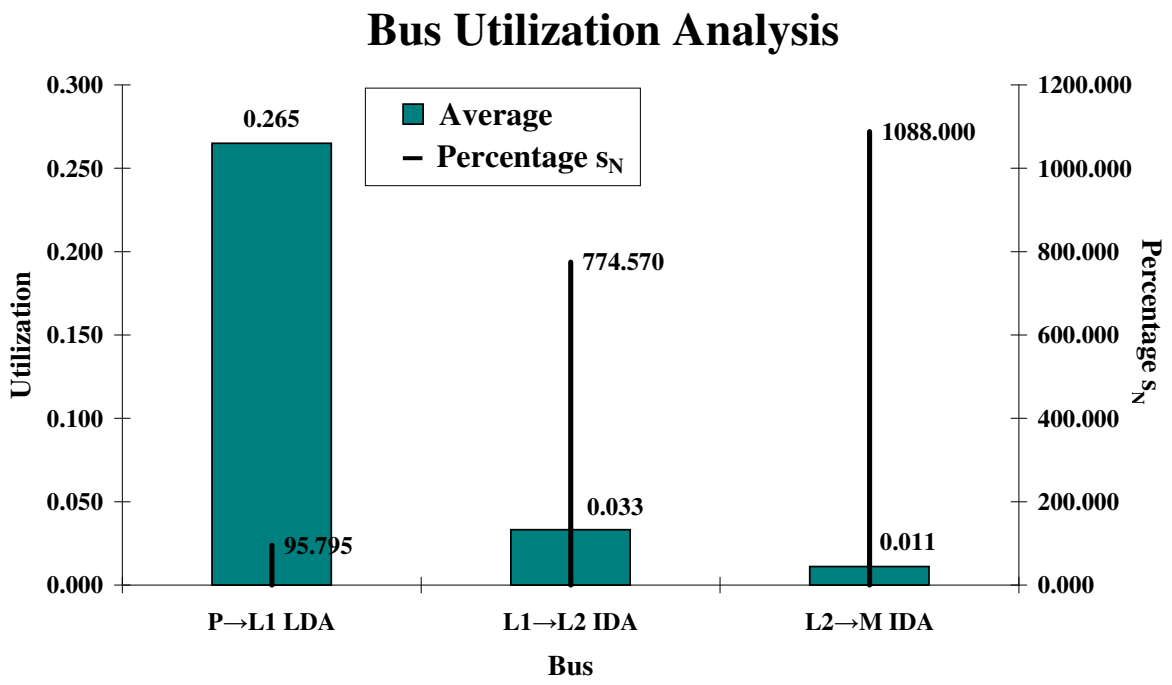


Figure 3.3: **Average Bus Utilization and Percentage Standard Deviation (s_N) Across Different Buses.**

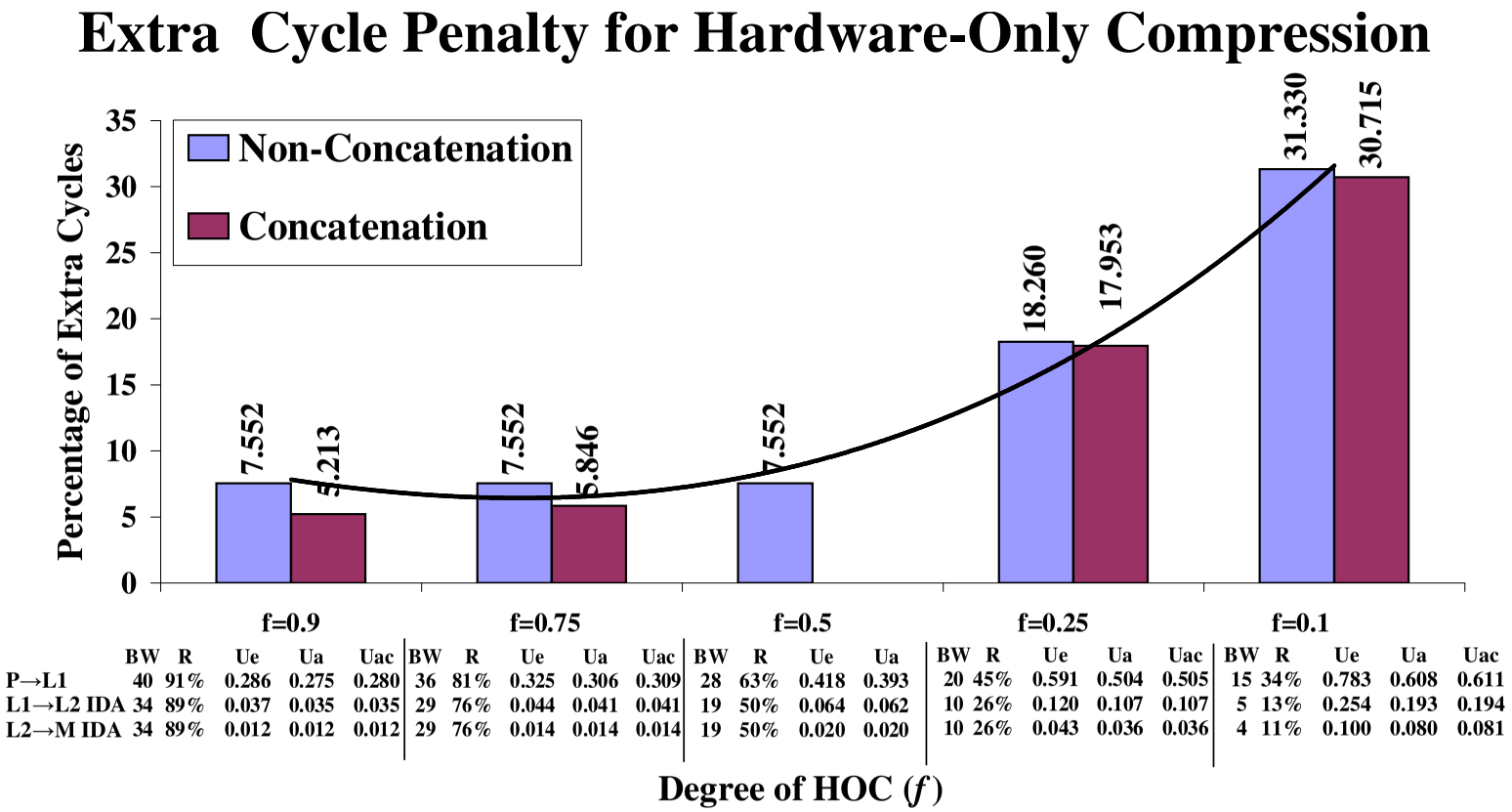
late the percentage standard deviation (s_N) of the bus utilization w.r.t. the average utilization for each benchmark and report this value in the figure. The percentage standard deviation of the utilization is an indicator of the burstiness of the address traffic due to each benchmark program. Using the utilization, $U_{bus\,type}$, for a particular bus obtained from Fig. 3.3, we choose the narrow bus width b for the remainder of our simulations according to the following relation: $n * [U + f \times (1 - U)]$. Here n is the original bus width and f is a parameter called *degree of HOC*, which denotes the portion of the underutilized capacity of the bus that we want to use. Thus, $f = 1$ represents the original bus width (no HOC) and $f = 0$ represents the case where the utilization of the compressed bus will be ideally 100%. The maximum savings in cost can be realized without performance penalty for HOC for this bus

width assuming that references are spaced evenly apart. Intermediate values of f like 0.25 and 0.5 represent different degrees of HOC and hence different cost savings. We observe from the figure that, as expected, as we move away from the processor, buses become more and more underutilized and bursty.

3.6 Performance Overheads

3.6.1 Extra cycle penalty for same degree of HOC across all buses

We first conducted experiments keeping the degree of HOC (f) value same for all buses in the system. Thus, different bus widths were used for buses at different memory levels. From Fig. 3.4, we observe that the performance penalty increases non-linearly as the degree of HOC is increased. The increase follows the trend curve $y = 0.1968x^3 + 0.8605x^2 - 5.3203x + 12.087$. This non-uniform increase may be as a result of the fact that misses at that level are not evenly distributed. For the case $f = 0.1$, we observe that about 31% extra cycles may be needed for the program to complete execution when concatenation is not used. For lower cost savings ($f = 0.9$, $f = 0.75$, and $f = 0.5$), the performance penalties are still substantial (about 7.5%). Another interesting observation is that the performance penalty remains at about 7.5% until almost 50% bus compression ($f = 0.5$) is reached. This is because, for values of f ranging from 1 to 0.5, regardless of the degree of HOC, two cycles are required to transmit each address. Beyond $f = 0.5$, reducing the bus width by a few bits is enough to double the number of cycles needed to transmit the address. This explains the sudden increase in performance penalty after $f = 0.5$. From the same figure, we also observe that



	BW	R	Ue	Ua	Uac	BW	R	Ue	Ua	Uac	BW	R	Ue	Ua	BW	R	Ue	Ua	Uac	BW	R	Ue	Ua	Uac
P→L1	40	91%	0.286	0.275	0.280	36	81%	0.325	0.306	0.309	28	63%	0.418	0.393	20	45%	0.591	0.504	0.505	15	34%	0.783	0.608	0.611
L1→L2 IDA	34	89%	0.037	0.035	0.035	29	76%	0.044	0.041	0.041	19	50%	0.064	0.062	10	26%	0.120	0.107	0.107	5	13%	0.254	0.193	0.194
L2→M IDA	34	89%	0.012	0.012	0.012	29	76%	0.014	0.014	0.014	19	50%	0.020	0.020	10	26%	0.043	0.036	0.036	4	11%	0.100	0.080	0.081

Figure 3.4: **Extra Cycles for HOC:** Performance penalty (with and without concatenation) when same degree of compression is applied to all three buses: P→L1 LDA, L1→L2 IDA, and L2→M IDA bus. BW represents bus width (in bits), R represents percentage amount of bus compression, U_e represents the expected bus utilization, U_a the actual utilization from simulations, and U_{ac} the actual utilization with concatenation. Concatenation is not possible in L1→L2 and L2→M buses for f=0.5 and hence the value is not reported.

address concatenation helps lower the performance penalty a little (up to 2.3%) in some cases where concatenation is possible. We will see next how applying different degrees of HOC to different buses based upon their differing performance sensitivities yields better results.

3.6.2 Extra cycle penalty for HOC in individual buses

The above results point to the fact that it may be unfair to compare across narrow buses at different levels of the system after applying the same degree of HOC in all of them. This is because a bus closer to the processor which is utilized more will suffer a greater penalty due to reduction in bus lines than a bus at higher level of the memory system. Hence, we study in Fig. 3.5 how the degree of HOC (f) affects the number of extra cycles when applied to one bus only at a time. From the plot, we observe that for degrees of HOC (f) closer to unity, the P→L1 LDA bus is the most sensitive followed by the L1→L2 IDA and L2→M IDA buses. This is because the P→L1 LDA bus is the most utilized among the three and reducing its width causes most penalties in the system. However, as the L1→L2 IDA bus width is reduced more and more, beyond $f = 0.5$, the performance penalty begins to increase sharply, thus making the L1→L2 IDA bus more sensitive than P→L1 LDA bus to HOC. This is because for a reduction in bus width corresponding to values of f from 0.5 to 0.2, the number of cycles taken for transmission of an address on the L1→L2 IDA bus increases from 2 to 5 cycles, whereas the corresponding number of cycles for an P→L1 LDA increases from 2 to 3 only. The L2→M IDA bus has much lesser penalties till about $f = 0.2$, possibly because of the high degree of underutilization of that bus.

To compare fairly across buses, we keep the extra cycle penalty when HOC is applied to

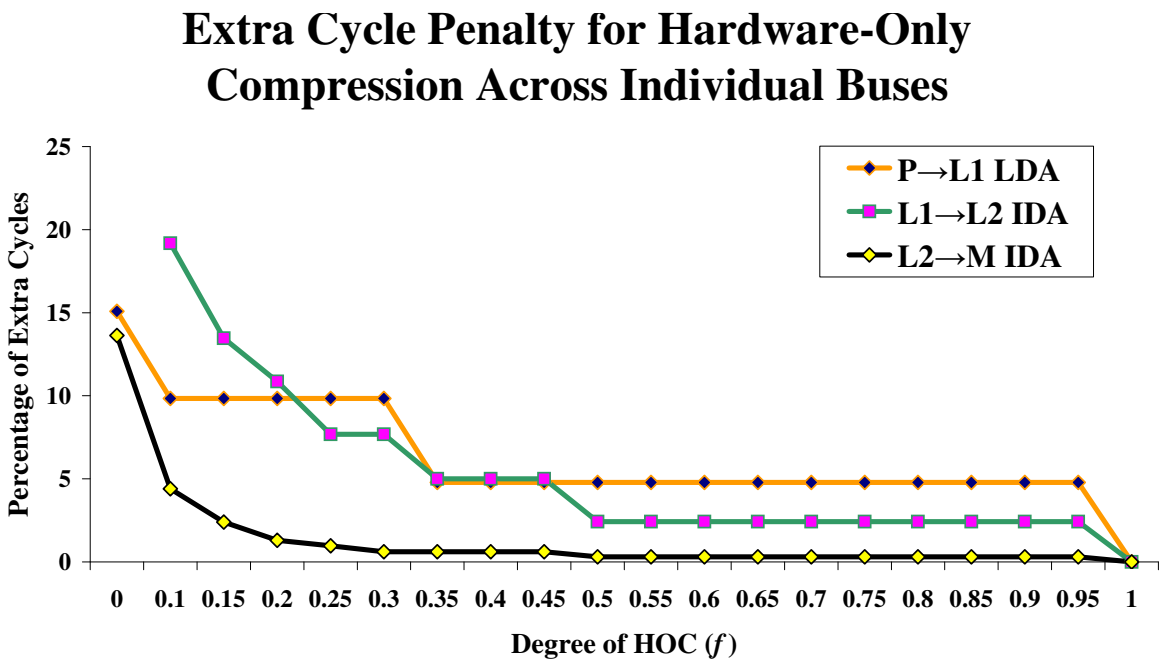


Figure 3.5: **Extra Cycles for HOC:** Extra cycle penalties for different degrees of HOC for P→L1 load address, L1→L2, and L2→M buses.

only one bus in the system at a nominal value (say, 5%) and find the value of f that gives this value for each bus. This value, denoted as f_{base} for a bus, represents the maximum degree of HOC possible in that bus such that the extra cycle penalty in the system does not exceed a given value. These f_{base} values for different buses can be obtained directly from Fig. 3.5 that shows the extra cycle penalty as a function of f for three buses. For a nominal system performance penalty of 5%, we find that the values of f that give us 5% performance penalties for P→L1 load, L1→L2, and L1→L2 address buses are 0.354, 0.354, and 0.12 respectively. We consider the maximum among these values f_{max} (f_{max}=0.354 is obtained for P→L1 load address and L1→L2 bus). Now, we introduce a parameter, *relative degree of HOC, p*, that can take values between 1 and $1/f_{max}$. The degree of HOC, f, that we used

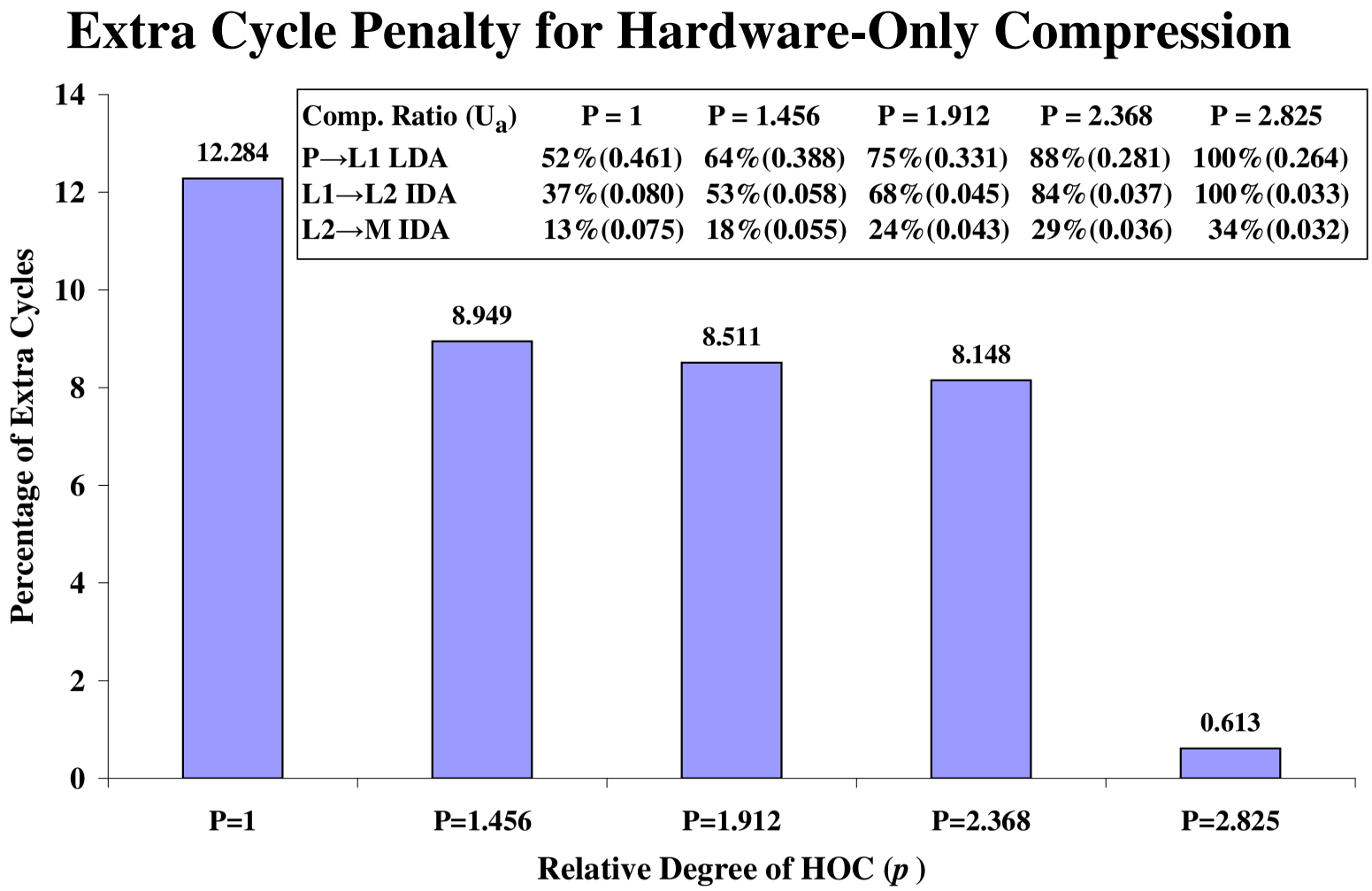


Figure 3.6: **Extra Cycles for HOC:** Extra cycle penalties for different relative degrees of HOC.

previously is related to p as follows: $f = f_{base} \times p$.

3.6.3 Extra cycle penalty for different relative degree of HOC

For different values of the relative degree of HOC, we run simulations and measure the extra cycle penalty. The results are shown in Fig.3.6. In this figure, we find that extra cycles decrease as the relative degree of HOC is increased from 1 to 2.825. In the last set (p=2.825), the P→L1 load address bus and the L1→L2 bus are uncompressed ($f = f_{base} \times p = 0.354 \times 2.825 = 1$). Although these buses are uncompressed, the other bus (L2→M) can be compressed to 34%, according to our results, with only an extra cycle penalty of 0.613%. Thus, applying HOC intelligently in address buses in the memory hierarchy can yield much better cost savings with low extra cycle penalties.

3.7 Energy-Efficient Transmission Formats

To minimize power overhead of HOC, we propose various techniques to ensure the best energy-efficient transmission format for uncompressed information transmission on narrow buses. An efficient transmission format is important because the number of self and coupling transitions and hence bus energy consumption/dissipation depend on relative positioning of different bits in the uncompressed address. Next, we present our proposed techniques. Each successive technique we present is an improvement over the previous one and results in progressively better energy reductions. The results are collected by simulating 50 million committed instructions after skipping 2 billion committed instructions initially.

3.7.1 Technique 0 (T0): HOC baseline format

Our proposed baseline transmission format, T0, for HOC is shown on the left in Fig. 3.7. The original address is split from lower order to higher order into multiple partitions and placed on the narrow bus in successive cycles. In the design of this format, we followed certain principles as described below to ensure their energy-efficiency.

- To minimize transitions on coupling capacitance between neighboring lines, correlation between the bits is necessary, i.e., bit i should be correlated with both bit $i+1$ and $i-1$. In our transmission format, different partitions transmitted in successive cycles are taken from the original address from lower order to higher order to ensure highly correlated bits are placed together. This also helps limit decoding complexity.

- During each cycle of transmission, the bits are placed on the bus starting from the LSB so that inactive lines (lines that do not carry any new data in that cycle) are placed in the higher order portion of the bus

Our T0 technique targets to minimize both self and coupling transitions and is different from the T0 code [6], which is a bus encoding scheme designed to minimize the switching activity the address bus.

3.7.2 Technique 1 (T1): HOC bus arrangement

It can be observed easily that, with the baseline transmission format, the addresses have to be transmitted in full in multiple cycles, bus energy dissipation will be higher because of misaligned bits, i.e., bits that have no correlation with bits in the same position transmitted in

the previous cycle—causes a self-transition—and bits that are uncorrelated with neighboring bits in the same cycle which causes coupling transitions. In this section, we propose a new transmission format based on the following principles of arranging the different fields to minimize self and coupling transitions.

Due to the highly sequential nature of addresses, the lower order bits of the address will be more active than higher order bits. To reduce the coupling energies of the lower order portion of the bits transmitted on the compressed bus, we place U_i, the LSB of the entire portion in the MSB line of the compressed bus for each cycle as shown in the figure on the right in Fig. 3.7. Thus, the bit U_{i+1} now occupies the LSB line of the bus and its coupling energy is reduced because it can no longer cause a toggle transition with the U_i bit. Further, the U_i bit which has been placed next to the original MSB bit also results in lesser coupling energies since its neighbor is expected to change state less frequently. Also, the edge lines have less coupling capacitance since they have only one neighboring line and this will lead to lesser coupling energies.

3.7.3 Technique 2 (T2): HOC Idle-bit insertion

When an address is sent in multiple cycles over the narrow-width bus, the last cycle of transmission of the address is likely to be poorly utilized when the compressed bus width is a non-integral fraction of the uncompressed address width. In such cases, the *idle* bits can be used to reduce coupling energies by placing them between *active* bits in different cycles of the miss. Note that if a bit is designated as idle in the current cycle, then it means that it holds the value from its previous cycle. Thus, an idle bit can never have a toggle transition

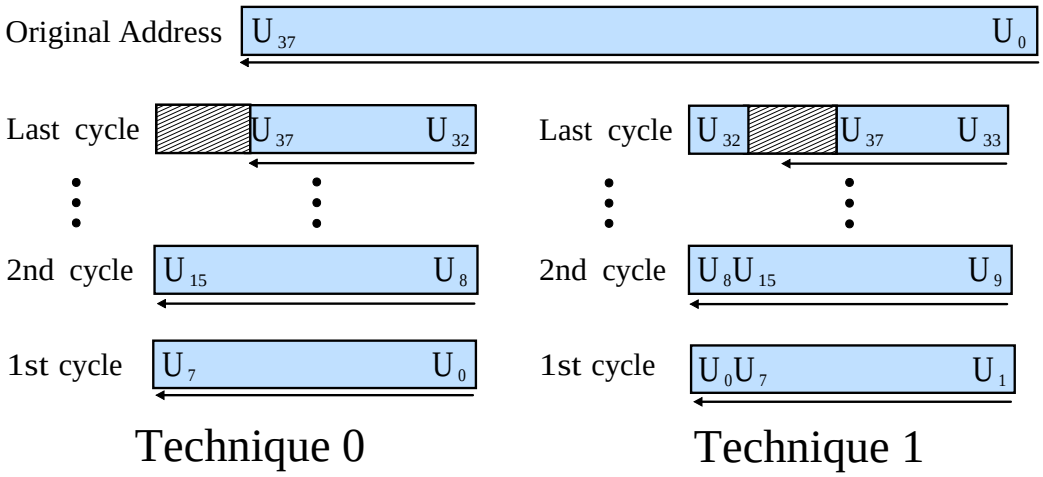


Figure 3.7: **Proposed Bus Arrangement Techniques.** The figure on the left shows the new basic transmission format that we propose for HOC. The figure on the right further reduces energy by rearranging some bits to reduce unwanted coupling transitions.

with either of its neighboring bits for the current cycle.

Given a fixed number of idle bits that we can insert, we assign the maximum possible number of idle bits to each cycle starting from the first cycle. For a *w*-bit compressed bus the maximum number of idle bits that can be assigned to each cycle is $\lfloor w/2 \rfloor$. Now, suppose *k* bits were assigned to the first cycle, then the idle bits are interspersed alternately with active bits in the cycle starting from an active bit at the LSB to achieve maximum benefits for coupling energy reduction. After assigning to the first and second cycles as above, if idle bits remain, then they are assigned to the third cycle and so on till all the idle bits are exhausted.

3.7.4 Technique 3 (T3): HOC address encoding

Transition signaling code involves a simple bit-wise XOR operation of the current word to be transmitted on the bus with the previous word to minimize transitions for off-chip

buses [61]. In address buses, due to temporal and spatial redundancy of the bits, the result will have more zero-valued bits than the current address. Thus, potentially many self and coupling energies can be reduced by XOR-ing the address word before placing on the narrow compressed bus. Note that computing bitwise XOR of two *n*-bit addresses requires constant time and little hardware and hence this will not add much extra latency to the bus interface.

3.7.5 Technique 4 (T4): HOC transmission encoding

In the transmission encoding, we again XOR each bit of the new address word with the bit transmitted at the corresponding bit position on the bus in the previous cycle. Since T3 step yielded more zero-valued bits, T4 will make the address pattern similar to the one transmitted on the bus in the previous cycle thus reducing both self and coupling energies.

3.7.6 Techniques 5 (T5) and 6 (T6): Using idle bits as active shields

In this technique, we use the idle bits that we inserted using the idle-bit insertion technique discussed earlier as active shields. In the default idle-bit insertion technique, the bits designated as idle held the value transmitted at that bit position in the previous cycle. We mentioned that toggle transitions for that bit with its neighbors can be eliminated using this technique. To further reduce total coupling energy by eliminating the occurrence of coupling charge and discharge energies at the cost of some self-energy, we propose to change the value held by the idle bit according to the changes of the values of its neighboring active bits. In T5, the value held by an idle bit becomes equal to its adjacent bits when the adjacent bit pair transitions from $00 \rightarrow 11$ or from $11 \rightarrow 00$. In T6, the value held by an idle bit becomes

equal to its adjacent bits when the two adjacent bits are equal. Some self energy is expended due to the extra switching activity of the idle bit due to this scheme but that will be a small portion compared to the savings in coupling energy. Our idle-bit shielding techniques are different from the active shielding in [31] because our technique is an info-pattern dependent shielding approach.

On-chip and off-chip energy results for all the proposed transmission schemes for L1→L2 address bus are shown in Figs. 3.8 and 3.9. T1 is slightly better, 1% more on-chip energy saving, than T0. Since T1 doesn't affect the self transitions, the off-chip energy is the same as T0. On average, idle-bit insertion technique, T2, causes more on-chip and off-chip energy than T0. This is due to the increased charge and discharge transitions cause more on-chip energy than the reduced toggle transitions. However, after we apply T3 on top of T0, T1, and T2, 5% on-chip energy saving can be obtained compared to the uncompressed bus and 18% off-chip energy improvement achieved w.r.t. T0. For off-chip bus, T4 provides the best improvement, 45% w.r.t. T0. T5 and T6 yield about 8% and 16% on-chip energy reductions on the average, respectively.

3.8 Address Compression and Bus Encoding

Bus-invert (BI) encoding [60]: Bus-invert encoding was one of the original techniques proposed for reducing self-transitions on buses. The scheme examines the number of bits that are different (Hamming distance) between the current input pattern and the pattern transmitted on the bus in the last cycle. If this number is greater than half the bus-width, all the bits

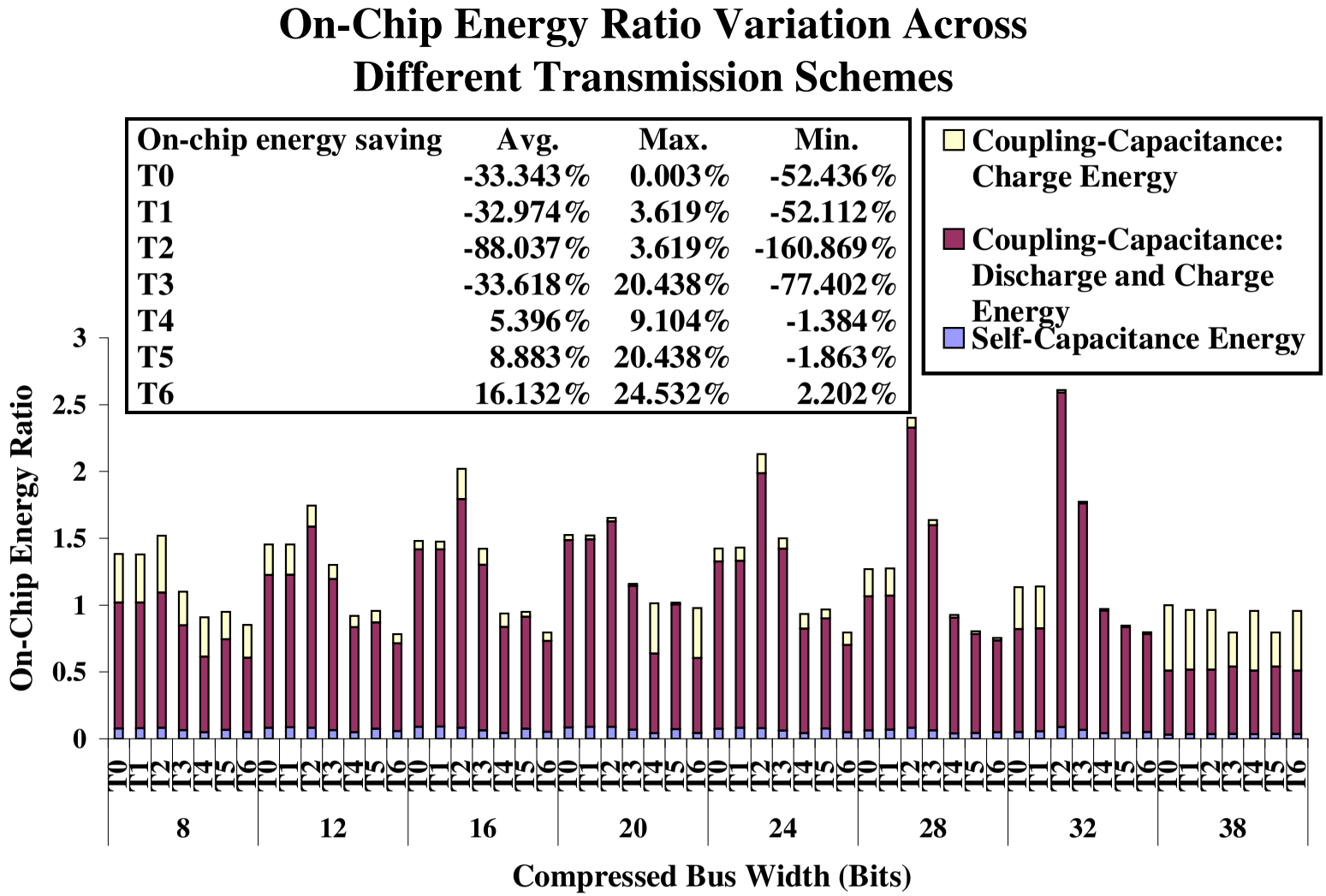


Figure 3.8: **On-Chip Energy Reduction Using All the Proposed Techniques.**

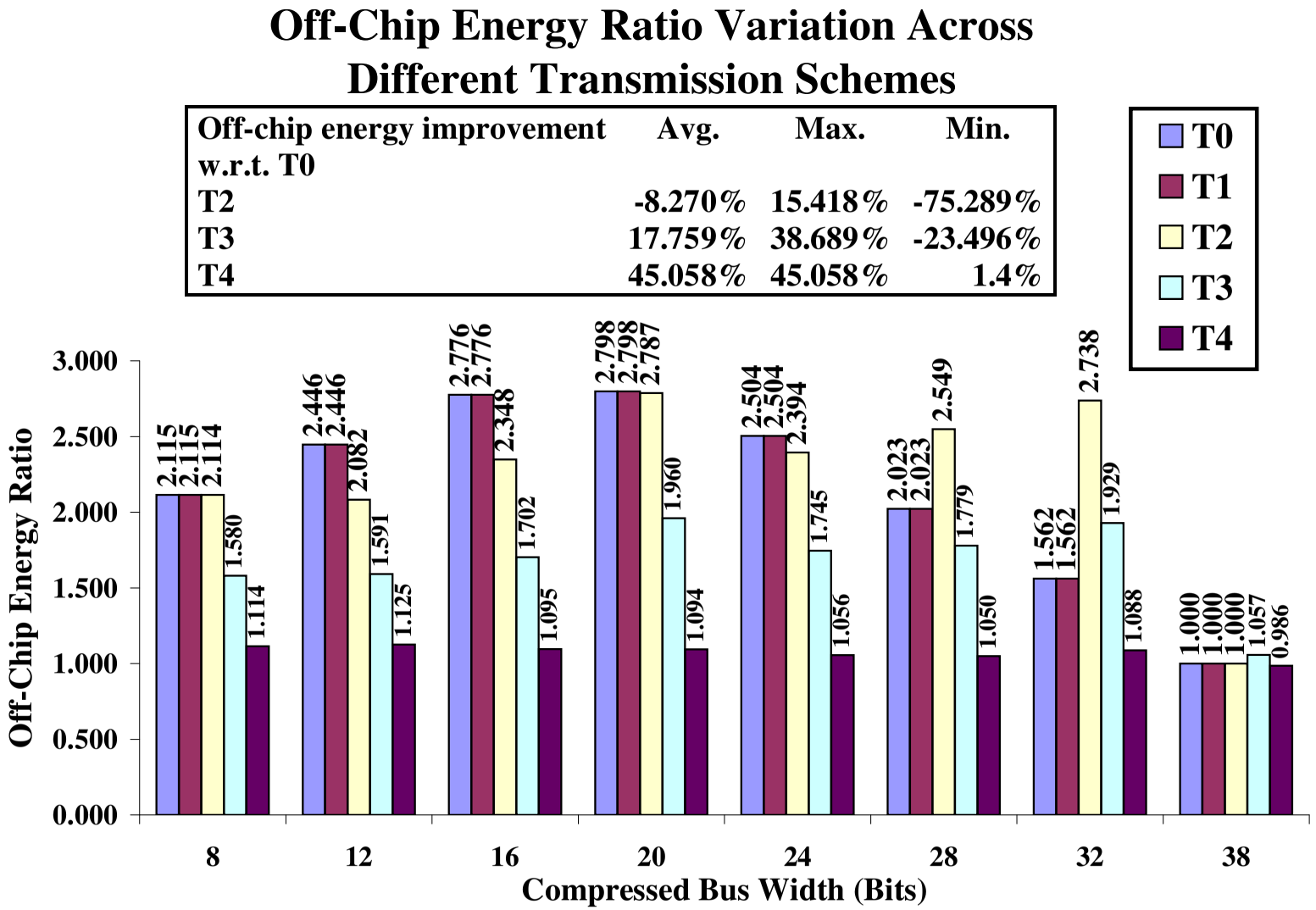


Figure 3.9: **Off-Chip Energy Reduction Using All the Proposed Techniques.**

of the current input pattern are inverted and a separate *invert* line is held high. Else, the value is transmitted in original form and the invert line is held low.

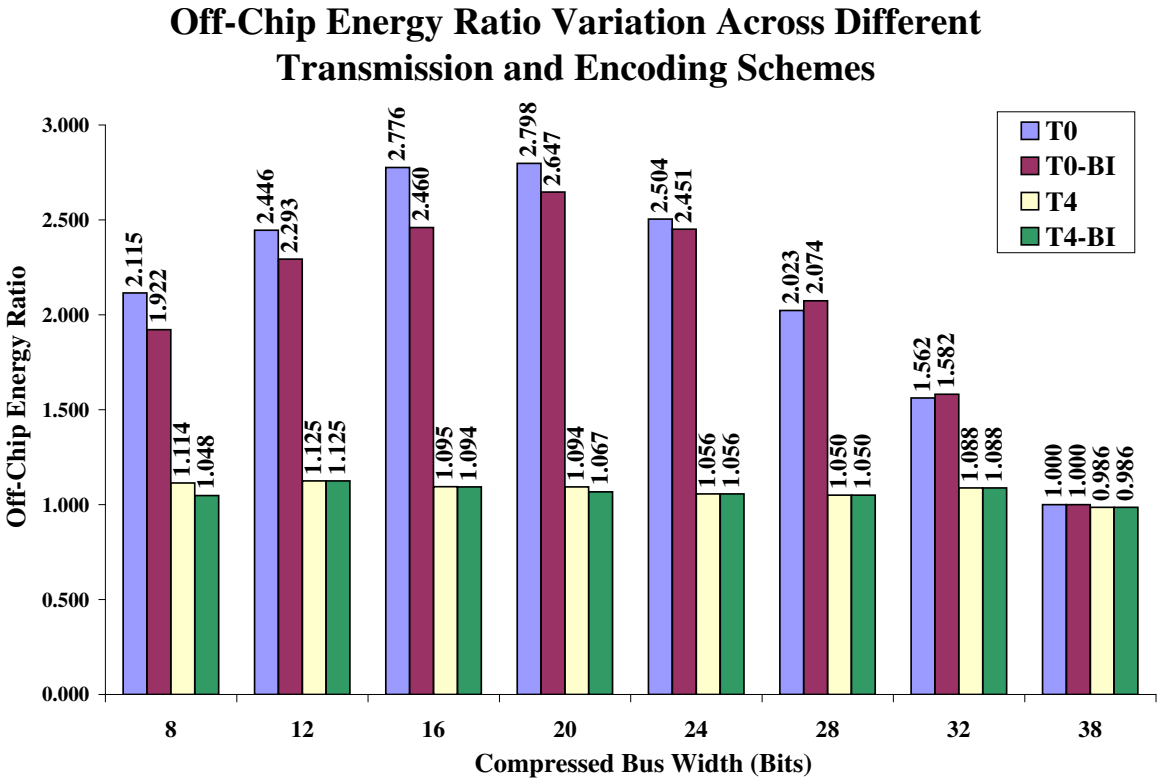


Figure 3.10: **Off-Chip Energy Variation Across Transmission and Encoding Schemes.**

Odd/even bus-invert (OEBI) encoding [77]: Since adjacent wire coupling occurs between an even-numbered and its adjacent odd-numbered wire on a bus, this method encodes even and odd bit positions separately and uses two invert lines to indicate one of four modes of transmission: 00 – none of the bits are inverted, 01 – even bits are inverted, 10 – odd bits are inverted, and 11 – all bits are inverted. A two phase transfer method (TPTM) was also suggested to prevent the occurrence of the *toggle* case (01→10 or 10→01) between any two adjacent bit-lines. The toggle case results in the maximum power energy dissipated since the effective coupling capacitance between the toggling wires is four times the normal value.

To minimize toggling, one of the lines in the toggling pair is delayed by one cycle, i.e., 01 becomes 11 or 00 and finally 10, which results in less energy dissipated than the worst case, although it costs an extra clock cycle.

Coupling-driven bus-invert (CBI) encoding [34]: This method examines all pairs of adjacent bits and counts the number of coupling transitions. The current input bit pattern is inverted if the coupling effect of the inverted pattern is less than that of the original pattern. Since odd and even lines are not handled separately, this scheme requires only one extra invert line.

In this study, we apply three encoding schemes on HOC address buses to investigate if bus encoding can decrease actual energy further. Since T0 is our baseline transmission format, T4 is best for off-chip buses, and T6 is best for on-chip buses, we apply encoding on top of these three techniques. As we can see in Fig. 3.10, BI with T0 provides better off-chip energy saving than T0 itself. For BI with T0, off-chip energy ratio is reduced by 0.09 compared to T0 whereas T4 itself can improve the off-chip energy by 0.98 w.r.t. T0 on average. We also apply BI on on-chip HOC address buses to see how it performs. Fig. 3.11 shows the on-chip energy ratios across these schemes. The three encoding schemes actually consume more on-chip energy than without encoding on HOC address buses. The extra energy might be caused by the extra control lines used in the schemes. Our T6 without any extra control lines, which reduces on-chip energy by 15%, is the best for on-chip HOC address buses among all schemes we examine. So for both off-chip and on-chip compressed address buses, our transmission techniques are much more effective than the encoding schemes.

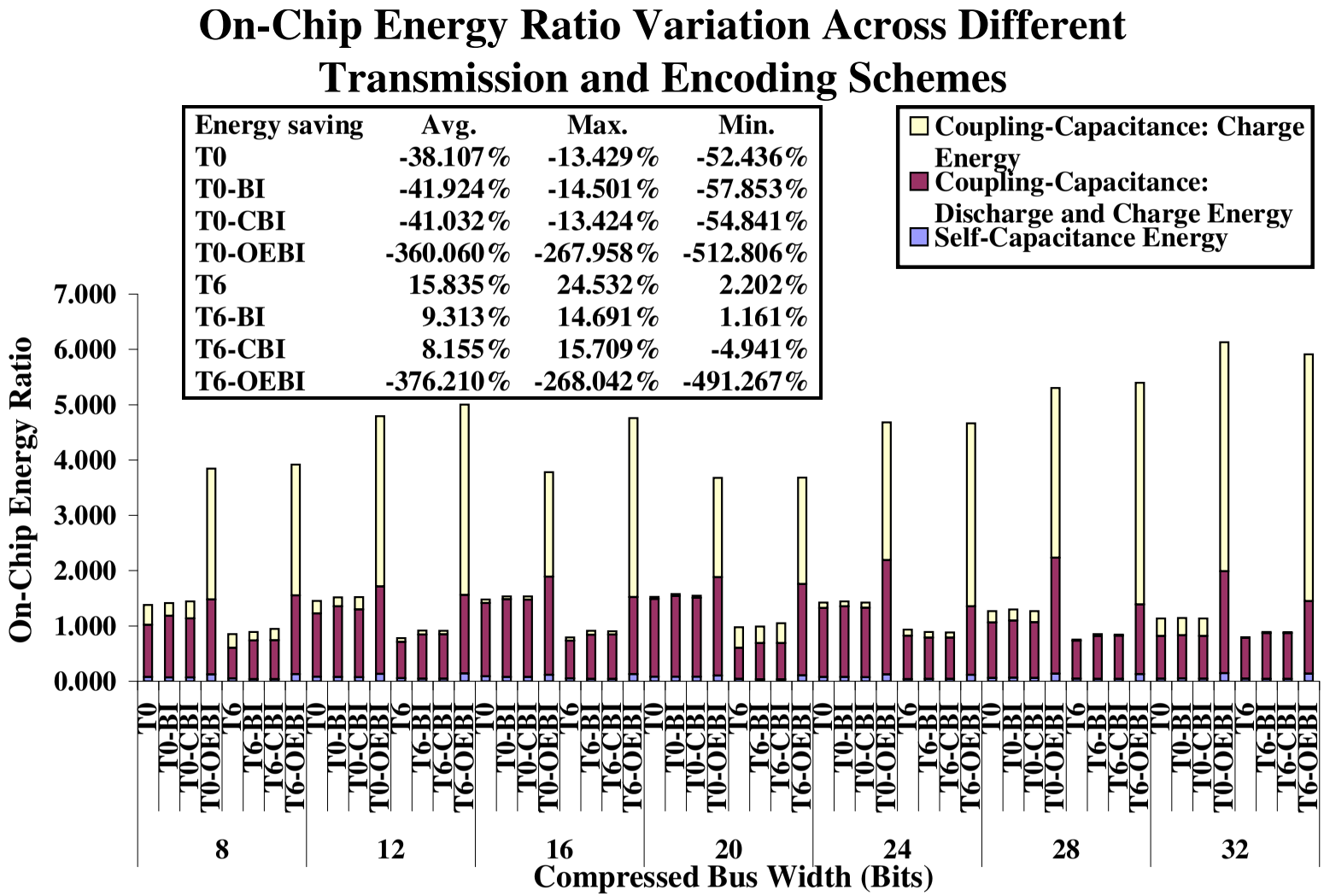


Figure 3.11: **On-Chip Energy Variation Across Transmission and Encoding Schemes.**

3.9 Performance and Energy Optimization with Wire Spacing

As mentioned in Sec. 3.3.1, the wires in a compressed bus can be spaced further apart while maintaining the area footprint smaller or equal to the original bus to minimize the wire delay and coupling capacitance. In this study, we spaced the wires of the compressed L1→L2 buses at different spacing degrees (SD), the percentage of original bus area, to improve performance and reduce energy. When the spacing degree is less than 100%, $1 - SD$ of the original bus area can be saved.

Fig. 3.12 shows that the less the number of bus wires the compressed bus has, the more effective the wire spacing scheme is. This is because narrower compressed bus provide more extra area for spacing. The wire delay decreases dramatically when the spacing between wires increases. For 8-bit compressed bus, 88% wire delay reduction can be obtained when SD is 100%. Even for 32-bit compressed bus with the same SD, the wire delay can be reduced by 27%. On average, the wire delay can be improved by 61% with HOC and wire spacing. In Fig. 3.13, we examine the performance improvement for the compressed buses with wire spacing. HOC with wire spacing can actually improve the performance up to 0.8% to 15% w.r.t. original address bus. As shown in Fig. 3.14, HOC with wire spacing can also reduce on-chip energy to large extent. With T6 and wire spacing, up to 88% energy can be reduced for 8-bit bus and 42% for 32-bit bus, whereas T6 itself can only reduce 14% and 20% for 8-bit and 32-bit buses, respectively. On average, 60% on-chip energy can be saved

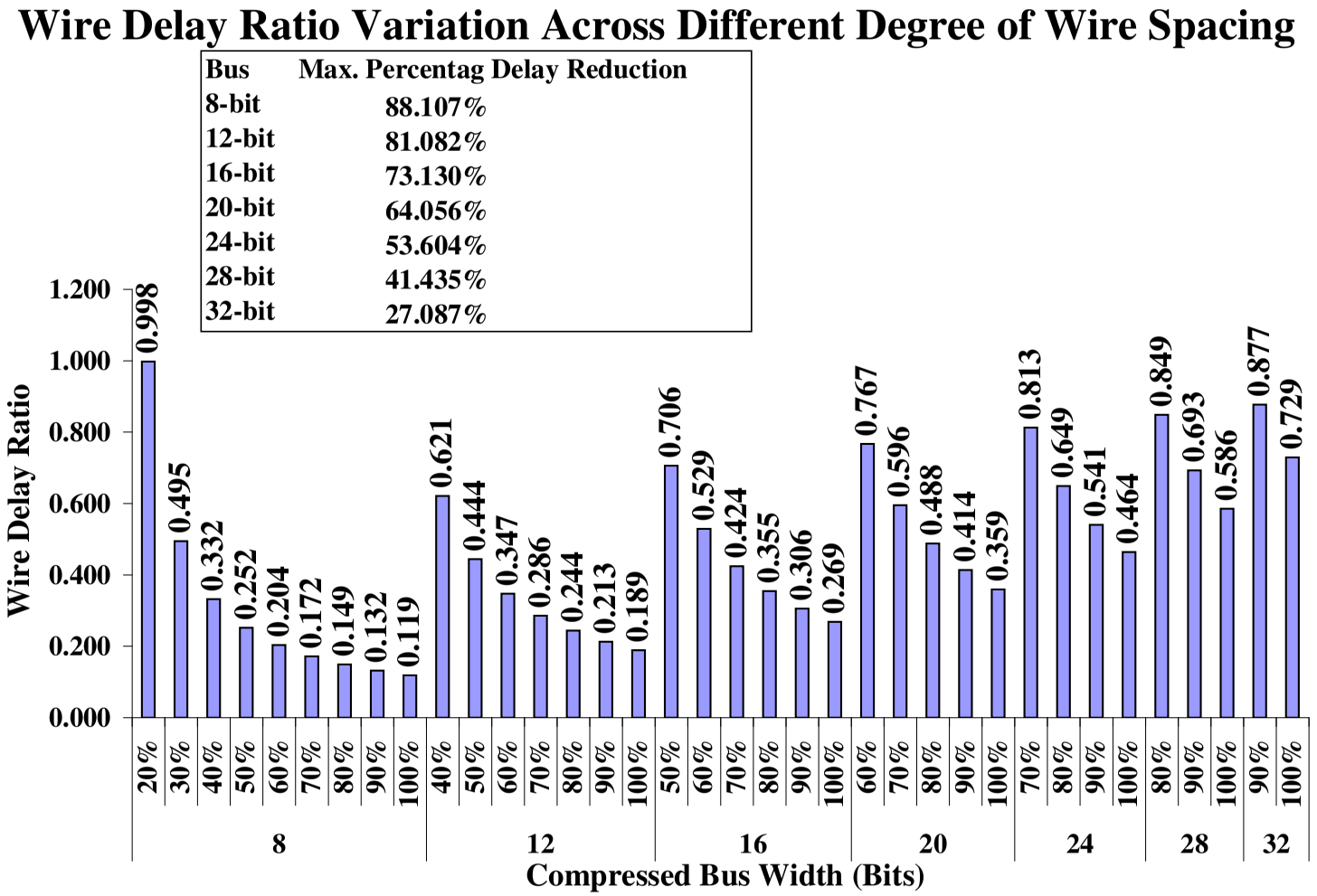
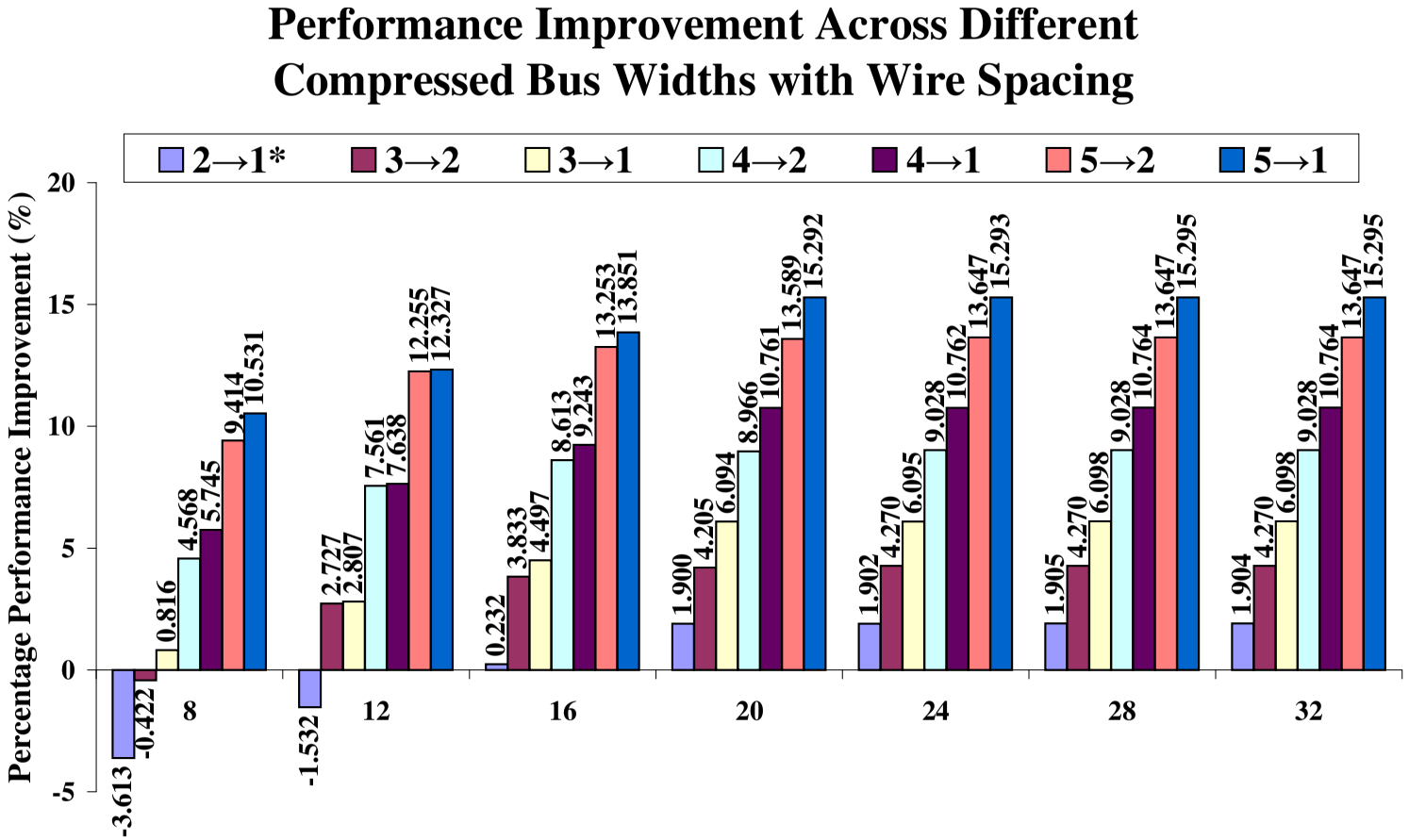


Figure 3.12: **Wire Delay Reduction Using HOC with Wire Spacing.**



***2→1: Percentag performance improvement when bus latency, 2 CPU cycles, is reduced to 1 CPU cycle after wire spacing**

Figure 3.13: **Performance Improvement Across Different Compressed Bus Widths With Wire Spacing.**

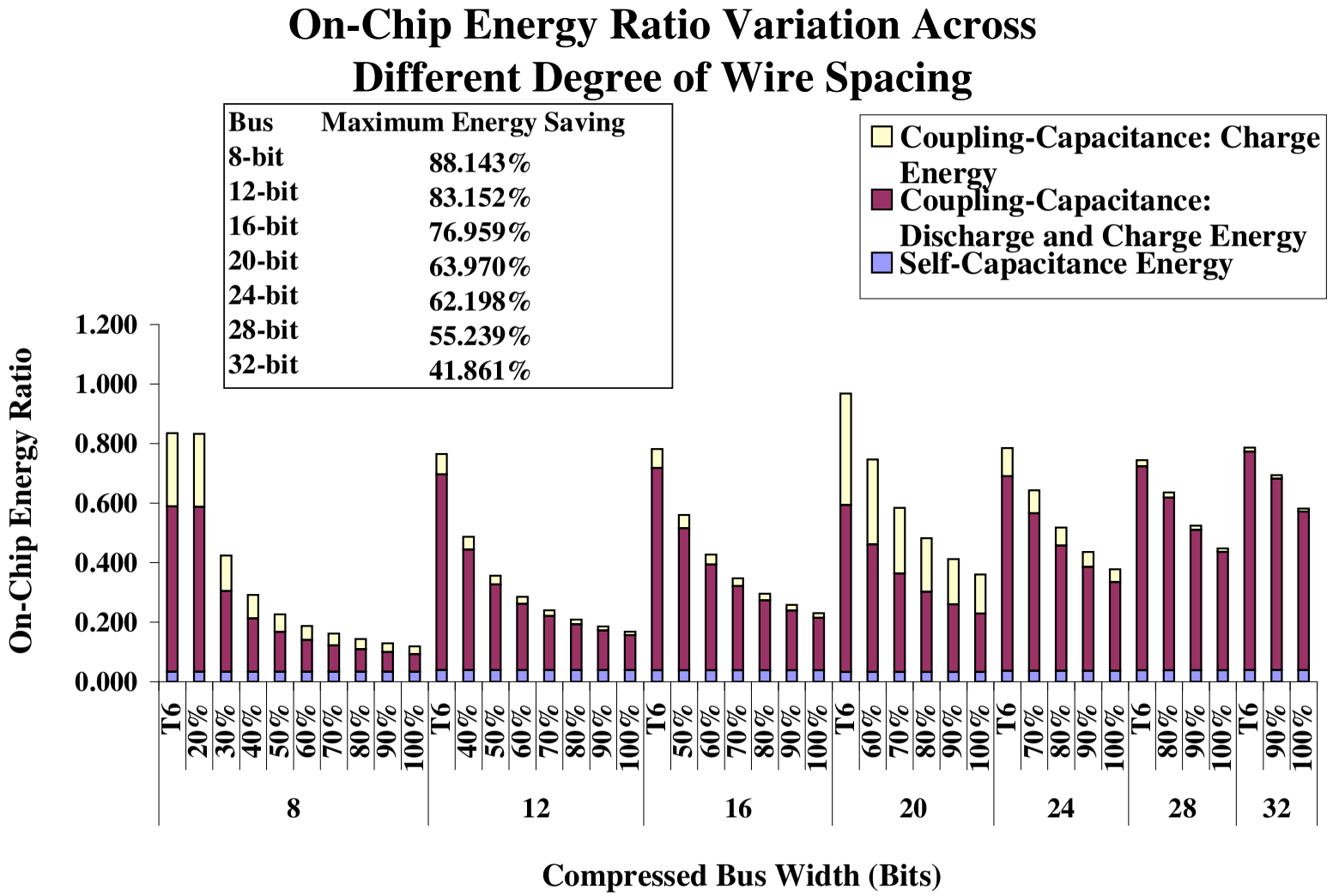


Figure 3.14: **On-Chip Energy Reduction Across Different Compressed Bus Widths With Wire Spacing.**

using HOC with T6 and wire spacing.

3.10 Conclusions

We proposed and analyzed the overheads (extra cycles required and power consumption) of a hardware-only compression scheme to reduce costs and improve power consumption of underutilized address buses in the memory system. Our simulations show that by carefully choosing the relative address bus widths at different levels of the memory system, hardware only compression schemes can result in reductions of up to 34% in number of lines of a L2→M address buses with only a 0.613% increase in number of cycles required for execution. For 8-bit compressed bus, 88% wire delay reduction can be obtained when SD is 100%. On average, the wire delay can be improved by 61% with HOC and wire spacing. Up to 0.8% to 15% performance improvement can be achieved for L1→L2 compressed address buses with wire spacing. We have also proposed energy-efficient transmission format to minimized on-chip and off-chip energy. On average, 16% on-chip energy reduction can be obtained using our best transmission technique, T6. With wire spacing, T6 can provide up to 88% energy saving for 8-bit compressed bus. On average, 60% on-chip energy can be saved with HOC and wire spacing. Off-chip energy can be improved by 45% with T4 compared to the baseline transmission format.

Chapter 4

Analysis of Dynamic Address Compression Schemes

4.1 Introduction

Address compression, when applied to on-chip address buses in current microprocessors or systems-on-chip (SoCs) can potentially reduce or mitigate some of the problems associated with interconnect scaling in current nanometer-scale technologies. Employing compressed addresses for certain buses in the design will help reduce the number of address lines needed for those buses, result in less area overheads and lower costs, and may also potentially facilitate their routing. Also, due to the smaller area occupied by the bus, capacitance and thus bus energy may reduce. Further, by using area no more than a bus of original width, a narrow bus can: (1) use greater spacing between bus lines, which will

reduce inter-wire capacitance and hence delay, bus energy, and cross talk; and/or (2) use wider wires to reduce resistance and hence delay and potentially improve performance. Using an address compression scheme may itself entail some performance, area, and power consumption overheads due to extra logic. But these overheads will not be much compared to the savings potentially obtained by compressing long on-chip buses and off-chip buses. This is because the size, speed, and power consumption of logic (which will be used to do compression/decompression) scale better than those of interconnect (which will be used to communicate the information), and hence these overheads will continue to decrease over time.

4.1.1 Related work and our contributions

Address buses have been studied widely in previous work and schemes have been proposed to improve their performance, power consumption, and/or area/cost. Various bus encoding schemes have been proposed to reduce power consumption in address buses many of which are surveyed in [8]. Compression, which is related to encoding, can also provide similar or greater energy benefits in addition to performance improvements and cost reduction for almost all components in a processor-memory system as we found in our earlier work [50]. We also proposed a simple scheme to reduce cost and improve the utilization of address buses in [49].

A specific scheme for address compression using a small *compression cache* (cache specially used for compression) at the sending end and base register files at the receiving end of a bus was first proposed in [52, 20], and subsequently used for compressing instruction

and data buses in [11]. However, none of the above works consider address compression as a means of improving energy efficiency while reducing costs at the same time. Also, they do not study on-chip buses. Only recently, the effectiveness of these schemes in reducing the switching activity in data buses was studied in [3]. However, results reported in the above work too do not reflect actual energy reductions for current technologies since: (i) only switching activities were considered and (ii) it does not provide an estimate of the effectiveness of address compression in reducing *self-energy* (bus energy dissipated due to transitions in the line self-capacitance) and *coupling energy* (bus energy dissipated due to transitions on the coupling capacitance between two adjacent lines) separately. This is important because, in current and future technologies, coupling energy dominates self-energy by almost an order of magnitude. Recently, address compression was also used to compress addresses and data transmitted between processors in a multiprocessor server to improve bandwidth and reduce costs due to pins [30].

In our study, using a simulator that models a realistic processor, we present results on how address compression schemes perform when applied in on-chip or off-chip buses in modern superscalar processors. In particular, we explore the performance, energy, and cost benefits of address compression, the effect of techniques like bus pipelining, and the effect of technology scaling on energy-efficiency of compressed address buses. We use two metrics in our study – *extra cycle penalty* and *energy ratio* – that help us quantify: (1) the actual performance penalty due to address compression (including the effect of hardware latency and pipeline stalls) on the system and (2) energy dissipation in buses including the effect of

inter-wire capacitances for various metal routing layers in nanometer-scale technology nodes for on-chip buses. We consider buses carrying physical addresses—instruction and data addresses are carried on the same bus—between level-one (L1) and level-two (L2) caches in a system and report results for two cases: (i) when the address bus connects L1 and L2 caches that are both on-chip like in most modern processors; and (ii) when the address bus connects L1 cache to off-chip memory (L2 cache or DRAM) like in the case of many SoCs. Overall, our work is the first to study address compression in detail, from the perspective of optimizing performance, energy, and cost, for these types of buses.

The organization of the rest of this chapter is as follows. In Sec. 4.2, we discuss two dynamic address compression schemes and discuss ways to optimize system performance and area/costs when using these schemes practically. Next, in Sec. 4.3, we describe our simulation environment and methodology. Then, in Sec. 4.4, we describe our experiments and discuss results. Finally, we conclude in Sec. 4.5.

4.2 Dynamic Address Compression

Our study focuses on two schemes: dynamic base register caching (DBRC) and bus expander (BE) that have been proposed previously for processor-memory address compression. These are described briefly below.

4.2.1 Dynamic base register caching

Since higher order portion of the address has more redundancy than lower order portion, in dynamic base register caching, the original address is split into a higher order and a lower

order component and the former is stored in a compressor, a cache of base registers, at the processor side in Fig. 4.1. Upon a cache hit, the index and entry number to the base-register cache (BRC) is transmitted on the bus with the uncompressed lower order part of the original address in a single cycle. Due to address locality, a hit will occur most of the time and since the number of bits in the entry number is shorter than in the higher order portion, a narrower bus can be used to transmit the address. As shown in Fig. 4.2, a miss in the processor BRC (sending end) is indicated by sending a reserved bit pattern on the bus in the first cycle followed by the missed address in subsequent cycles. The memory (receiving) side consists of a register file that is loaded with this missed address. The BRC on the processor is also updated simultaneously. Based on their simulations, the authors conclude that using a 16-bit bus for transmission of 32-bit addresses with the DBRC scheme will result in a miss rate of only 2% and most of the time the memory address could be transmitted using the 16-bit bus in a single cycle. For this bus width, a fully associative BRC of at least 15 entries or a direct mapped BRC of at least 63 entries was suggested as the optimal configuration.

4.2.2 Bus expander

In this scheme, the sending end has a look-up table (LUT) that caches the higher order portion of the address at the processor side [11]. Upon a hit in the sender-LUT, control signals, index, and the uncompressed portion are transmitted on the address bus in a single cycle. Similar to DBRC, a narrow bus is used in this scheme also and in case of a miss, the entire address is transmitted in multiple cycles. But, as we can see in Fig. 4.2, a miss is not explicitly indicated with a reserved bit pattern as is done in the DBRC scheme. Rather, the

BE logic at the sending end begins to transmit the entire address immediately starting from the first cycle. A separate control signal line is used to indicate hit or miss in the sender-LUT. Using this control signal, the BE at the receiving side, which is a set of base registers similar to the one used in DBRC, can determine whether the information on the bus corresponds to a compressed address or a missed address. In the latter case, the logic updates the receiver-LUT with the missed address after it has been received. Therefore, for the same compressed bus, the miss penalty for BE is less than for DBRC. However, if both the number of bus lines and the number of entries are same for DBRC and BE, the compressed portion in BE is one bit more than in DBRC due to the control signal line, which might increase the miss rate in BE. To further reduce the number of cycles taken to service a miss, the authors propose that a smaller LUT with 4 entries can be used for the 4 higher order bits of the address. This increases the number of control lines required to two. In [11], the authors show that using a 16-bit narrow bus for 32-bit addresses, only a single cycle is needed to transmit the address 90% of the time.

4.2.3 Overheads of address compression

Using address compression schemes in microprocessor buses entail performance, area, and power consumption overheads that have not been considered in previous work. The performance overheads are due to missed addresses that require more than one cycle to be transmitted. These overheads occur due to two reasons. First, since addresses occur non-uniformly over time, buffering of missed addresses will be required at the sending end and even then these buffers may fill up due to a burst of misses, necessitating pipeline stalls.

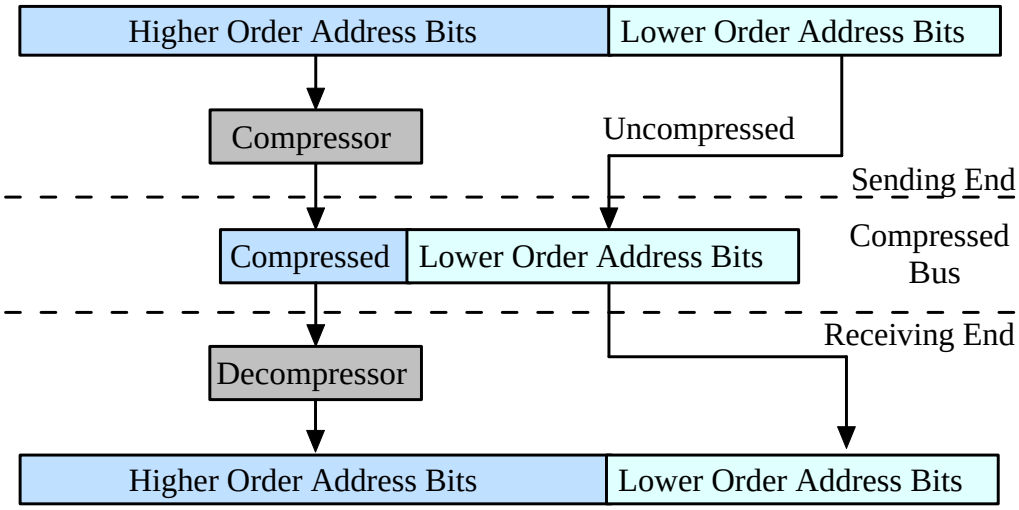


Figure 4.1: **Dynamic Address Compression Schemes:** General schematic of a dynamic address compression scheme.

Second, since missed addresses arrive at the receiving end later, cache/memory access is delayed and this delayed fetch may cause a dependent instruction to stall the pipeline. However, modern processors using dynamic scheduling can minimize the occurrence of such stalls by executing instructions out of order. Performance overheads can be mitigated in two ways. First, more buffering can be done at the sending end to avoid buffer-full related stalls. Second, missed addresses can be transmitted in *w*-bit groups from the high- to low-order end so that address tag and index fields are received quickly to start up cache/memory at the receiving end early even before the entire missed address is received. We describe next how address buses can be optimized for performance and cost by choosing cache sizes and bus widths appropriately.

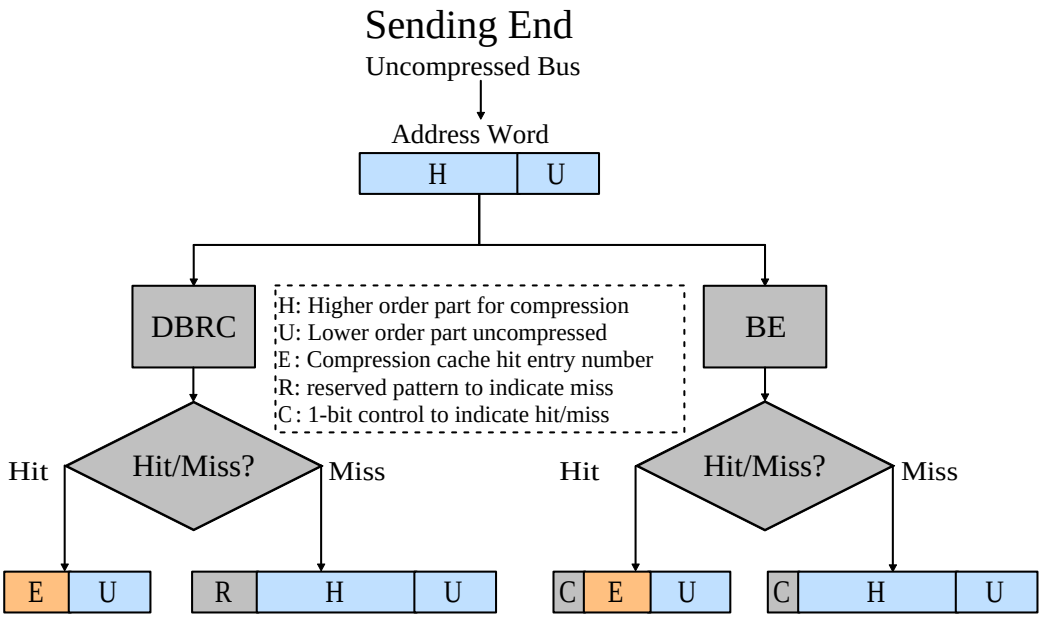


Figure 4.2: **Dynamic Address Compression Schemes:** Schematic depicting how DBRC and BE form a compressed address word differently before sending it on the compressed bus.

4.2.4 Optimal index sizes

As described earlier, a narrow bus is used to transmit compressed addresses in DBRC and BE. The compressed bus width w is determined using the relation: $w = u + \log_2(e)$ (plus 1 control bit for BE), where u is the portion of the address (lower order part) that is left uncompressed and e is the number of entries in the compression cache. Note that if W is the original address width (also the original bus width) and compression cache is direct mapped, then $t = W - (\log_2(e) + u)$ is the width of the tag that is stored in the compression cache. Decreasing the address bus width (w), while maintaining the width of the uncompressed portion (u) the same, will reduce the number of entries (e) in the compression cache. This may lead to higher miss penalties (the number of extra cycles needed to transmit an address

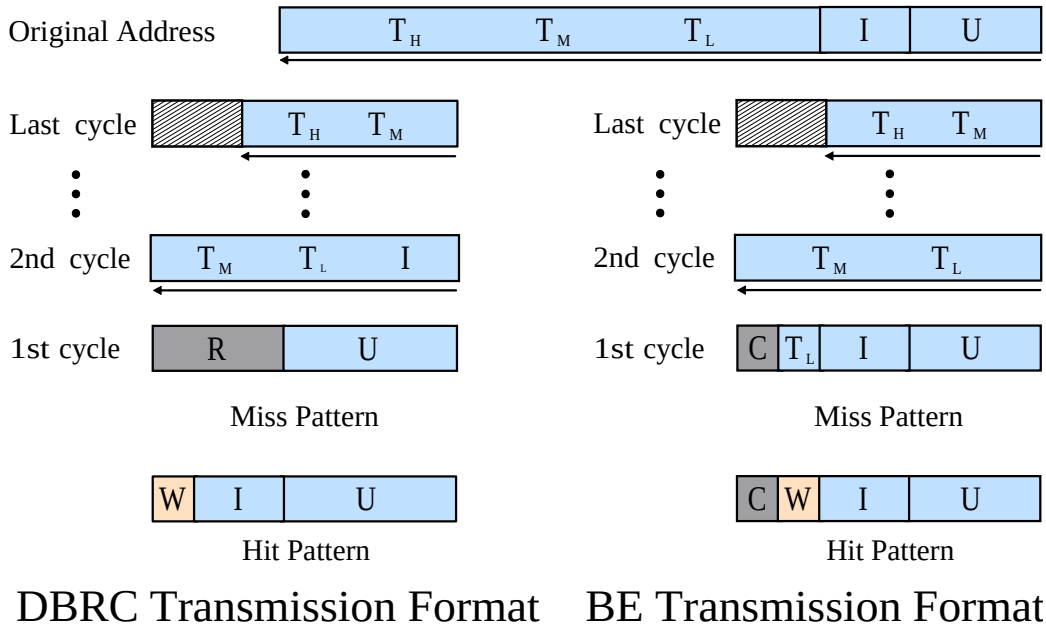


Figure 4.3: **Dynamic Address Compression Schemes:** Our default transmission format for DBRC and BE.

if it misses in the compression cache) and therefore degrade system performance due to the following reasons: (1) fewer entries in the compression cache cause more misses and (2) wider tag (t) to find a match for reduces the chances of a hit. Thus, there exists a range of values for w and e for which the system performance will be affected the least when address compression is used. We are interested in finding these ranges of values. Note that the combination of w and e is proportional to the cost of using address compression—e represents the extra compression cache hardware added and $W - w$ represents the number of bus lines that can be removed.

4.2.5 Compressed address transmission format

In this section, we describe an energy-efficient transmission format for compressed addresses. Previous work has not clearly specified any format for transmission of compressed

addresses. An efficient transmission format is important because the number of self and coupling transitions and hence bus energy consumption/dissipation depend on relative positioning of different bits in the compressed address. In this work, we suggest a basic transmission format for DBRC and BE.

Our proposed baseline transmission format for DBRC and BE is shown in Fig. 4.3. In the case of a hit, when the compressed address is transmitted, the entry (E) field consists of an *index* (I) field, which points to line number which hit in the sender cache, and a *way* (W) field which points to the set number for the tag match in the line. The rest of the bus lines are filled with the control bit (C) field which occupies the most significant bit (MSB)—only in the case of the BE scheme— and the uncompressed (U) field which occupies the remaining bits in the lower order portion of the bus. It can be noted that the absence of control bit in the compressed address for DBRC means that the I or U fields can be one bit wider.

In the case of a miss, during the first cycle of transmission, the entire E-field is replaced by a reserved bit-pattern (R) in the DBRC scheme whereas in BE it is replaced with a portion from the higher order part of the address. Other fields in the first cycle—U-field in the case of DBRC and C- and U-fields in BE are derived in a manner as in the case of a hit. In subsequent cycles, other portions of the higher order part of the address (tag or H-field) are transmitted on the bus but this time they can fill up the bit lines corresponding to the U and C fields too. Note that the U- and C-fields must be transmitted in the first cycle of a miss so that the decompresser can detect the miss and calculate the number of cycles in which it will receive the complete address.

In the design of these formats, we followed the principles mentioned in Sec. 3.7.2 with some changes to ensure their energy-efficiency.

- To minimize transitions on the self-capacitance of the wires, bits in the same position should be correlated across consecutive cycles. Hence, where possible, different fields of the compressed address like the U-field, I-field, W-field, or different parts of the H-field (T_L, T_M, and T_H) should be placed at the same position on the bus irrespective of hit or miss. This also helps limit decoding complexity. In the transmission format shown on the left in Fig. 4.3, we have placed the U-field and the I-field in the same position (lower order bits of the bus for hit as well as miss) for this reason.

- To minimize transitions on coupling capacitance between neighboring lines, correlation between the bits is necessary, i.e., bit i should be correlated with both bit $i+1$ and $i-1$. In our transmission format for a hit, the I-field and U-field are placed together since they are both derived from the original address and hence can be expected to be highly correlated than the case in which W-field and U-field are placed together. Further, to minimize transitions on self-capacitance as discussed earlier, this arrangement is followed in the first cycle of a miss also, although there is no way-field in that case. Here, the bits corresponding to the W-field are occupied by bits from the lower order portion of the H-field represented by T_L in Fig. 4.3.

- Finally, during each cycle of transmission, the bits are placed on the bus starting from the LSB so that inactive lines (lines that do not carry any new data in that cycle) are

placed in the higher order portion of the bus

4.3 Simulation Methodology

We used *sim-alpha*, the validated Alpha 21264 simulator [18] based on the SimpleScalar tool, as the platform for our experiments. The benchmarks and simulator configuration we used are summarized in Table 3.1. In this simulator, we implemented DBRC and BE for address compression on the L1→L2 address bus. As mentioned earlier, we report results for two cases: (i) when the bus connects L1 and L2 caches that are both on-chip like in most modern processors and (ii) when the bus connects to off-chip memory (L2 cache or DRAM) like in the case of many SoCs. In either case, we that the assume compression hardware is placed after the L1-cache but before the buffer chains that drive the L1→L2 address bus. Thus, we assume that the L1 miss address file (MAF) stores uncompressed addresses from L1-cache and is drained when the compression hardware is free. We also assume that instruction and data addresses are compressed using the same hardware. Further, in the default case, we assume that the compression and decompression of addresses take negligible time and that buses are not pipelined. The former assumption is justified even for the largest size of our compression cache which is of the order of a few kilobits because L1 cache sizes in current systems—which are at least 10 times larger—have only a single cycle latency. The extra cycle penalty and bus energy model for performance and energy evaluation are mentioned earlier in Sec. 3.4. 50 million committed instructions are simulated after skipping 2 billion committed instructions initially.

4.4 Simulations and Results

4.4.1 Performance, energy, and cost tradeoffs

In this experiment, we examine three-way tradeoffs between performance, energy, and cost when using DBRC and BE schemes for L1→L2 addresses. Table 4.1 reports values for five quantities: extra cycle penalty, cache size, on- and off-chip energy ratios, miss rates, and compression ratios for various bus widths. These bus widths were chosen so that all trends for variations in the above-mentioned quantities can be captured with minimum number of bus widths.

As explained in Sec. 4.2.4, an optimal number of bits that should be allotted to the index field that results in the minimum extra cycle penalty for a given bus width can be found. These values, which we determined experimentally, are reported in the bottom lines of each row in Table 4.1. For example, the extra cycle penalty of a 16-bit bus for the optimal index width is reported as: $[5, 0.18\%]$, i.e., if an index width of 5 bits is used, the extra cycle penalty will be only 0.18% compared to address transmission on an uncompressed bus. The corresponding cache size is given on the bottom line of the next row which is 1575 bits. Similarly, the energy ratios are 1.16 (16% energy overhead) and 0.94 (6% energy reduction) for the off-chip and on-chip cases, respectively, when the optimal index width is used for this bus.

We also found that, by tolerating a slightly higher extra cycle penalty, it may be possible to reduce the compression cache size (hardware cost) substantially. Results for this configuration are indicated on the top lines of each row. In this study, we limited the extra cycle

		Bus Width				
		8	10	12	14	16
Extra Cycle Penalty	Min.			[3, 1.55%]	[2, 0.92%]	[3, 0.32%]
	Opt.	[1, 5.42%]	[3, 2.95%]	[6, 1.49%]	[3, 0.64%]	[5, 0.18%]
Cache Size in Bits	Min.			[3, 435]	[2, 189]	[3, 375]
	Opt.	[1, 99]	[3, 465]	[6, 3683]	[3, 405]	[5, 1575]
Off-Chip Energy Ratio	Min.			[3, 1.23]	[2, 1.29]	[3, 1.21]
	Opt.	[1, 1.32]	[3, 1.20]	[6, 1.27]	[3, 1.23]	[5, 1.16]
On-Chip Energy Ratio	Min.			[3, 0.85]	[2, 0.93]	[3, 0.97]
	Opt.	[1, 0.91]	[3, 0.81]	[6, 0.85]	[3, 0.90]	[5, 0.94]
Miss Rate	Min.			[3, 0.20%]	[2, 0.19%]	[3, 0.10%]
	Opt.	[1, 0.42]	[3, 0.25]	[6, 0.20]	[3, 0.15]	[5, 0.08]
Comp. Ratio	Min.			[3, 0.48]	[2, 0.50]	[3, 0.49]
	Opt.	[1, 0.56]	[3, 0.48]	[6, 0.49]	[3, 0.48]	[5,0.48]

		Bus Width				
		20	24	28	32	36
Extra Cycle Penalty	Min.	[1, 0.06%]	[1, 0.00%]	[1,0.00%]	[1,0.00%]	
	Opt.	[8, 0.01%]	[5, 0.00%]	[9, 0.00%]	[3, 0.00%]	[1, 0.00%]
Cache Size in Bits	Min.	[1, 63]	[1, 51]	[1, 39]	[1, 27]	
	Opt.	[8,10731]	[5, 1071]	[9, 13299]	[3, 135]	[3, 15]
Off-Chip Energy Ratio	Min.	[1, 1.08]	[1, 1.00]	[1, 1.00]	[1, 1.00]	
	Opt.	[8,0.99]	[5, 1.01]	[9, 1.00]	[3, 1.00]	[1, 1.00]
On-Chip Energy Ratio	Min.	[1, 1.01]	[1, 1.00]	[1, 1.00]	[1, 1.00]	
	Opt.	[8, 0.99]	[5,1.01]	[9, 1.00]	[3, 1.00]	[9, 1.00]
Miss Rate	Min.	[1, 0.042%]	[1, 0.00]	[1, 0.00]	[1, 0.00]	
	Opt.	[8, 0.00]	[5,0.00]	[9, 0.00]	[3, 0.00]	[1, 0.00]
Comp. Ratio	Min.	[1, 0.55]	[1, 0.63]	[1, 0.74]	[1, 0.84]	
	Opt.	[8, 0.53]	[5,0.63]	[9, 0.74]	[3, 0.84]	[1, 0.94]

Table 4.1: **Extra Cycle Penalty, Optimal Index Widths, Cache Sizes, Bus Energy Ratios, Miss Rates, and Compression Ratios for Address Compression Using DBRC Scheme.** For a given bus width (column) and metric (rows), [A1, A2] means that A1 is the index width (minimum or optimal) and A2 is the value for the metric for that index width. For column corresponding to bus width=8, 10, and 36, the minimum and optimal values are the same. Hence only one is reported.

penalty to 0.3% higher values than those for the optimal index and experimentally determined the compression cache size, energy ratios, and miss rates. For the 16-bit example explained above, we found that the minimum index that can be used is only 3-bits. Thus the cache size can be reduced from 1575 bits (as in the optimal case) to only 375 bits—a

		Bus Width				
		8	10	12	14	16
Extra Cycle Penalty	Min.	[1, 4.98%]	[2, 2.34%]	[3, 1.62%]	[2, 0.83%]	[2, 0.46%]
	Opt.	[2, 4.91%]	[3, 2.34%]	[4, 1.61%]	[5, 0.63%]	[6, 0.27%]
Cache Size in Bits	Min.	[1, 136]	[2, 256]	[3, 480]	[2, 224]	[2,208]
	Opt.	[2, 272]	[3, 512]	[4, 960]	[5, 1792]	[6, 3328]
Off-Chip Energy Ratio	Min.	[1, 1.19]	[2, 1.07]	[3, 1.06]	[2,1.11]	[2, 1.11]
	Opt.	[2, 1.18]	[3, 1.08]	[4, 1.06]	[5, 1.06]	[6, 1.07]
On-Chip Energy Ratio	Min.	[1, 0.88]	[2, 0.82]	[3, 0.84]	[2,0.88]	[2, 0.94]
	Opt.	[2, 0.87]	[3, 0.82]	[4, 0.84]	[5, 0.87]	[6, 0.92]
Miss Rate	Min.	[1, 0.40]	[2, 0.28]	[3, 0.21]	[2,0.18]	[2, 0.12]
	Opt.	[2, 0.39]	[3, 0.28]	[4, 0.21]	[5, 0.15]	[6,0.10]
Comp. Ratio	Min.	[1, 0.54]	[2, 0.48]	[3, 0.47]	[2, 0.48]	[2, 0.50]
	Opt.	[2, 0.53]	[3, 0.48]	[4, 0.47]	[5, 0.47]	[6, 0.48]

		Bus Width				
		20	24	28	32	36
Extra Cycle Penalty	Min.	[1, 0.07%]	[1,0.00%]	[1,0.00%]	[1, 0.00%]	
	Opt.	[6, 0.01%]	[10, 0.00%]	[8, 0.00%]	[4, 0.00%]	[1, 0.00%]
Cache Size in Bits	Min.	[1, 88]	[1, 72]	[1, 56]	[1, 40]	
	Opt.	[6, 2816]	[10, 36864]	[8, 7168]	[4, 320]	[1, 24]
Off-Chip Energy Ratio	Min.	[1, 1.10]	[1, 1.00]	[1, 1.00]	[1, 1.00]	
	Opt.	[6, 1.00]	[10, 1.00]	[8, 1.00]	[4, 1.00]	[1, 1.00]
On-Chip Energy Ratio	Min.	[1, 1.01]	[1, 1.00]	[1, 1.00]	[1, 1.00]	
	Opt.	[6, 1.00]	[10, 1.00]	[8, 1.00]	[1, 1.00]	[1, 1.00]
Miss Rate	Min.	[1, 0.06]	[1, 0.00]	[1, 0.00]	[1, 0.00]	
	Opt.	[6, 0.00]	[10, 0.00]	[8, 0.00]	[4, 0.00]	[1, 0.00]
Comp. Ratio	Min.	[1, 0.56]	[1, 1.63]	[1, 0.74]	[1, 0.84]	
	Opt.	[6, 0.53]	[10, 0.63]	[8, 0.74]	[4, 0.84]	[1, 0.94]

Table 4.2: **Extra Cycle Penalty, Optimal Index Widths, Cache Sizes, Bus Energy Ratios, Miss Rates, and Compression Ratios for Address Compression Using BE Scheme.**For a given bus width (column) and metric (rows), [A1, A2] means that A1 is the index width (minimum or optimal) and A2 is the value for the metric for that index width. For column corresponding to bus width=36, the minimum and optimal values are the same. Hence only one is reported.

76.2% reduction. However, this may result in slightly worse energy ratios as observed from Table 4.1. For address compression using the BE scheme (Table 4.2), the corresponding cache size reduction for a 16-bit bus is from 3328 bits when using the optimal index size (6 bits) to 208 bits when using the minimum index of 2 bits—a 93.75% reduction. Other values

of extra cycle penalty can also be used to derive corresponding minimal index widths and similar trends as reported below will be observed.

Comparing across the two schemes for same bus widths, the following observations can be made. First, both schemes result in negligible performance penalty when address bus widths are reduced up to 20 bits from the original 38 bits. This results in immediate savings of 18 out of 38 (47.4%) bus lines and their associated buffers, repeaters, and receiving circuitry. For these savings, the compression cache size needed is also small (maximum of 63 bits). Thus, there will be net savings in area/cost even if the size of address compression hardware is taken into account. Reduction of bus width (beyond 20 bits) increases the extra cycle penalty for both DBRC and BE. Also, the energy ratios show a broadly decreasing trend as we move towards narrower bus widths. For these buses, BE results in greater energy reduction than DBRC for most of the bus widths we considered. Finally, compression cache miss rates for both schemes are roughly similar for all bus widths—they vary between 0% (very few misses) for larger bus widths to about 39% for narrow bus widths.

4.4.2 System performance and bus energy for fixed hardware costs

In the previous experiment, we set limits on how much extra cycle penalty can increase to determine the minimum index widths or cache sizes that can be used for various bus widths. In this experiment, we assume that the designer is allowed to use a compression cache in a given range of sizes thus fixing the hardware cost for address compression in this range. For seven different bus widths which represent different area/cost reductions of the address bus, we estimate the extra cycle penalty and energy ratios that result. For each bus width, we

consider compression cache sizes ranging from 4 to 2048 entries.

Performance penalty and miss rates

In Fig. 4.4, for narrower buses (e.g., a 12 bit bus), we observe that the extra cycle penalty first decreases as the number of entries is increased from 4 to 64 and then increases dramatically as we increase the number of entries to 2048. The reason for this is the following. A larger number of entries means more bits of the compressed bus need to be used for transmitting the index during a hit. Thus, a lesser number of bits of the bus can be used for the uncompressed low-order portion of the address word. Reducing the uncompressed portion, in turn, means increasing the compressed portion, which lowers the compression cache hit rate to some extent and this is the reason for this degradation in performance. For slightly wider buses (e.g., 14 and 16-bit buses), the number of entries in the compression cache becomes less critical to the performance than the narrow bus does as our results show. This is because, for the same compressed portion of an address word, the increased bus width allows more entries in the compression cache, which can reduce the miss rate. Also, even in the case of miss, the wider bus facilitates transfer of the missed address in fewer cycles and hence the miss penalty becomes smaller. For even wider buses (24-bits or more), the uncompressed part is relatively large across different compression cache sizes, so varying the compression cache size does not have any impact on the performance.

Another observation from Fig. 4.4 is that DBRC performs better than BE when smaller number of entries (notably four and eight entries) are used and this trend is true for all bus widths. The reason for this is the following. As mentioned in Sec. 4.2, the miss penalty for

Extra Cycle Penalty Variation Across Different Compressed Bus Widths and Compression Cache Sizes

[Min. Extra Cycle Peanlty, Opt. Index]	8	12	14	16	20	24	28
DBRC	[5.420%, 1]	[1.295%, 5]	[0.643%, 3]	[0.178%, 5]	[0.009%, 8]	[0.004%, 1]	[0.004%, 1]
BE	[4.907%, 2]	[1.615%, 4]	[0.634%, 5]	[0.274%, 6]	[0.007%, 6]	[0.001%, 10]	[0.004%, 1]

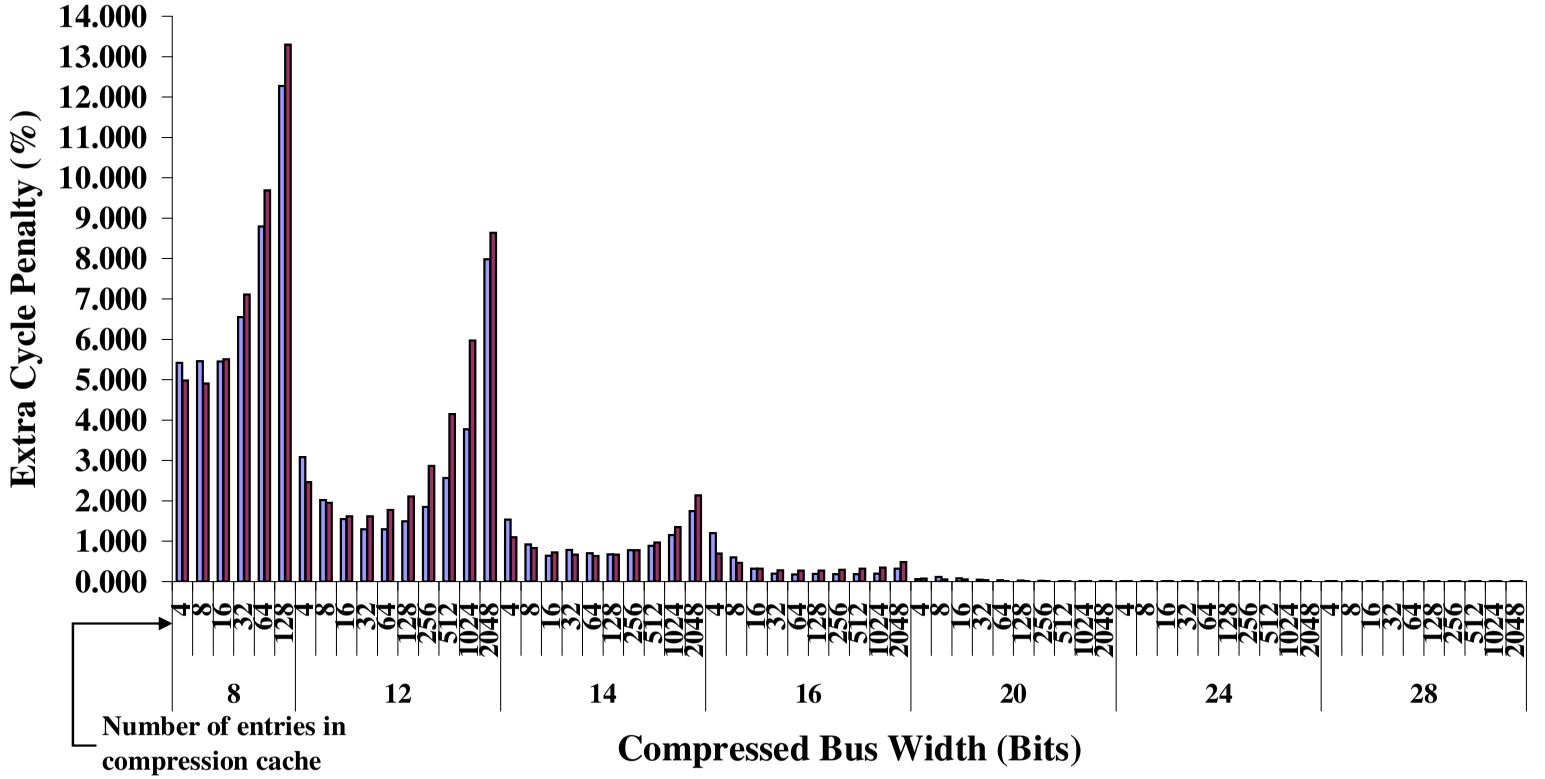


Figure 4.4: **Extra Cycle Penalty for DBRC and BE for Different Compression Cache Sizes.**

BE will be less than or equal to that for DBRC for the same bus width since the former uses a single bit and the latter uses a longer reserved bit-pattern to indicate a miss. However, for a given bus width and also fixed number of entries in the compression cache, the compressed portion for BE is one bit wider than that for DBRC due to the control bit, so the extra cycle penalty in the case of BE can become higher due to increased miss rate leading to worse performance than DBRC.

Bus energy dissipation

Off-chip bus energy ratios are reported in Fig. 4.6. From this plot, we observe that BE consumes less energy on the average than DBRC for most bus widths—average results are shown in the table on the top-left corner of each plot. BE is more energy-efficient because it has lesser miss penalty than DBRC for the same bus width. But, for a bus width of 24 bits or greater, not much off-chip energy savings can be obtained with both address compression designs. This is because, for wider buses, the uncompressed part has more bits and the compressed part has a lesser number which leads to low miss rate (smaller than 0.002%) for both schemes. So most bus lines are used for the uncompressed portion and hence the bit pattern of the compressed address word is similar to the bit pattern of uncompressed original address word.

Figs. 4.7 and 4.8 show on-chip energy ratio, including the contribution of self-capacitance energy and components of coupling energy across different number of entries in compression cache, for four bus widths. These are shown across two plots: bus widths 12 and 14 in Fig. 4.7 and 16 and 24 in Fig. 4.8. From these figure, it can be observed that energy savings

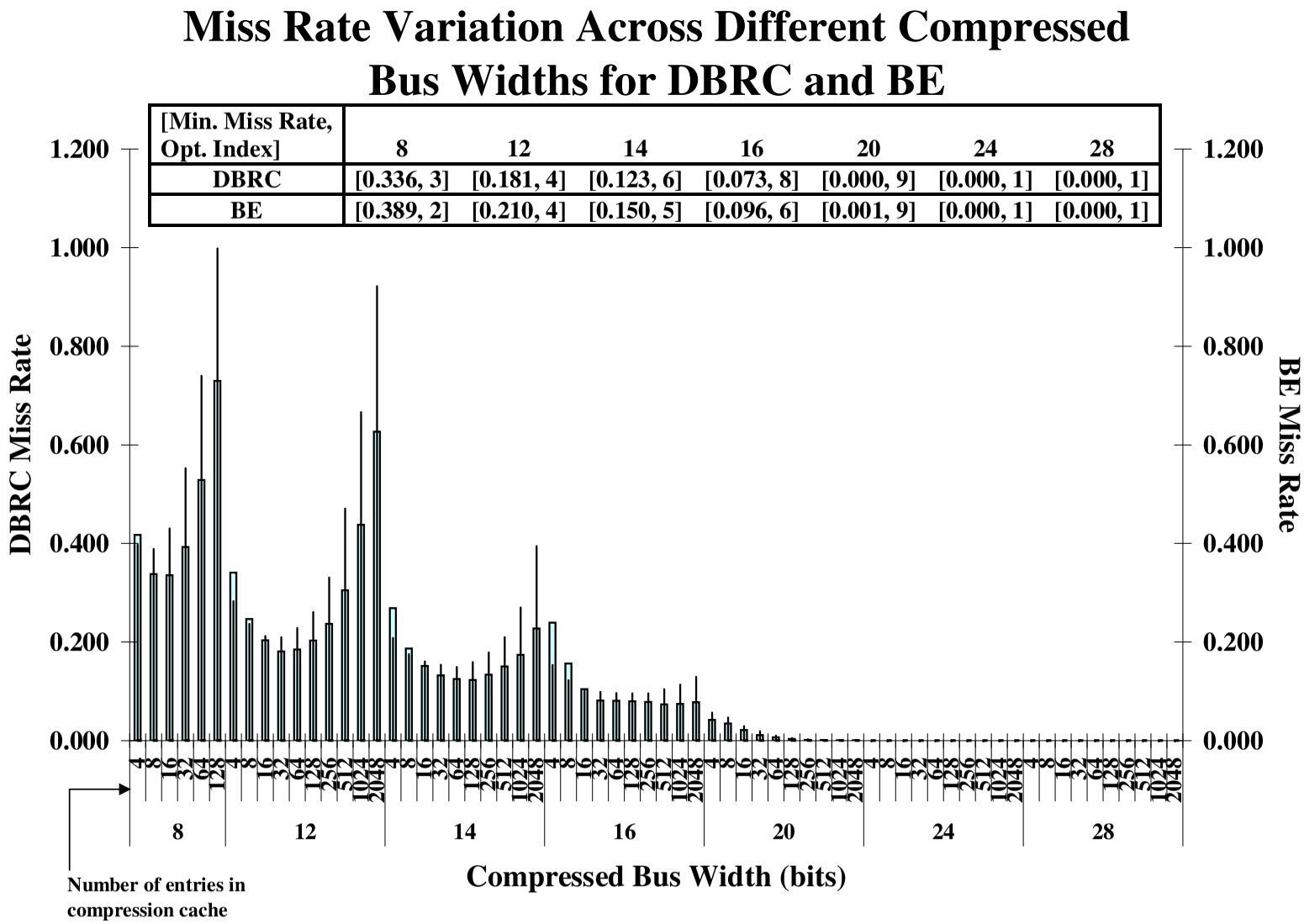


Figure 4.5: **Miss Rate for DBRC and BE for Different Compression Cache Sizes.**

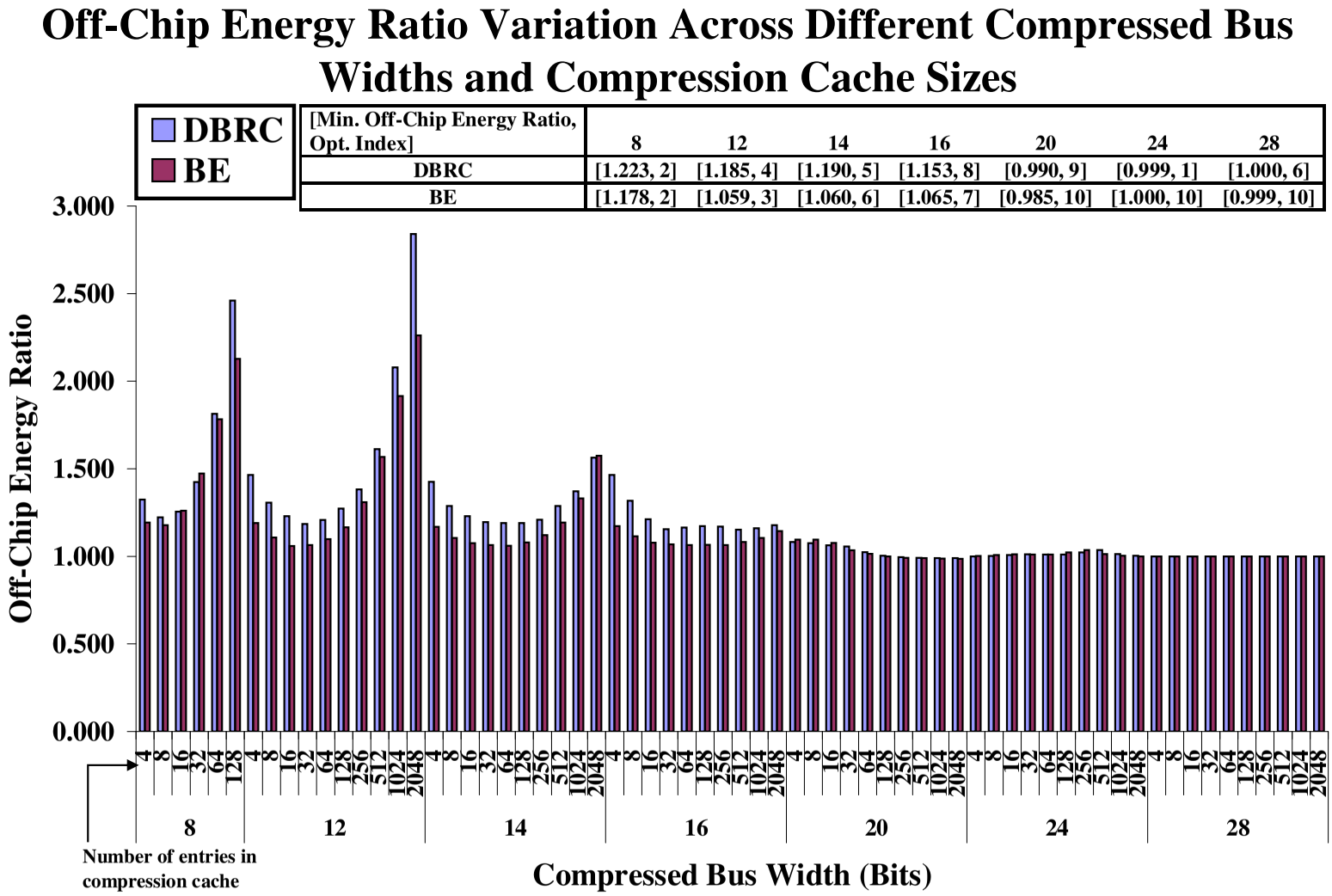


Figure 4.6: **Influence of Compression Cache Size on Off-Chip Bus Energy Dissipation:** Off-chip bus energy dissipation ratio for DBRC and BE for different compressed bus widths.

obtained with address compression in on-chip buses is more savings for off-chip buses. It can also be seen that, across bus widths, most of the energy saving is due to reduction in toggle energy and across different compression cache sizes in the same bus width, the savings—which are better when smaller cache sizes are used—are due to reductions in coupling charge and discharge energies. Also, similar to what was observed in Sec. 4.4.2 for performance, BE resulted in a worse energy ratio compared to DBRC when smaller compression caches (number of entries of 4 and 8) are used. The reason for this is also the same as that described in Sec. 4.4.2. Finally, we also observe that, similar to trends in off-chip energy dissipation, the wider the compressed bus width is, the less energy can be saved with dynamic address compression.

4.4.3 Influence of technology parameters on energy efficiency

To study the energy-efficiency of address compression as technology scales down, we plotted energy ratios as a function of λ, the ratio of coupling capacitance to the self-capacitance of a wire. The parameter λ takes values of approximately the following for topmost layer interconnects in current and future nanometer technologies according to [28]: 2.08 for 130-nm, 2.34 for 90-nm, 2.73 for 65-nm, and 3.05 for 45-nm. For future nanometer technologies, the values of λ are bound to increase further. From the plot shown in Fig. 4.9, we observe that address compression improves energy efficiency of compressed address buses even as technology scales down. For current technology (130-nm), the reduction is about 10% on the average for most buses.

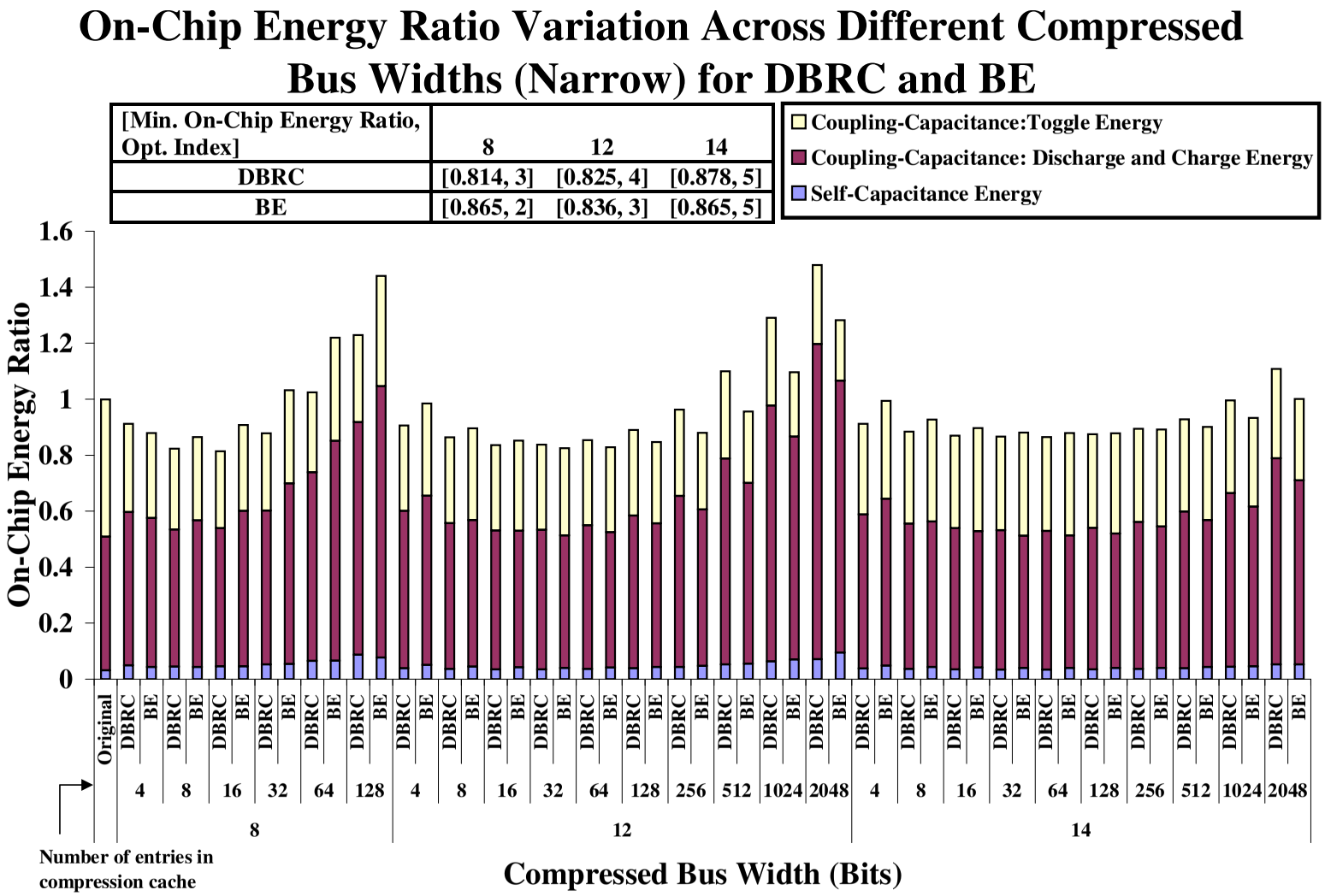


Figure 4.7: **Influence of Compression Cache Size on On-Chip Bus Energy Dissipation:** On-chip bus energy consumption ratio for DBRC and BE for different compression cache sizes. (Narrow Bus)

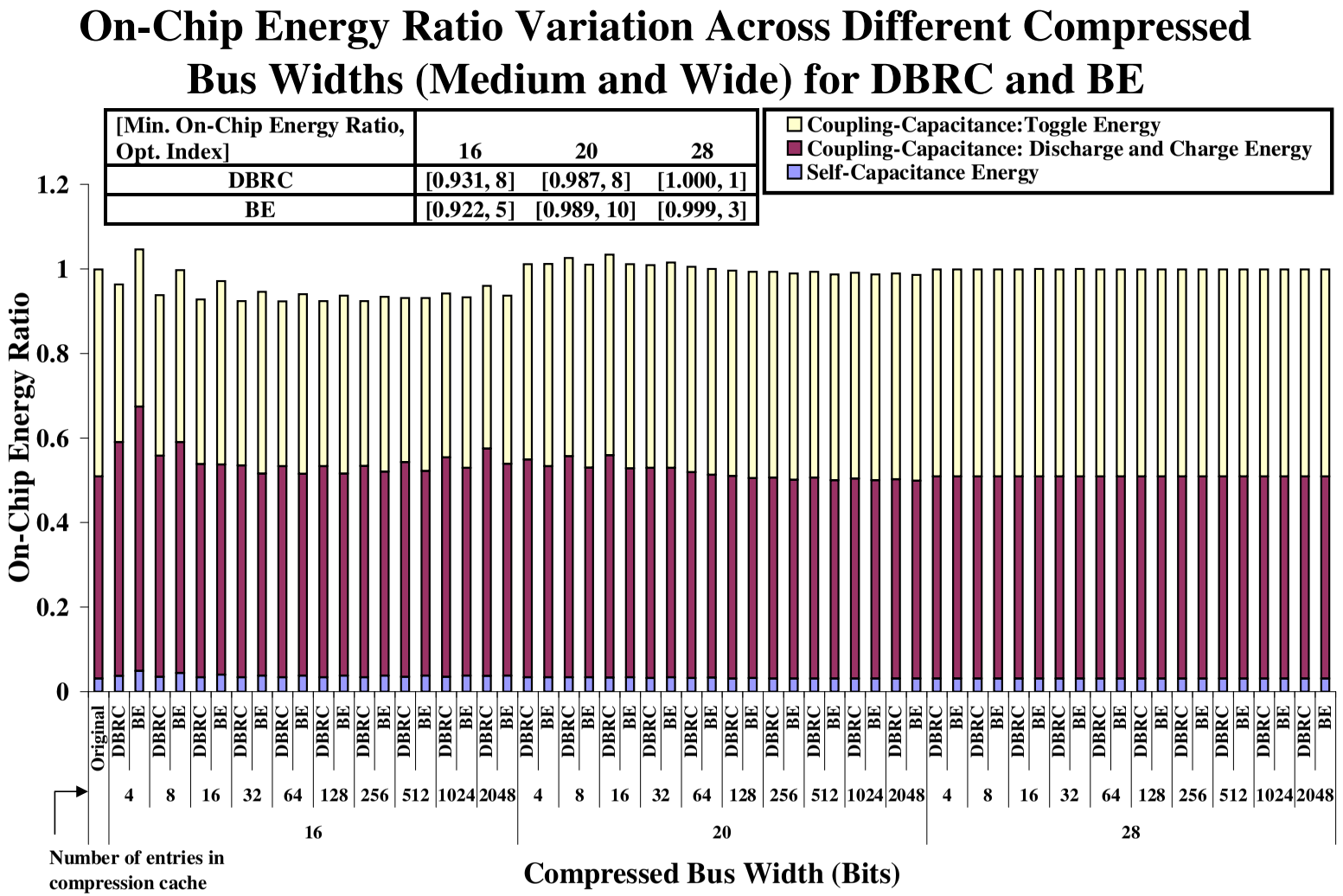


Figure 4.8: **Influence of Compression Cache Size on On-Chip Bus Energy Dissipation:** On-chip bus energy dissipation ratio for DBRC and BE for different compression cache sizes. (Wide Bus)

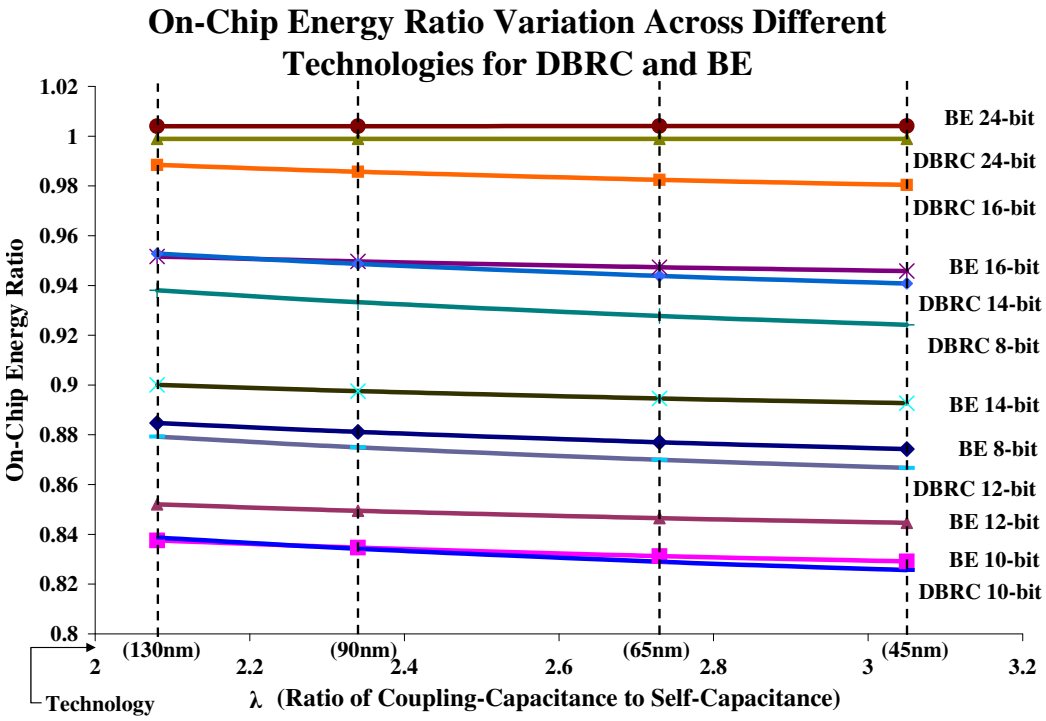


Figure 4.9: **Energy Reduction in Compressed Address Buses for Different Technologies.** This plot shows the effect of technology on compressed address buses of various widths.

4.4.4 Influence of extra compression/decompression latency

In this experiment, we examine how compression/decompression latency affects system performance. Previously, we assumed that there is no extra compression or decompression latency at the sending and receiving ends respectively. We observed that the extra cycle penalty was large (about 5%) for small bus widths (8-bit and 12-bit bus) and dropped off rapidly for bus widths beyond 16 bits. This trend can be observed in Fig. 4.10 too for compression/decompression latency of zero cycles. Realistically, compression may take a longer time—if complex schemes are used for compression cache lookup and/or multiplexers are used to rearrange bits to ensure energy-efficient transmission—than decompression which involves only register access but only if a hit occurs at the sender compression cache. Hence

we consider two cases in this experiment: (1) compression takes one extra clock cycle and decompression takes zero cycles, and (2) compression takes one cycle and decompression takes one cycle only if a hit occurs in the sender compression cache. In the second case, a miss is assumed to cause no extra decompression latency because there is no need for a register file access to decompress the address. Rather, the multiple cycles needed for transmission of the missed address already takes into account the latency for a miss.

From results shown in Fig. 4.10, we observe that the extra cycle penalty does not drop off rapidly with bus width as in the default case (both compression and decompression occur in zero cycles). This is because the hit-rate of the compression cache with wider buses becomes better which necessitates one extra cycle for decompression each time a hit occurs. We also observe that pipelining the address bus helps reduce the extra cycle penalty to some extent but the effect is not much since L1→L2 address references are spaced more or less randomly apart in time; bus pipelining yields best results when references on the bus occur continuously spaced apart by a fixed time gap.

From all the above results we observe that both DBRC and BE have similar trends and in most cases, the results for BE are slightly better. Hence in the rest of the chapter, we report results only for BE.

4.4.5 Influence of virtual→physical address translation

In a realistic system, a page can be placed anywhere in the available memory by the operating system (OS) and hence the higher order part (page number) of the L1→L2 physical address–which is obtained using the hardware translation look-aside buffers (TLB)—can

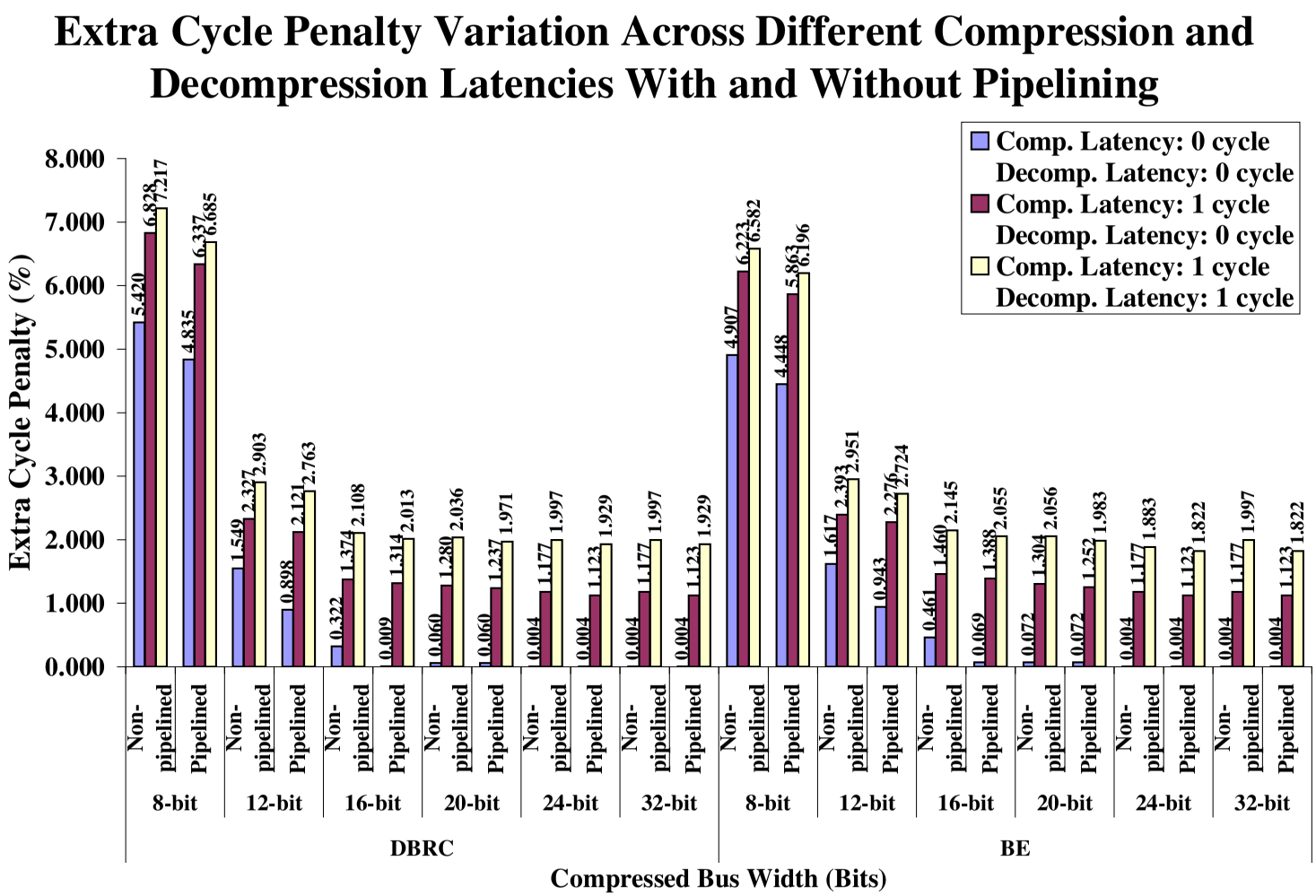


Figure 4.10: **Influence of Compression/Decompression Latency on Performance with and without Address Bus Pipelining.**

vary quite significantly. But this variation is not captured realistically in the direct page table translation mechanism that is used in sim-alpha. It uses the value from a sequential counter—which is incremented whenever a new page is loaded—as the physical page number and places the mapping information (physical and virtual page numbers) in the page table at the location pointed to by the virtual page number. Due to this, higher order parts of physical addresses issued in sim-alpha are likely to have more sequentiality characteristics than in a realistic system where contiguous entries of the page table are not likely to belong to the same program.

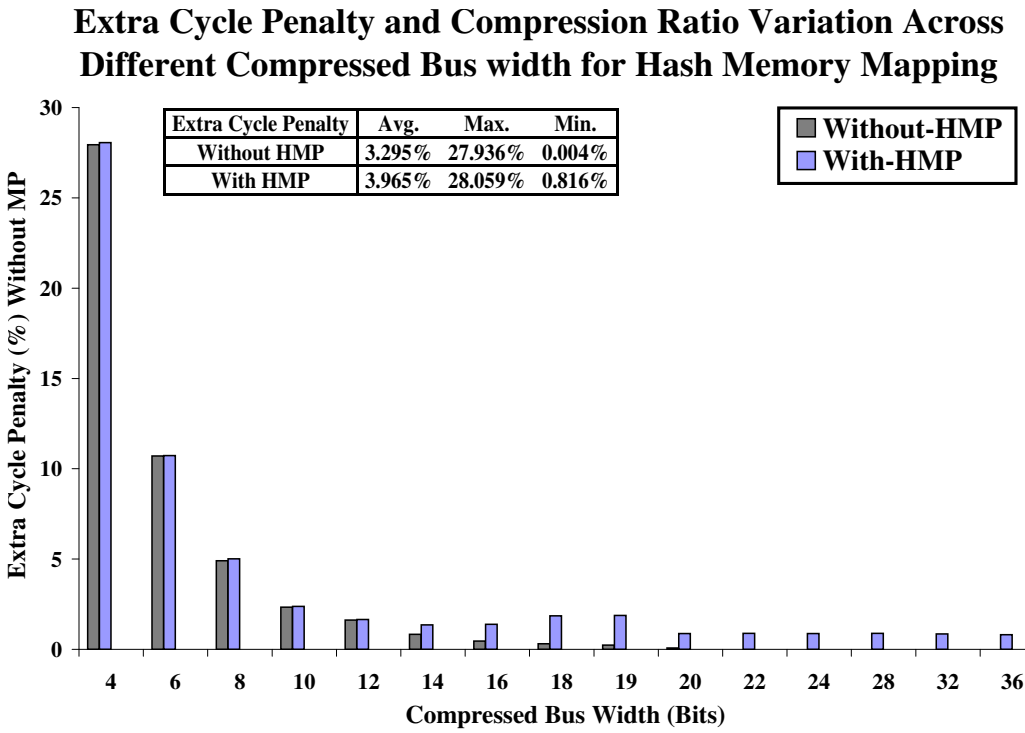


Figure 4.11: **Influence of Different Virtual→Physical Address Mapping Schemes on Performance.**

We study the effect of a more realistic scenario by implementing a hashing mechanism to randomize the physical page number of a newly loaded page. Note that, in an actual system,

available memory constraints dictate the physical page number and no hash mappings are used. Given the number of bits used for the physical page number (PB), the virtual page number (VN), and $\Phi = \frac{\sqrt{5}-1}{2}$; the physical page number (PN) using our mapping scheme is found by using:

$$PN = 2^{PB} \times (VN \times \Phi - \lfloor VN \times \Phi \rfloor).$$

Both instruction and data addresses are mapped using this scheme. Note that we have not changed the way TLBs operate but some extra TLB misses may be caused due to the new mapping mechanism. Our simulations in which we measure the extra cycle penalty also include this overhead, if any. Results for simulations using the default mapping scheme and our hash mapping scheme are shown in Figs. 4.11, 4.12, and 4.13.

As we observe in Fig. 4.11, the extra cycle penalty increases when new mapping is applied since the number of different patterns in the higher order portion of the addresses increases and consequently the miss rate of the compression cache is higher than before. However, this increase is small, only 0.7% on average but the energy savings are much more, especially for off-chip buses. From Figs. 4.12 and 4.13 we observe that, on the average, the off-chip and on-chip energy savings are improved by 0.271 and 0.009, respectively. These results show that address compression will be more effective in real systems even if different virtual memory organizations and/or mapping schemes are used.

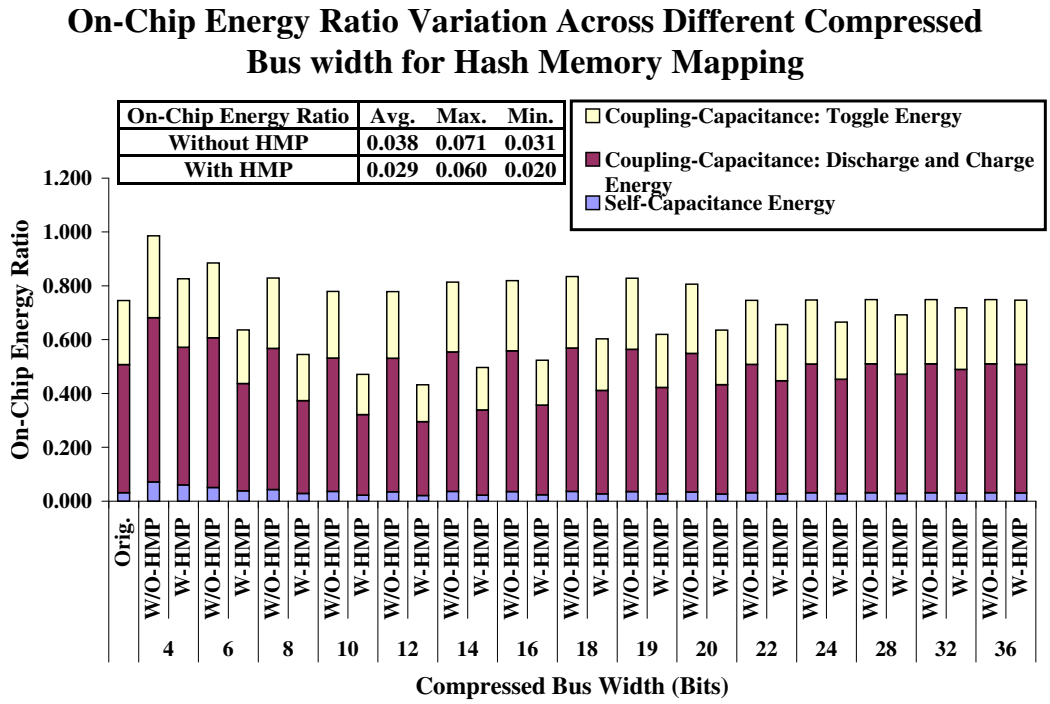


Figure 4.12: **Influence of Different Virtual→Physical Address Mapping Schemes on On-Chip Energy.**

4.4.6 Influence of compression cache set associativity and replacement policy

If the number of bits in the tag field and the number of entries in the compression cache are fixed, then the miss rate of the cache will change with change in set associativity. Thus, higher set associativities can reduce both extra cycle penalty and improve energies primarily due to an increase in the number of hits when shorter compressed addresses can be transferred in a single cycle. We studied the effect of different configurations: direct-mapped, 2-way, 4-way, 8-way, and fully-associative caches on extra cycle penalty and the results are shown in Figs. 4.14, 4.15, 4.16. Here, we use a compression cache with 16 entries which enables us to study the following range of bus widths: 8-bit, 10-bit, 12-bit, 16-bit, and 24-bit.

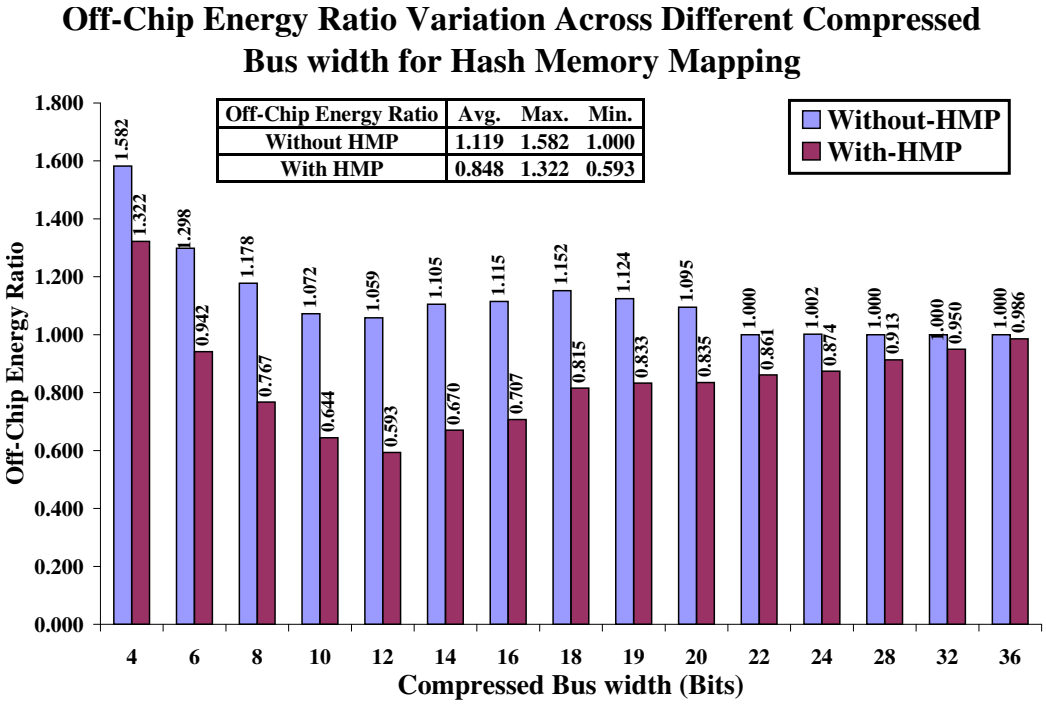


Figure 4.13: **Influence of Different Virtual→Physical Address Mapping Schemes on Off-Chip Energy.**

From our results, we observe that, across most bus widths, extra cycle penalties and energies reduce when set associativity increases from direct-mapped to fully-associative. Extra cycle penalties also reduce as the buses become wider because of reducing miss rates as described in earlier results. With fully-associative compression caches, on-chip energy ratios reduce by as much as 8% or so compared to direct-mapped caches and extra cycle penalties reduce by a few tenths of a percent. Thus, using fully associative compression caches, particularly since the number of entries is small can be very effective in address compression.

We also studied how different replacement policies will influence extra cycle penalty and energies in address compression caches. We tested the well-known first-in-first-out (FIFO) and least-recently-used (LRU) policies and a modified LRU policy recently proposed in [30].

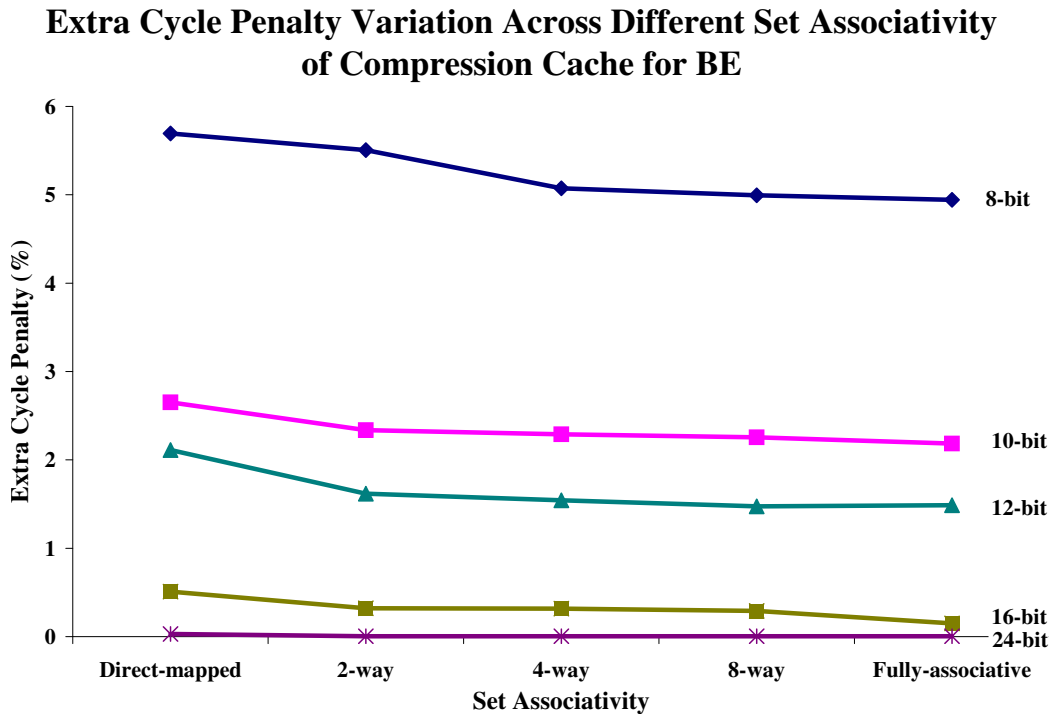


Figure 4.14: **Influence of Varying Compression Cache Set Associativity on Performance and Energy:** Extra cycle penalties are the least for fully-associative caches.

In MLRU, if a block was not accessed soon after it was brought into the cache, then it is replaced. This helps eliminate one-touch references. To implement MLRU, before an entry in a set is replaced, we update the value of the LRU counter with the value LRU_{new} as follows:

$$LRU_{new} = L + 0.25 \times N,$$

where L is the LRU counter value for the least recently accessed entry in the set and N is the total number of entries is a set. Thus, in our implementation, the incoming entry is assigned an LRU value 25% above the currently least recently used entry. Our results shown in Figs. 4.17, 4.18, 4.19, and 4.20 indicate that the basic LRU scheme performs the best in terms of both performance (extra cycle penalty and miss rates) and energies. Higher order portions of addresses which are stored in our compression cache tend to be more similar—

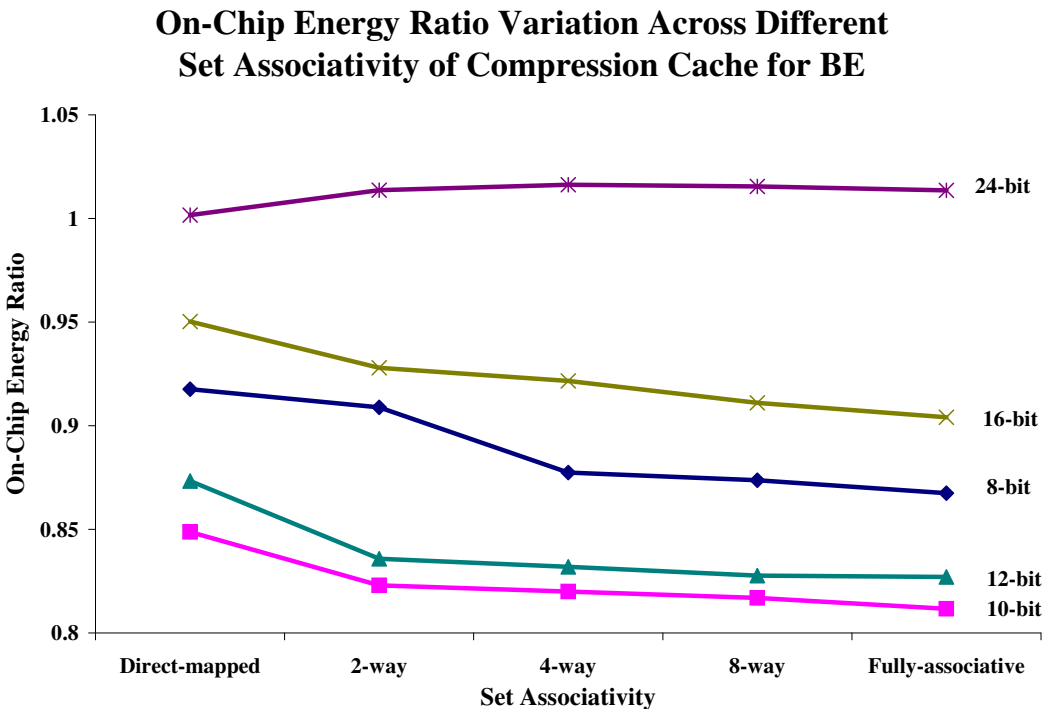


Figure 4.15: **Influence of Varying Compression Cache Set Associativity on Performance and Energy:** For most bus widths fully-associative caches also result compressed addresses that dissipate least energy during transmission.

the same (recently used) entry will be accessed again—and one-touch references are highly unlikely to occur. Hence MLRU and FIFO perform worse.

4.4.7 Influence of L1 cache size

The L1 cache size may also potentially affect the density of L1→L2 address traffic and hence the performance and energies when address compression schemes are used. We experimented with seven different L1 cache sizes with 2-way set-associativity and the results are shown in Figs. 4.21, 4.22, and 4.23 for six different compressed bus widths: 12, 14, 16, 19, 20, and 24 bits. The results show that, as expected, the extra cycle penalty decreases for larger L1 cache sizes since the address traffic reduces and the references are also spaced

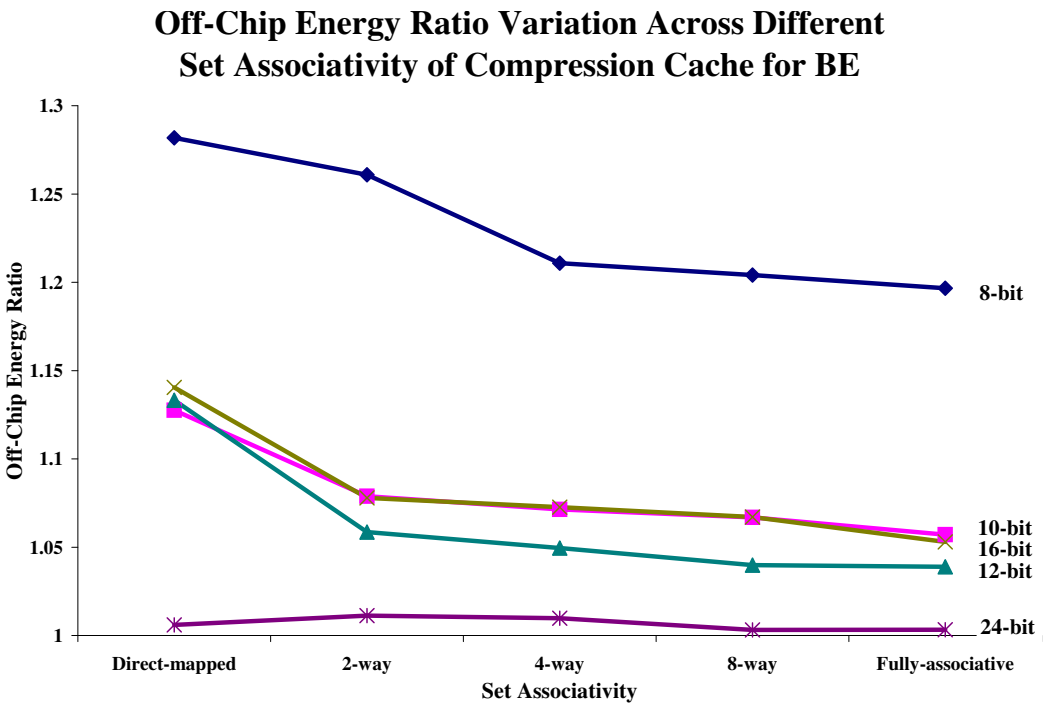


Figure 4.16: **Influence of Varying Compression Cache Set Associativity on Performance and Energy:** Off-chip bus energies also reduce when the associativity increases.

wider apart. For energy dissipation in on-chip buses shown in Fig. 4.22, most wider buses (19, 20, and 24 bits) show an increase in bus energy dissipation with larger L1 cache sizes. This may be because L1 cache misses that occur are spaced wider apart in time and hence these miss addresses are likely to be dissimilar resulting in more switching transitions. For off-chip buses a similar trend—some buses becoming more energy-efficient and others becoming less—can be observed.

L1 miss address buffer

In a pipelined processor, misses in the L1 caches are often buffered in a miss address file (MAF) and dependent instructions are not allowed to advance to the next stage until the miss is serviced. This delay adds to the overall program execution time but, in some cases, it may

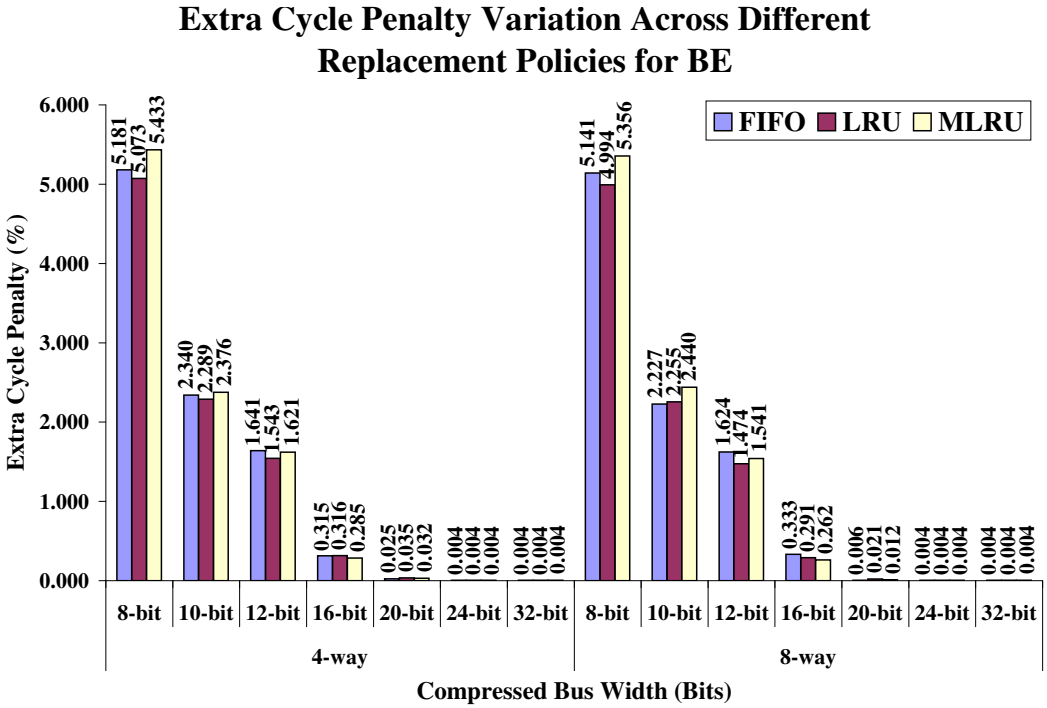


Figure 4.17: **Influence of Varying Compression Cache Replacement Policy on Performance.**

also lead to stalling of the pipeline. In our target system, we have assumed that the address compression hardware takes missed addresses from the MAF, compresses it and transmits on the bus in a single cycle (if the compression cache has a hit) and in multiple cycles (if the compression cache misses). Increasing the size of the MAF can, to some extent, offset the extra latency due to misses in the compression cache. We experimented with six different MAF sizes: 8, 16, 32, 64, 128, and 256 and three L1 cache sizes since lower miss rates for larger cache sizes will also reduce the need for larger MAF buffer.

Results for this experiment with five different bus widths are shown in Figs.4.24, 4.25, and 4.26. In Fig.4.24, we observe that, for all bus widths and L1 cache sizes, increasing the MAF size from 8 to 16 entries reduces the extra cycle penalty by a small amount but further

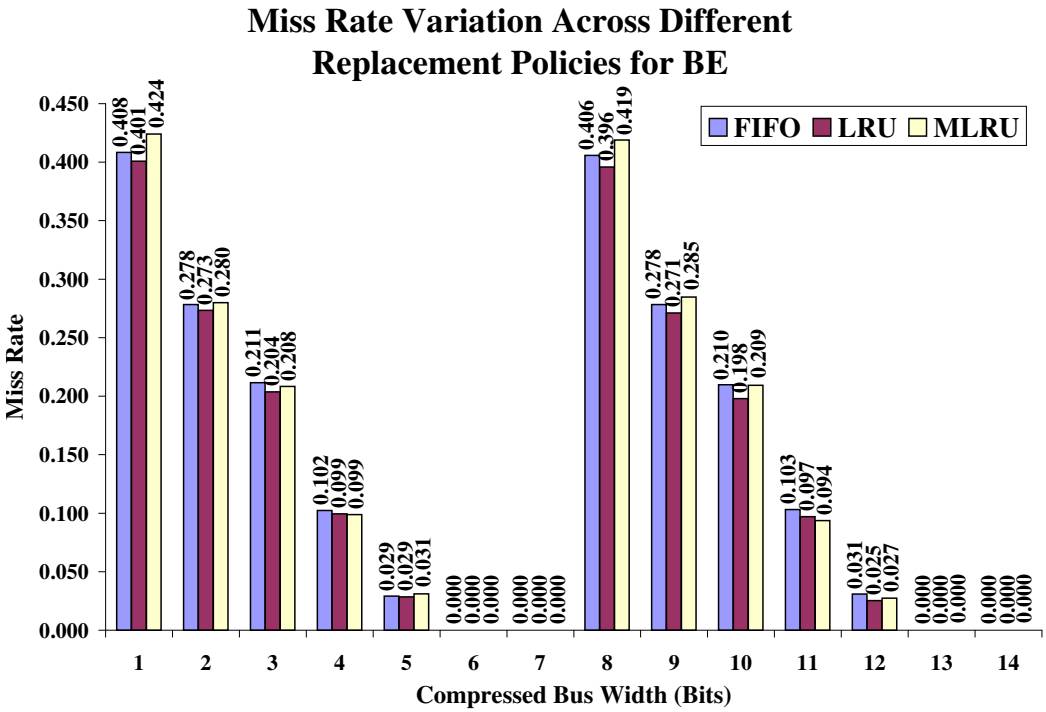


Figure 4.18: **Influence of Varying Compression Cache Replacement Policy on Miss Rate.**

increase in the MAF does not result in much benefits for most buses. Also, even for small L1 cache sizes (16KB), increasing the MAF size to 16 entries with a compressed address bus 24 bits wide, results in a net reduction in program execution time as seen from the negative value of extra cycle penalty. If L1 cache size is increased to 64KB, the percentage reduction in execution time also increase. Thus, address compression can actually reduce program execution time for a modest increase in the size of the MAF.

4.4.8 Address compression across memory system levels

In our previous experiments, address compression was applied to only the L1→L2 bus. In this experiment, we apply compression on: (1) L2→M address bus separately, and (2) in combination with L1→L2 address bus. Results are shown in Figs. 4.27 and 4.28. From

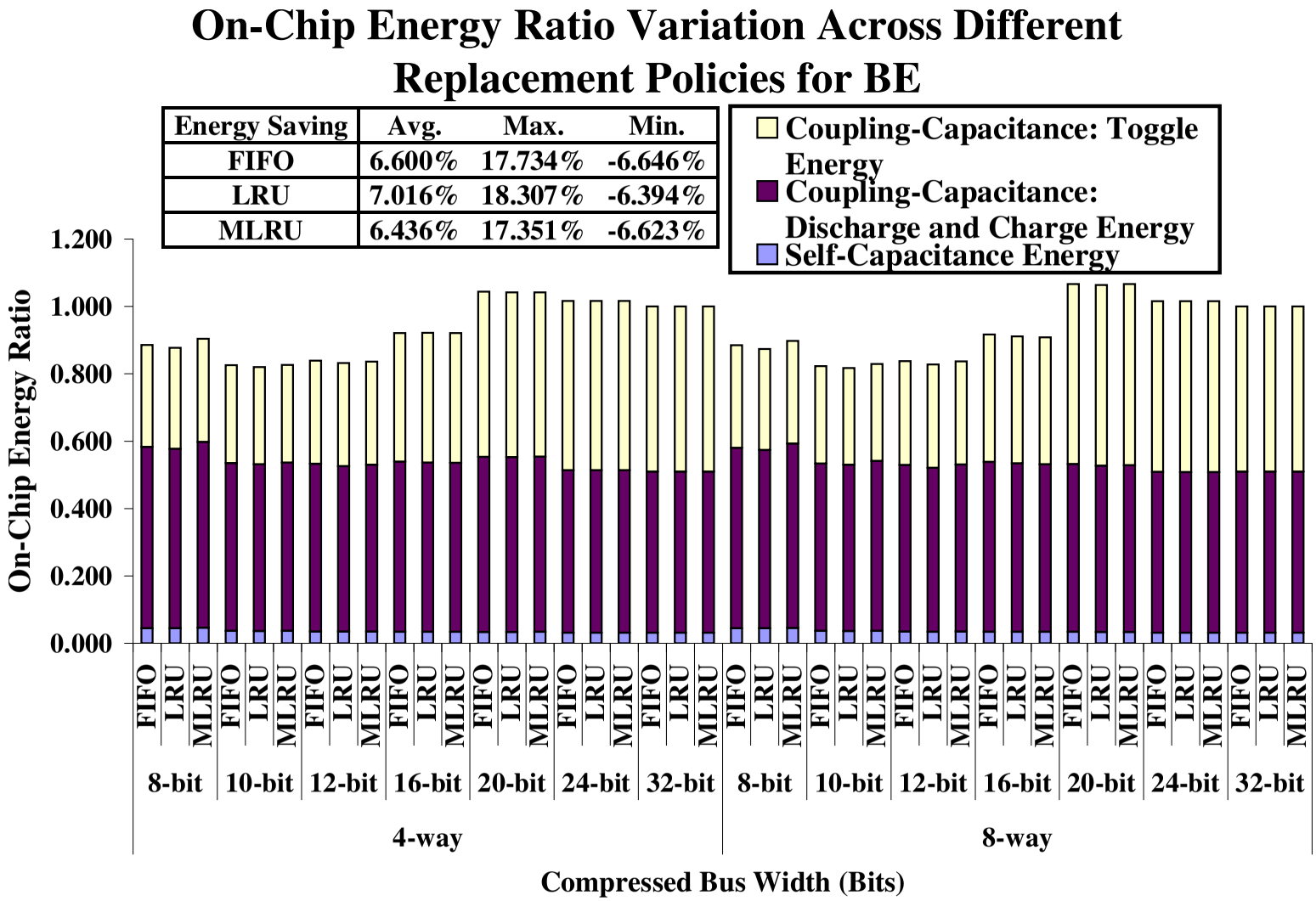


Figure 4.19: **Influence of Varying Compression Cache Replacement Policy on On-Chip Energy.**

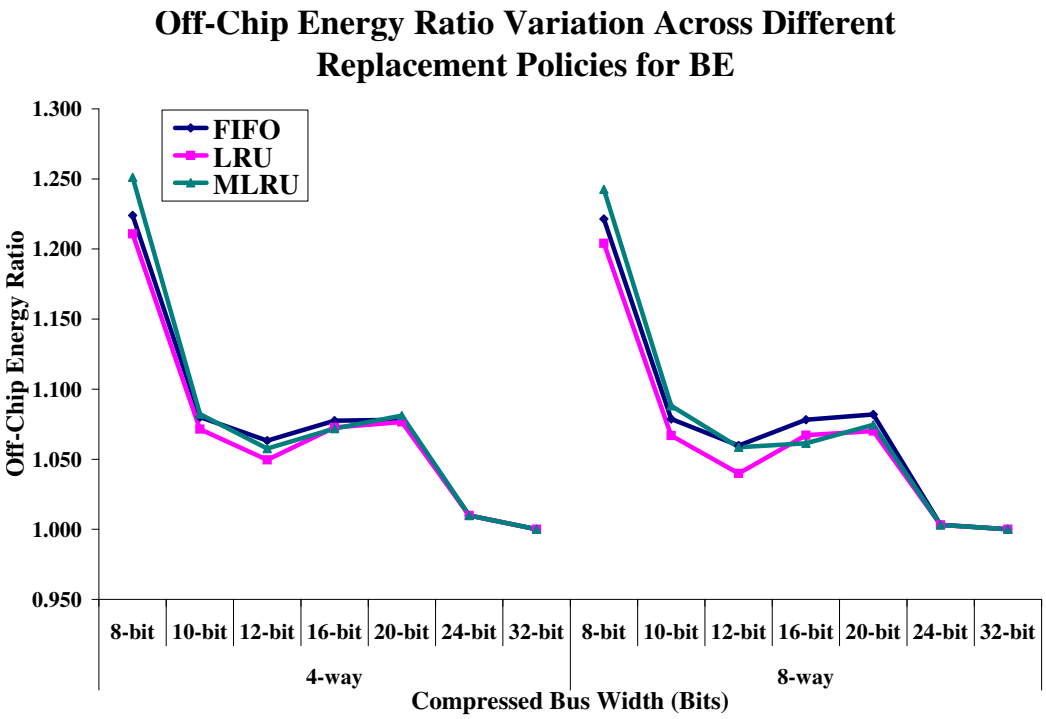


Figure 4.20: **Influence of Varying Compression Cache Replacement Policy on Off-Chip Energy.**

Fig. 4.27, we observe that the extra cycle penalty when the 38-bit L2→M address bus is compressed to 8 bits is only 1.65% and similar to the trend observed earlier for the L1→L2 bus, the penalty begins to drop off rapidly for larger bus widths. Also, the trend is similar when both L1→L2 and L2→M buses and addresses are compressed, but higher penalties are incurred. It can also be noted that the combined penalty is slightly less than the penalties when the buses are compressed individually. From Fig. 4.28, we observe that compressing L2→M addresses slightly worsens bus energy except for wider buses (24 and 32 bits) in which case the bus becomes energy-efficient to a small degree.

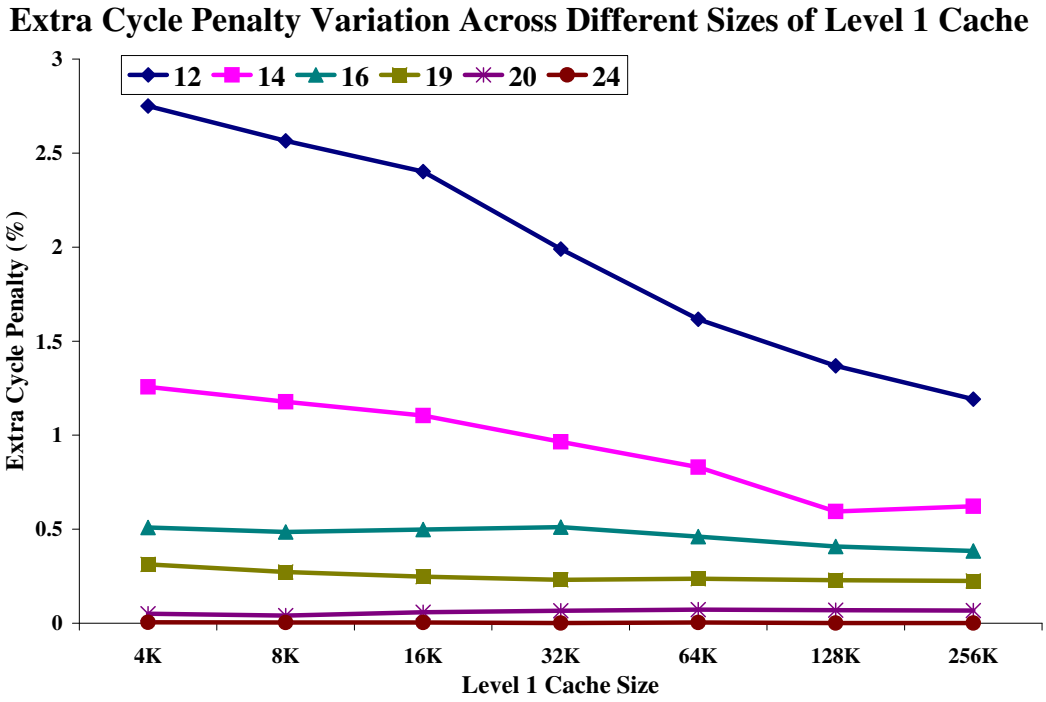


Figure 4.21: **Influence of Varying L1 Cache Sizes on Performance.**

4.5 Conclusions

In this chapter, we comprehensively analyzed system performance, bus energy dissipation, and cost savings (due to reduction in number of bus lines and associated hardware) when address compression schemes like dynamic base register compression or bus expander are applied to the L1→L2 level address bus. With simulations using a cycle-accurate simulator for fourteen SPEC CPU2000 benchmarks, we found optimal compression cache sizes that results in minimum extra cycle penalty, for each of these schemes, and for a wide range of compressed bus widths. We also reported energy savings, compression cache miss rates, and address compression ratios for the optimal configurations. We showed that aggressive bus-width reduction (as much as 63%, for example) will result in only an extra cycle penalty of about 1% or less and that energy dissipation in address buses will reduce appreciably (up

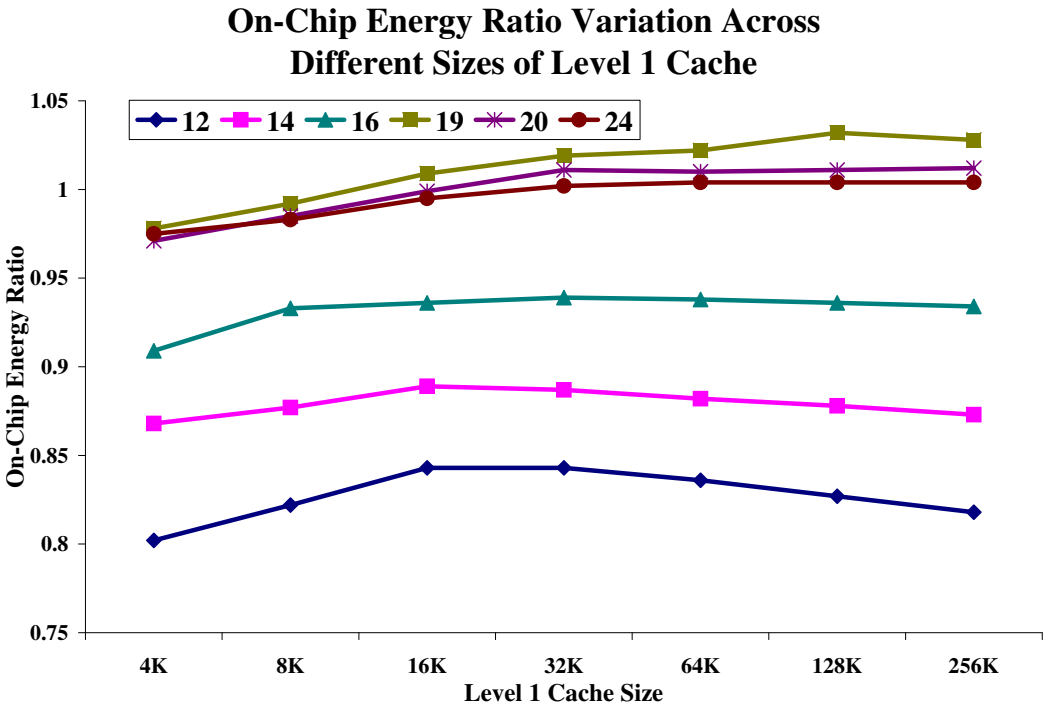


Figure 4.22: **Influence of Varying L1 Cache Sizes on On-Chip Energy.**

to 13%) with compression for current technologies. These savings were found to increase for future nanometer technology nodes.

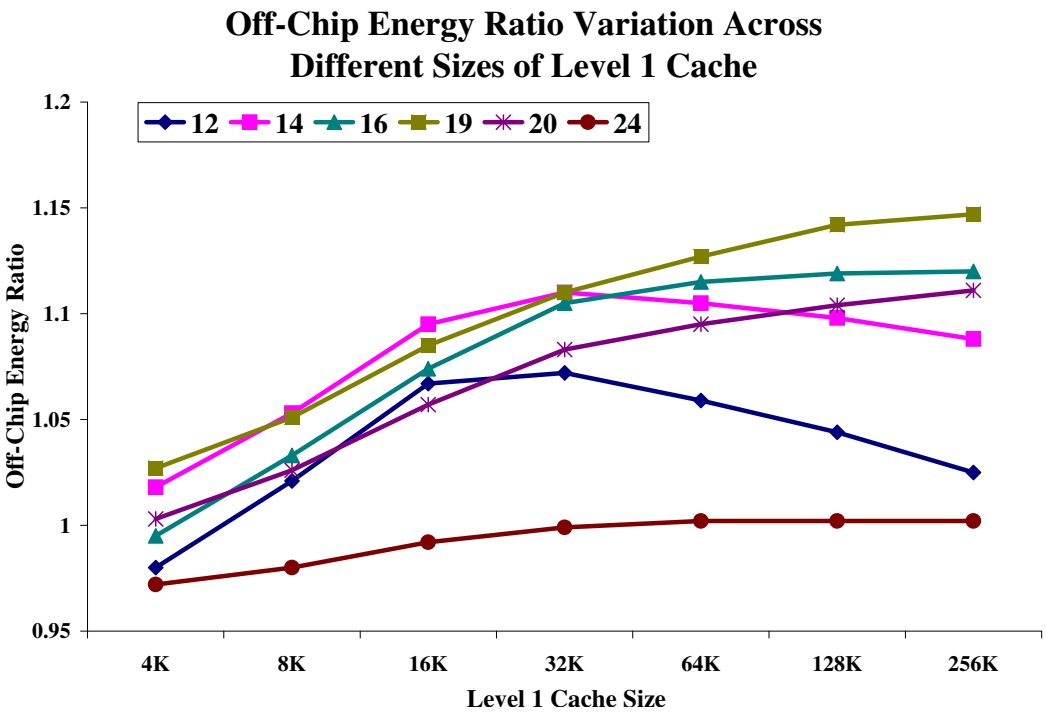


Figure 4.23: **Influence of Varying L1 Cache Sizes on Off-Chip Energy.**

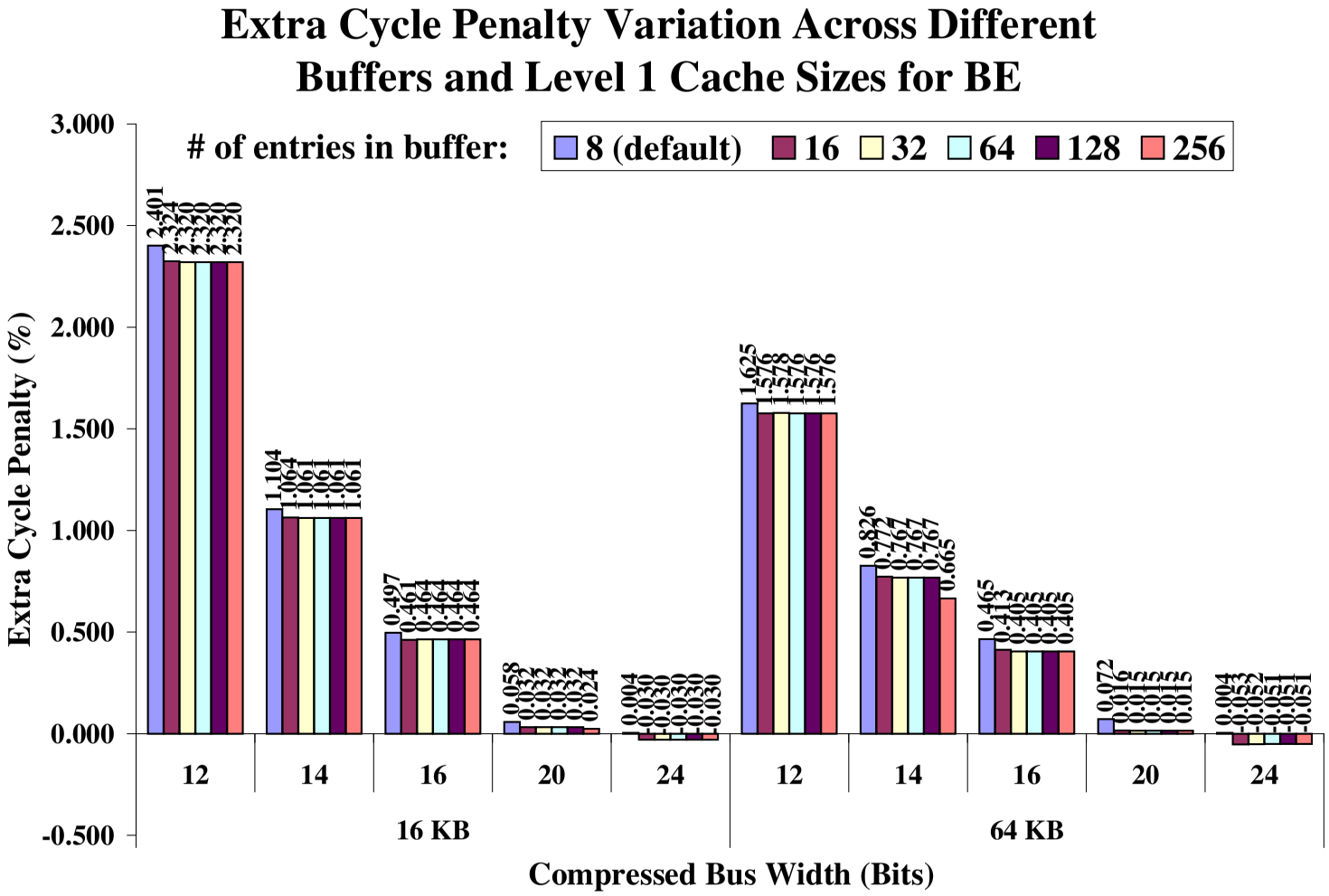


Figure 4.24: **Influence of Varying L1 Cache and Buffer Sizes on Performance.**

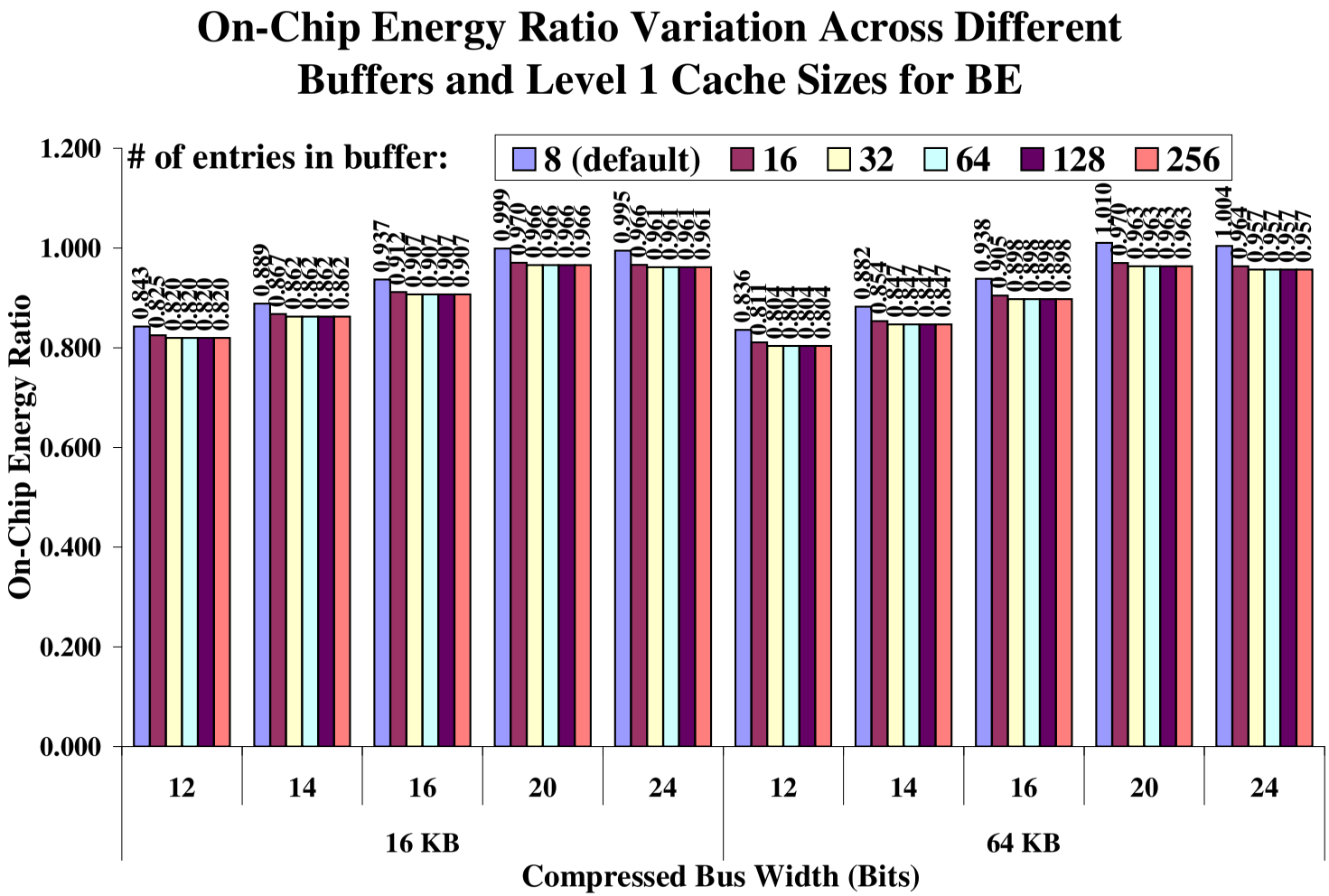


Figure 4.25: **Influence of Varying L1 Cache and Buffer Sizes on On-Chip Energy.**

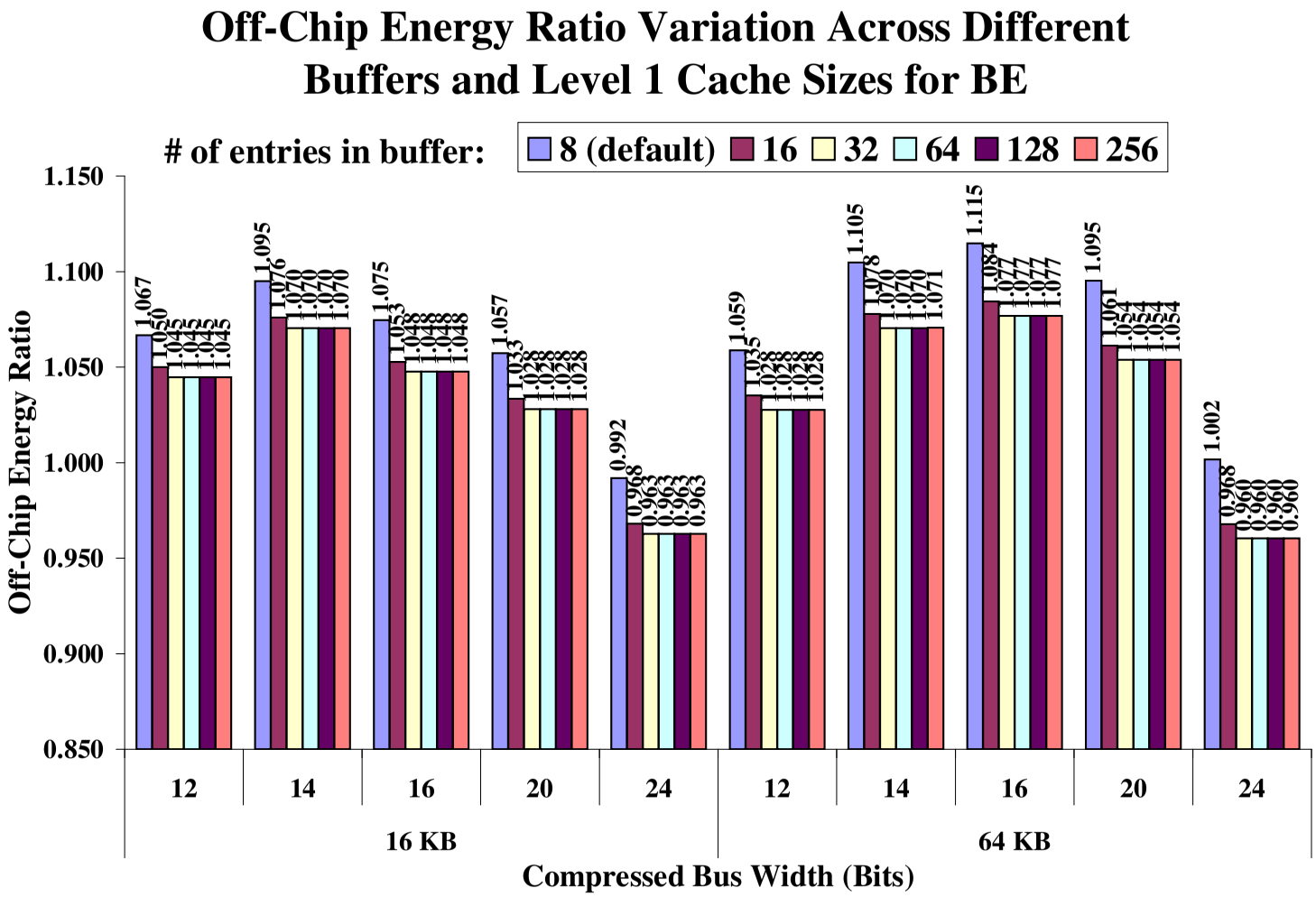


Figure 4.26: **Influence of Varying L1 Cache and Buffer Sizes on Off-Chip Energy.**

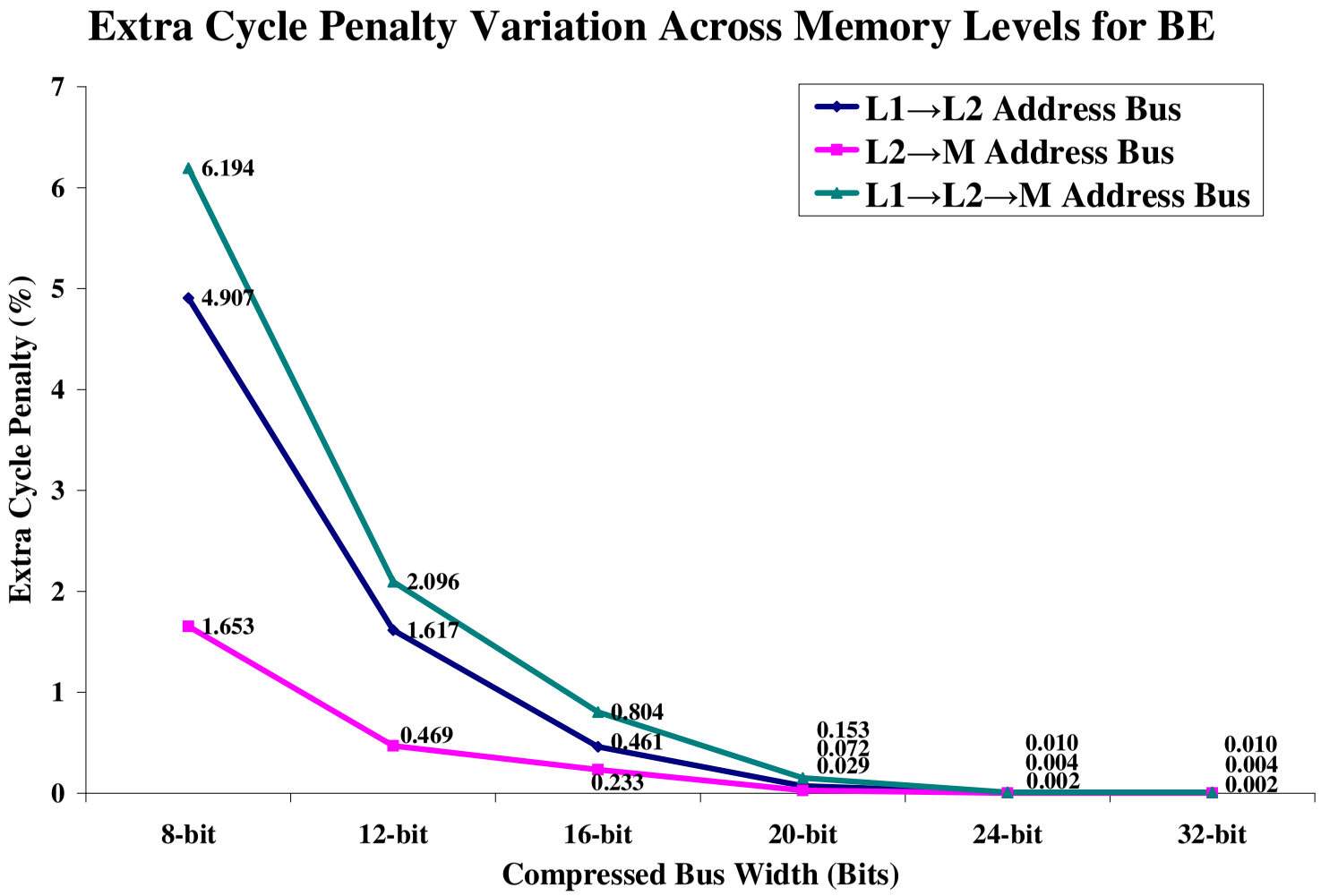


Figure 4.27: **Address Compression Across Different Memory System Levels.**

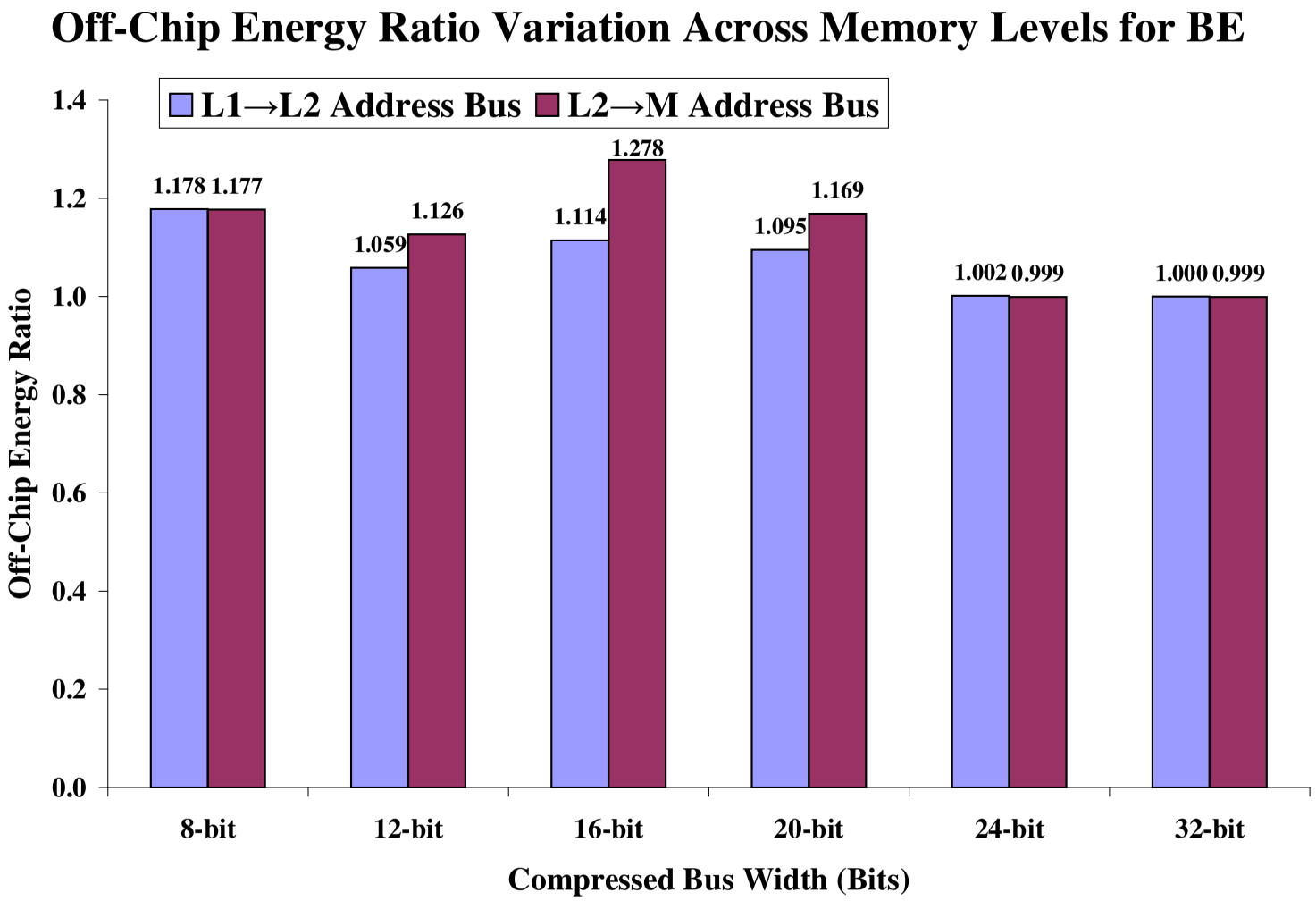


Figure 4.28: **Address Compression Across Different Memory System Levels.**

Chapter 5

Energy-Efficient Compressed Address Transmission and Partial-Match Address Compression

5.1 Introduction

Nanometer design, which will soon make billion transistor chips a reality, has been plagued with many problems that are related to the interconnect system [46]. Some of these problems are increasing delays in interconnects routed in the global layers, where most signal lines like address, instruction, and data buses are routed and increasing power consumption and signal integrity/reliability problems in these buses due to coupled inductance and capacitance effects. In deep sub-micron (DSM) design, circuit techniques like wire and driver sizing, use of

repeaters, etc. and/or physical design techniques like power- and delay-aware routing were adopted to keep such problems of interconnect scaling in check. But, due to rising number of metal layers, smaller spacings and hence the explosion in the amount of inter-wire coupling (both inductive and capacitive), such schemes are no longer complexity-effective for current nanometer designs. In contrast, an effective and scalable solution to alleviate the problems due to the interconnect system in nanometer design is to consider architectural-level techniques that can reduce the pressure on the interconnect system and remove or reduce the reliance on interconnect-aware circuit and physical design.

Since bus lines constitute a bulk of the interconnect system in the upper metal layers, many schemes involving encoding of data transmission for energy, delay, and cross talk have been proposed. Some older schemes use the fact that switching activities and hence energy dissipation in bus lines can be reduced by exploiting the spatial locality of information [6]. Others like the bus-invert scheme require no prior knowledge of data statistics [60]. Recently, due to the dominance of inter-wire capacitive coupling, encoding schemes that minimize inter-wire transitions have been proposed [77, 34, 58]. Another way to possibly reduce bus energy dissipation is compression—compression followed by encoding has been found to yield highest energy reductions for buses [50]. In a compression scheme for buses, data to be sent is compressed and transmitted on a narrow-width bus if the compression is successful or transmitted in multiple cycles if not. Compression can also result in overall cost benefits since savings obtained by reducing the number of bus lines, associated drivers, and repeaters can outweigh the area/cost incurred by the compression and decompression

hardware. However, when transmitting compressed information over narrow-width buses, energy dissipation may actually worsen if: (i) the compression hardware is unsuccessful most of the time, or (ii) bits of the compressed addresses are misaligned so as to cause an increase in the number of self and inter-wire transitions.

5.1.1 Scope and contributions of this work

Some previous work has proposed compression and decompression schemes for address, instructions, and data primarily to improve bandwidth and latency for buses [52, 11, 10]. The effectiveness of a compression scheme in reducing the switching activity in off-chip data buses was studied in [3] and these results are not necessarily relevant to on-chip buses in current nanometer-scale technologies where coupling capacitances dominate. Recently, energy efficiency of various data-value prediction schemes—which are often viewed as providing performance enhancement—were studied in [68] for on-chip data buses and in [63] for off-chip data buses. Thus, no previous work has considered energy-efficiency as a goal when designing compression schemes for buses.

In this work, we propose various techniques that can be used with existing compression schemes for buses to ensure the best energy-efficiency for compressed information transmission. Note that these techniques are complementary to others like bus encoding which can be applied after compression or techniques like low-swing signaling [75], charge recycling [33], or wire optimizations like spacing and shielding [13]. In fact, the area/cost of circuitry that is saved by adopting compression can be used to increase spacing or insert shield wires to obtain further benefits. Although the ideas behind many of our techniques are broadly appli-

cable to all buses (address, instruction, and data), the techniques presented in this chapter are somewhat specialized for the purpose of address compression. In addition, we propose new and optimized designs of compression-caches that are different and perform substantially better compared to those proposed earlier.

In Chapter 4, our simulations have shown that BE works better for on-chip address buses than another scheme, DBRC. Hence, we use BE as the default address compression scheme in this work and report energy reductions using our techniques for this scheme. However, many of our techniques will provide similar reductions when applied to the DBRC scheme also. The simulation methodology used in this study has been mentioned earlier in Sec. 4.3.

We will first present our proposed transmission techniques and results. The transmission techniques are based on the techniques proposed earlier for HOC in Sec. 3.7. Similarly, each successive technique we present is an improvement over the previous one and results in progressively better energy reductions. Then, in Sec. 5.8, we will present a highly energy- and performance-efficient dynamic address compression methodology for nanometer-scale address buses designed to improve the hit-rate and reduce miss penalty of dynamic compression caches and hence improve performance, energy, and cost.

5.2 Technique 1: Bus arrangement

It can be observed easily that, with the default transmission format described in Sec. 4.2, when misses in the sender cache occur and addresses have to be transmitted in full in multiple cycles, bus energy dissipation will be higher because of misaligned bits, i.e., bits that have

no correlation with bits in the same position transmitted in the previous cycle—causes a self-transition—and bits that are uncorrelated with neighboring bits in the same cycle which causes coupling transitions. Hence, in Chapter 4, we have modified the original transmission format used for BE to improve its energy efficiency by rearranging some of the fields to reduce unwanted self and coupling transitions, which is shown on the left in Fig. 5.1. In this section, we propose a new transmission format for compressed addresses—separately for hit and miss cases—based on the following principles of arranging the different fields to minimize self and coupling transitions.

Due to the highly sequential nature of addresses, the least significant bits (LSB) of the address will be the most active. For this reason, in address compression schemes, the lower order portion of the address is not compressed. To reduce the coupling energies of the lower order portion of the address (U-field), we place U_0, the LSB of the U-field in the MSB line of the compressed bus during the hit as well as during the first cycle of a miss as shown in the figure on the right in Fig. 5.1. Thus, the bit U_1 now occupies the LSB line of the bus and its coupling energy is reduced because it can no longer cause a toggle transition with the U_0 bit. Further, the U_0 bit which has been placed next to the C-bit also results in lesser coupling energies since its neighbor is expected to change state less frequently because of high hit rates in the sender cache. Also, the edge lines have less coupling capacitance since they have only one neighboring line and this will lead to lesser coupling energies. Note that we rearrange bits as described above only during a hit and the first cycle of a miss because, for subsequent cycles of a miss when the H-field is transmitted the two least significant bit

line may not necessarily be the most active lines to warrant being decoupled from each other.

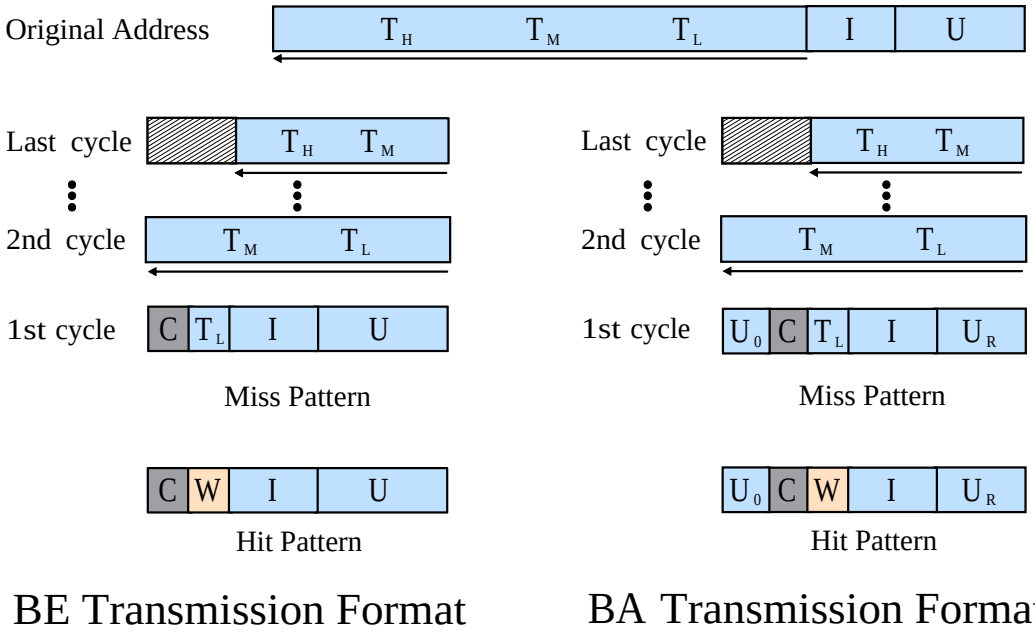


Figure 5.1: **Proposed Bus Arrangement Techniques.** The figure on the left shows the new basic transmission format that we propose for the BE address compression scheme. The figure on the right further reduces energy by rearranging some bits to reduce unwanted coupling transitions.

5.3 Technique 2: Idle-bit insertion for coupling energy reduction

When a missed address is sent in multiple cycles over the narrow-width bus, the last cycle of transmission of the address is likely to be poorly utilized. This effect is more pronounced when the compressed bus width is a non-integral fraction of the uncompressed address width. In such cases, the *idle* bits can be used to reduce coupling energies by placing them between *active* bits in different cycles of the miss. Note that if a bit is designated as idle in the current

cycle, then it means that it holds the value from its previous cycle. Thus, an idle bit can never have a toggle transition with either of its neighboring bits for the current cycle.

Given a fixed number of idle bits that we can insert, we first place idle bits at the W-field in the first cycle of the miss transmission which will potentially help reduce the coupling energy for that cycle. This is because the W-field is un-correlated with its neighboring fields and inserting idle bits between them reduces the chance of unwanted coupling transitions. Next, we assign the maximum possible number of idle bits to the second cycle of miss transmission. For a *w*-bit compressed bus the maximum number of idle bits that can be assigned to each cycle is $\lfloor w/2 \rfloor$. Now, suppose *k* bits were assigned to the second cycle, then the idle bits are interspersed alternately with active bits in the cycle starting from an active bit at the LSB to achieve maximum benefits for coupling energy reduction. After assigning to the first and second cycles as above, if idle bits remain, then they are assigned to the third cycle and so on till all the idle bits are exhausted.

5.4 Results for Address Arrangement and Idle-bit Insertion

Results for address arrangement combined with idle-bit insertion technique compared to the default transmission format for BE are shown in Fig. 5.2. The results show that, in all cases, net energy reductions are obtained over the default transmission format used for BE. Average energy reductions are about 5.5% with our proposed address arrangement and idle-bit insertion techniques. Further, most of the energy reductions are obtained as a result

of reduction in the number of toggle transitions. In most cases, where the bus widths are reduced less than 50%, the default BE scheme results in an increase in energy over an uncompressed bus, but with our techniques energy reductions are obtained in almost all cases.

5.5 Technique 3: LRU-encoded way-bits

We call this scheme BAL (Bus arrangement + LRU-encoding) and it applies only in the case of compressed addresses transmitted during a hit. We replace the W-field, which normally points to the way number of the tag that hit in a particular line of the sender cache, with the least-recently-used (LRU) number that the sender cache maintains for each entry. This encoding of the way-bits using the LRU-value is motivated by the fact that, in the sender cache, the most recently accessed entry is likely to be accessed again and in this case an LRU value of zero will be used to encode the way bits. If the most recently accessed is not accessed again, then it is highly likely then the one with the next higher LRU value (equal to one) will be accessed. Thus, compared to the previous transmission the LRU-encoded way bits will cause only transition in only one bit. If the way bits were not LRU-encoded, then they can take one of a different values in $0,\ldots,a-1$, a being the set-associativity of the sender cache. Our argument that LRU-encoded way-bits will take the zero value most of the time is supported by simulation results shown in Fig. 5.3. This figure shows that, for most bus-widths and set-associativities, way-bits encoded with LRU values are zero-valued and remain unchanged more than 50% of the time and this may lead to substantial self- and coupling-energy savings.

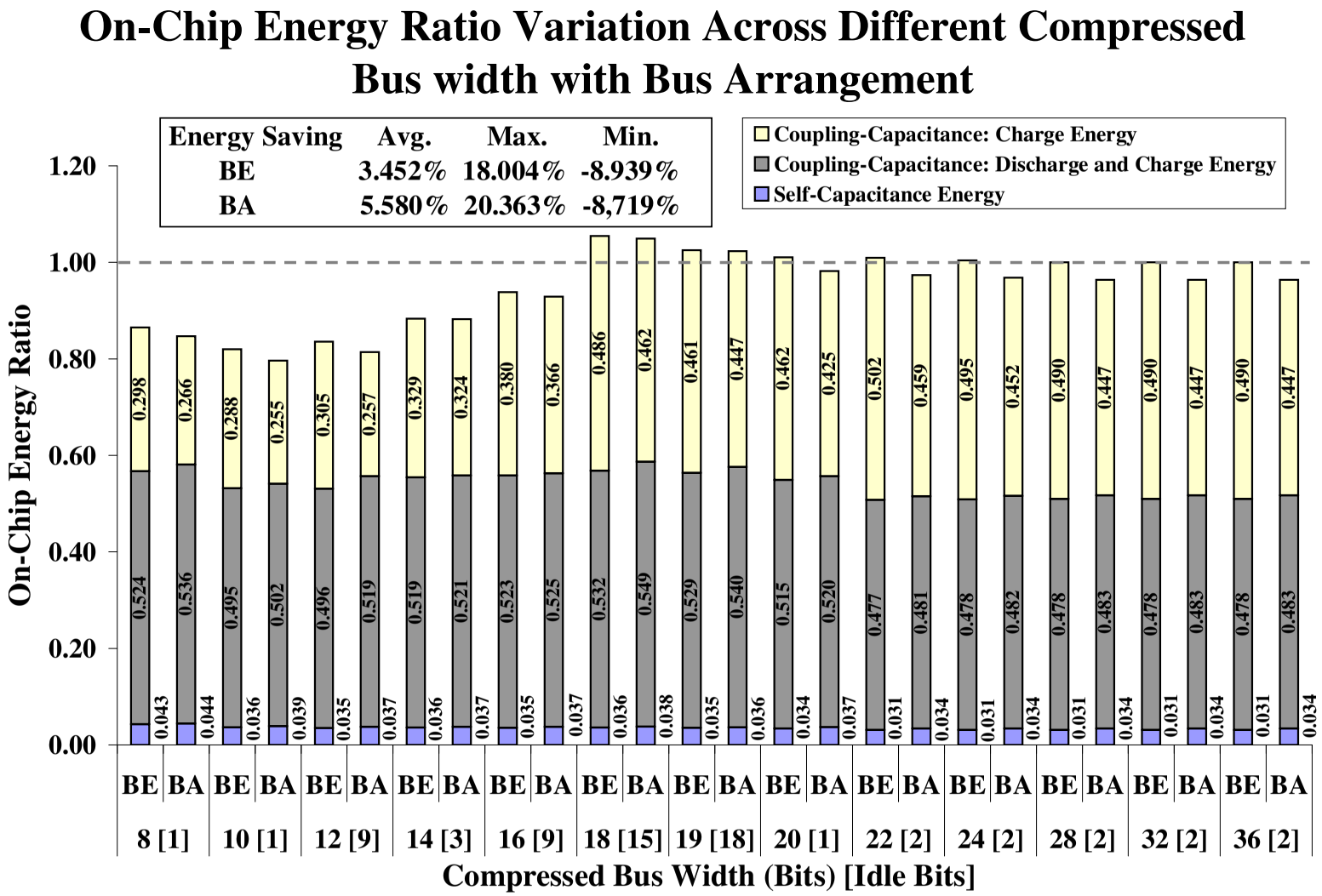


Figure 5.2: **Energy Reduction Using the Proposed Address-arrangement Technique.**

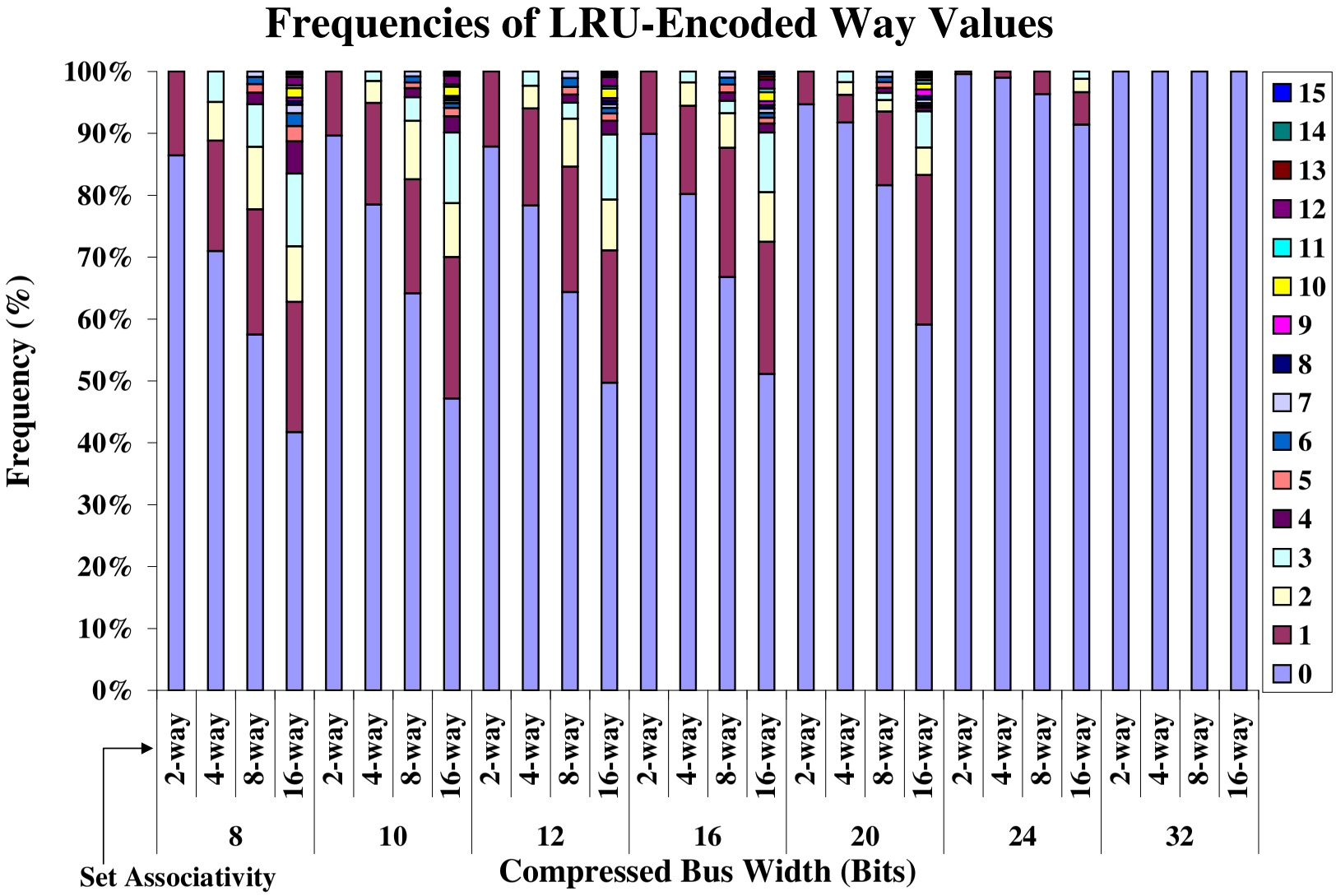


Figure 5.3: **Frequency of Values Taken by LRU-encoded Way Bits.**

Energy results for compressed address transmissions with and without using LRU-encoded way bits are shown in Fig. 5.4 and Fig. 5.5. In this study, we varied the set associativity of sender cache from 2-way to 16-way. Note that using direct-mapped caches does not make sense for this study since there are no way bits to encode in that case. Also, we considered the maximum size of the E-field to be 4 bits and hence 16-way set associativity represents a fully-associative sender cache. Average energy reductions of about 12% were obtained for on-chip buses due to reductions in both self and coupling energies with LRU-encoded way-bits. With off-chip buses energy reductions ranging from 0.8–4.5% were observed due to reductions in self-energies. It can also be observed that, in both cases, higher set-associativities yield better energy savings when LRU-encoded way bits are used. This is because higher set-associative caches provide better hit rates.

5.6 Technique 4: Encoding higher order part of the address

In this scheme, we encode the higher order part of the address (also called the tag-field since it is stored as tag in the sender cache) using a two step XOR process as shown in Fig. 5.6. Note that computing bitwise XOR of two n-bit addresses requires constant time and little hardware and hence this will not add much extra latency to the bus interface. We call this BALT (Bus arrangement + LRU-encoding + Tag-encoding) and it is applied only in the case of a missed address.

First, the H-field of the current address is XOR-ed with the H-field of the previous address.

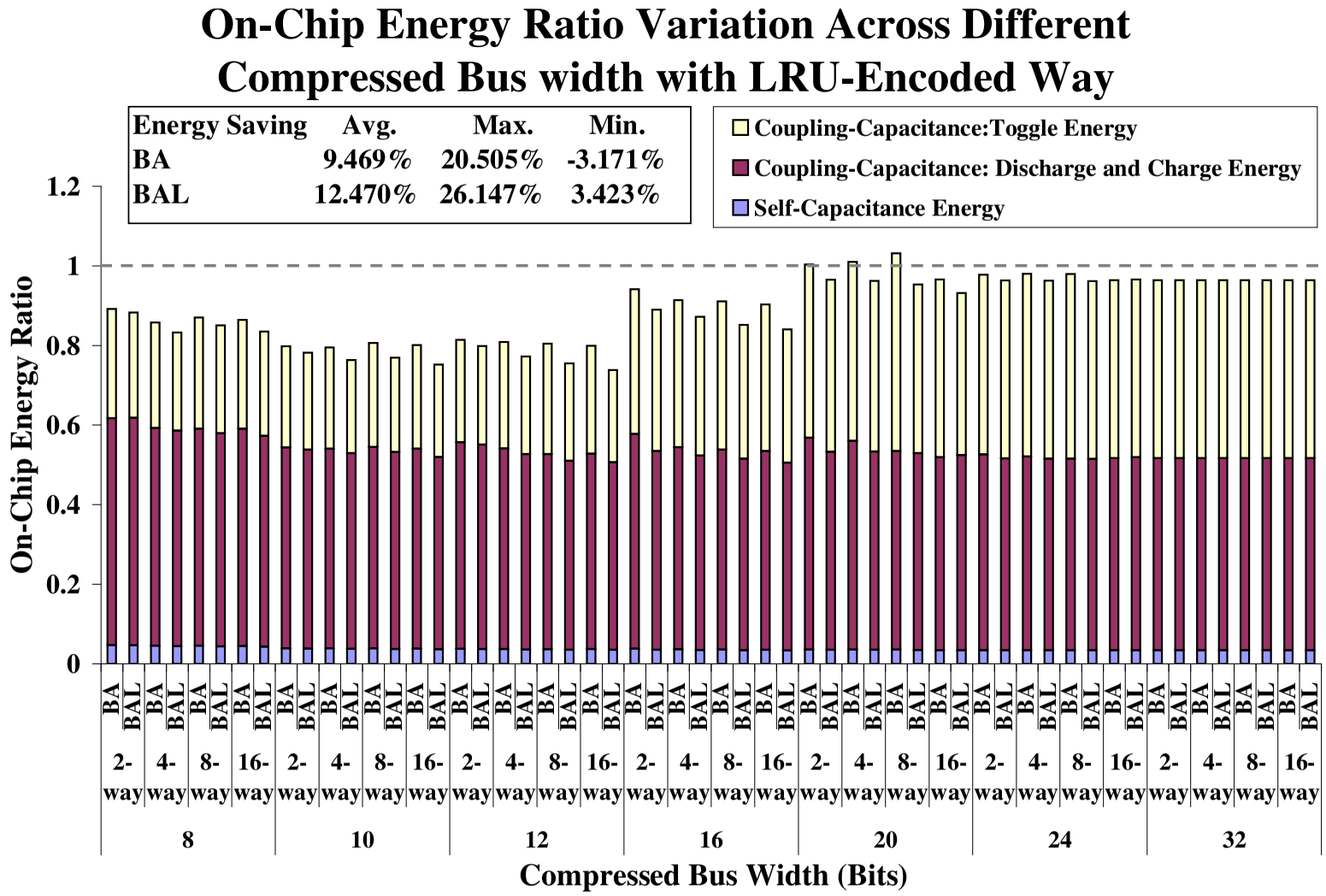


Figure 5.4: **Energy Reduction Using the LRU-encoded Way-bit Technique:** On-chip bus energy dissipation ratio for different compression cache set associativities.

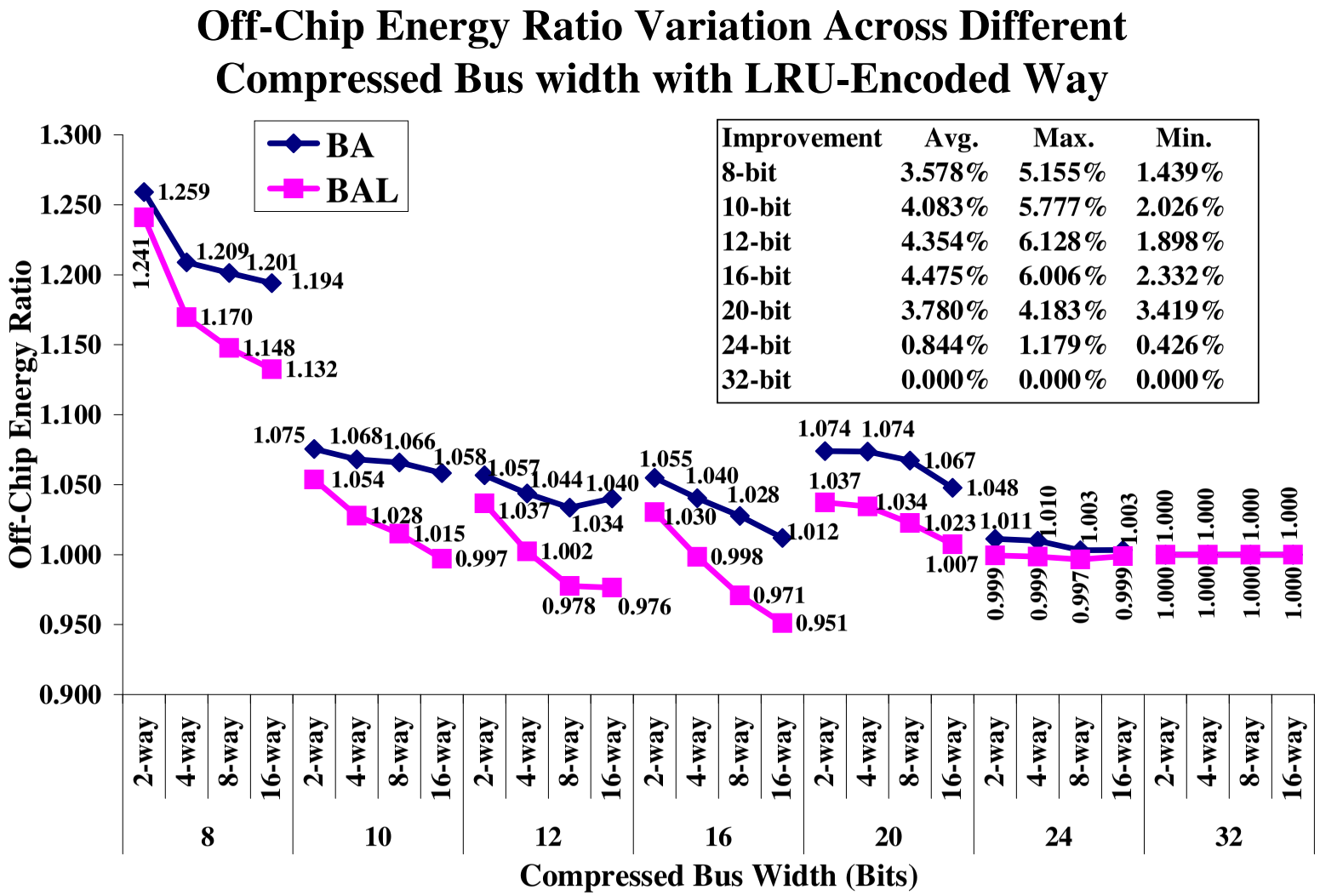


Figure 5.5: **Energy Reduction Using the LRU-encoded Way-bit Technique:** Off-chip bus energy dissipation ratio for different compression cache set associativities.

Due to temporal and spatial redundancy of the bits, the result will have more zero-valued bits than the H-field of the current address. However, this XOR-ed form of the H-filed may have a power disadvantage. To rectify this, in the next step, we again XOR each bit of the new H-field with the bit transmitted at the corresponding bit position on the bus in the previous cycle. Since the first step yielded more zero-valued bits, the next step will make the H-field pattern similar to the one transmitted on the bus in the previous cycle thus reducing both self and coupling energies. It can be observed from results shown in Fig. 5.6 that compared to the previous scheme (BAL), H-field encoding alone can result in extra average energy reductions of nearly 3.3% for on-chip buses and about 13.4% for off-chip buses. It can also be observed that this scheme provides the best energy reductions for off-chip buses because, by doing two-step XOR operation, the self energy between the the tag bits transmitted at the same position in two consecutive bus cycles reduces since the bits are likely to have the same value.

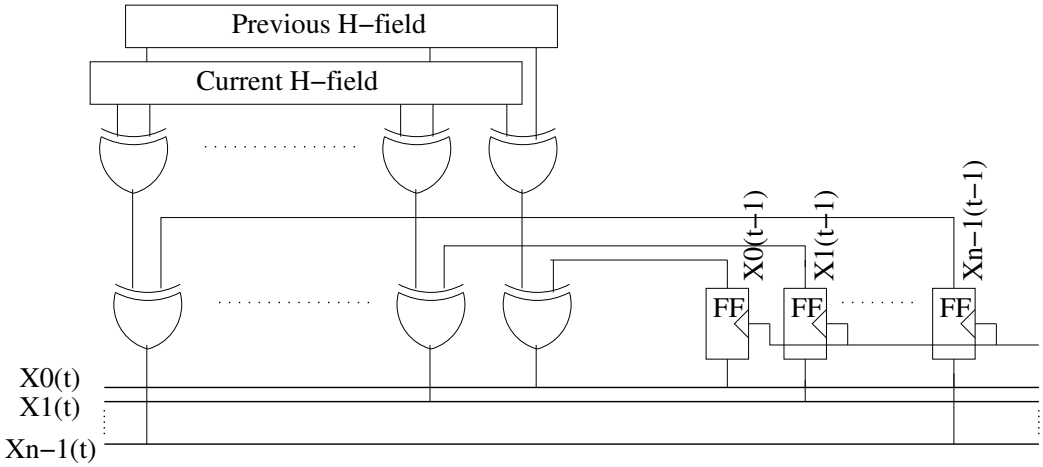


Figure 5.6: **Structure for Encoding the Higher Order Part of the Address.**

5.7 Technique 5: XOR encoding for the compressed address

It was observed earlier that using an XOR form of the H-field for transmitting missed addresses yield good reductions in self-energy and some reductions in coupling energy also. However, by using an XOR form of the entire address before address arrangement, LRU-encoding, or tag-encoding are applied, further coupling energy reductions can be obtained. Hence this scheme is called XOR-BALT.

In this technique, the incoming (uncompressed) address is first XOR-ed with the previous address and this XOR version of the address is used to form the compressed address depending on whether a hit or miss occurs in the sender cache. Note that we do not use the XOR version of the address to look up the sender cache because we found that doing so leads to no substantial performance or energy benefit. Thus, in the case of a hit, the U-field and the I-field which are obtained from the original address are in XOR form but the control bit and the W-field are not. The W-field is encoded with the LRU bits as discussed earlier. In the case of a miss, all fields except the control bit are in XOR form and the H-field encoding is done as described earlier; note that the first step of the two-step XOR process for tag encoding is already done in this case. Results in Figs. 5.7 and 5.8 show that this scheme results in about 0.7% better energy reduction than the previous scheme on the average for on-chip buses but also results in a 0.3% degradation for off-chip buses. The reason for this degradation is that, every bit transition in the original trace will cause two bit transitions in

the XOR trace, except when consecutive transitions occur in the original trace (not likely), in which case there will not be any transition in the XOR trace.

In addition, we apply the technique 5 proposed in Sec. 3.7.6, which use the idle bits as active shields, on top of the XOR-BALT. We call this scheme XOR-BALTI (XOR-BALT + Idle-bit encoding). Results for the XOR-BALTI scheme in Fig. 5.7 show that this scheme performs the best for on-chip buses yielding about 14.7% energy reductions on the average for compressed address transmission.

5.8 Partial-Match Compression Cache

To improve the hit-rate and reduce miss penalty of the compression cache used in the previous schemes, we propose *partial-matching* of the tag portion stored in the compression cache with the higher order portion of the address. In partial-match (PM) compression-cache, we check for the longest match between the tag portion stored in the cache and the higher order portion of the incoming address. We consider k possible groups of bits ending at the most significant bit (MSB) of the incoming address as shown in Fig. 5.9. For the hardware schematic shown in Fig. 5.10, the value of k is four. If a partial match in any of these k groups occurs, the control number for each group is transmitted along with the index. The remaining portion of the higher order part of the address (that did not match the tag) are sent in uncompressed form, as is the lower order portion of the address. In case of a miss, where none of the partial matches succeeded, the entire address is sent.

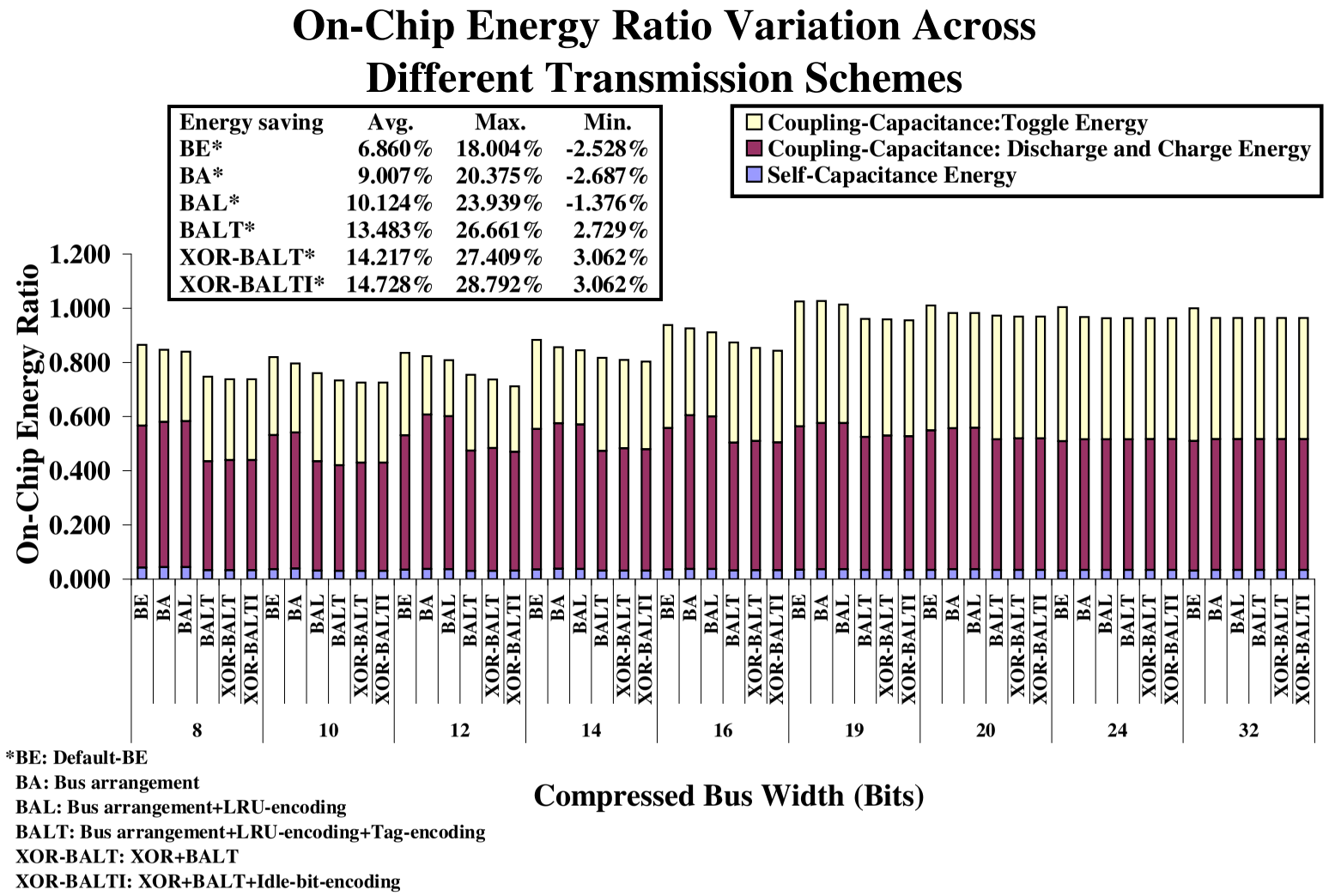


Figure 5.7: **On-Chip Energy Reduction Using All the Proposed Techniques.**

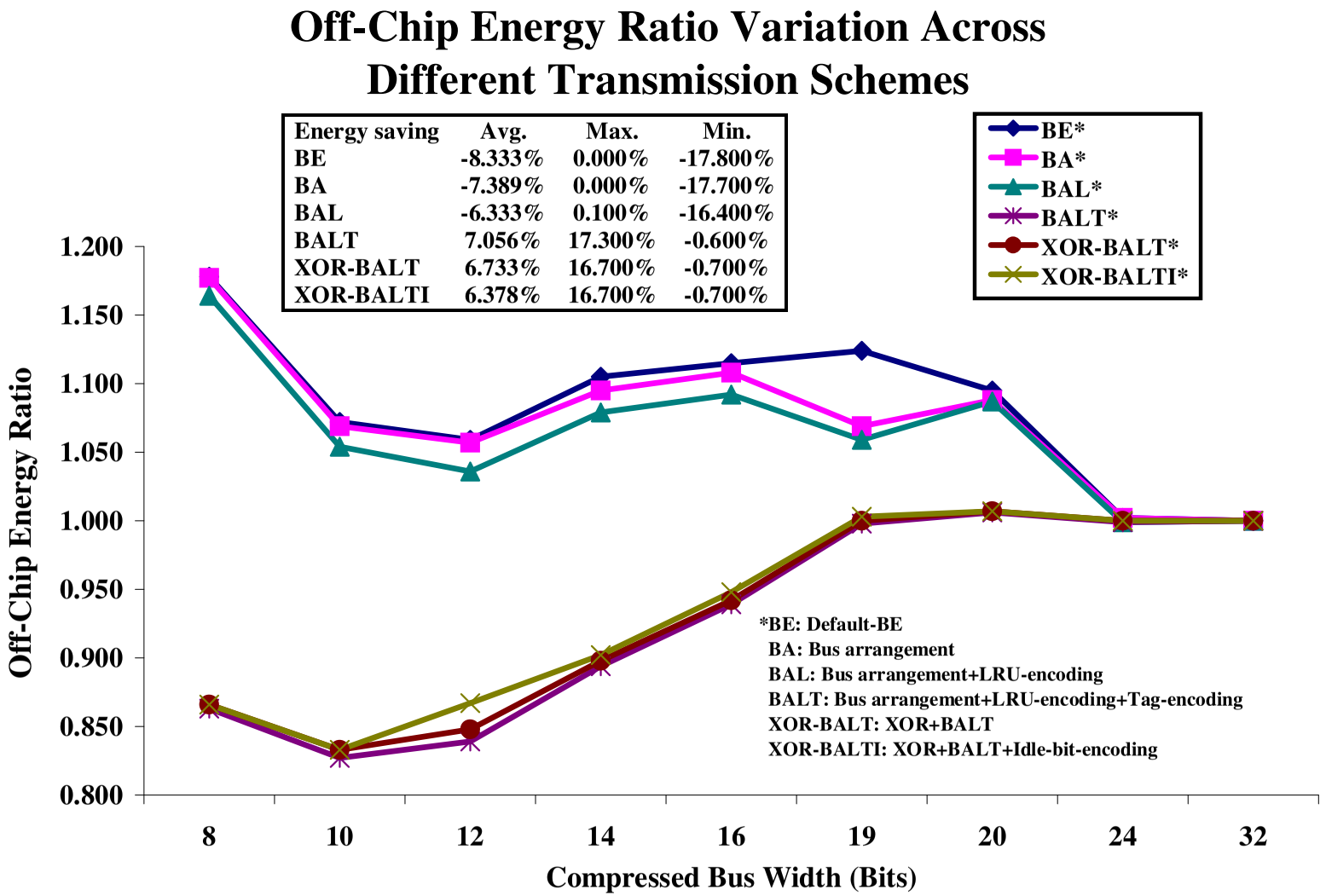


Figure 5.8: **Off-Chip Energy Reduction Using All the Proposed Techniques.**

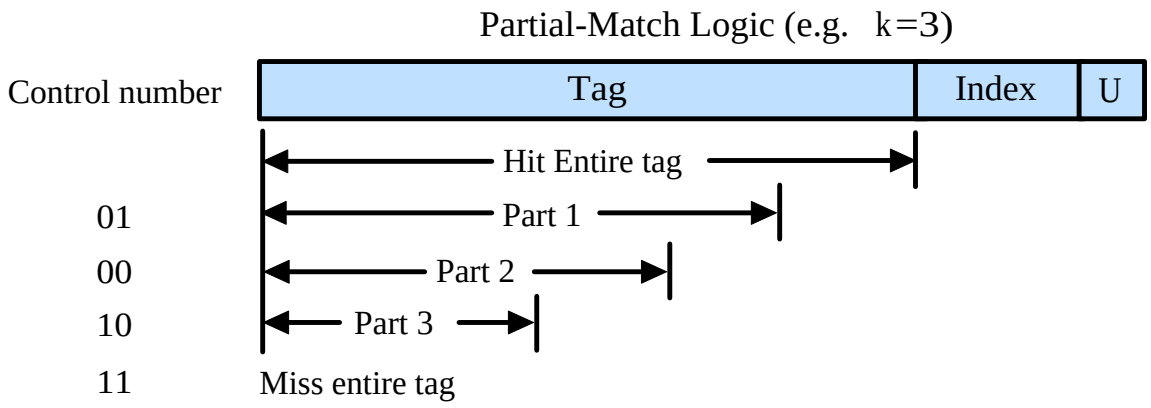


Figure 5.9: **Partial-Match Logic.**

5.8.1 Partial-match encoding and transmission format

The transmission techniques we proposed for BE can also be applied in PM transmission. However, for tag encoding, the tag bits are XOR-ed with the tag in the PM compression cache entry instead of the tag of the previous address when partial hit happens. Since the tag of the current address partially matches the tag of the entry in the compression cache, the result will have more zero-valued bits than XOR-ing the tag of the current address with the tag of the previous address. As shown in Figs. 5.11 and 5.12, this technique provides the same on-chip and off-chip energy reduction as the one we proposed in Sec. 5.6. The benefit of applying this technique is that the performance of the compression can be improved since the XOR tag can be taken from the comparison result obtained when checking compression cache hit/miss instead of calculated separately. In addition, since the LRU-encoded way-bits are zero-valued more than 50% of the time, shown in Fig. 5.3 and the percentage will increase with PM, the LRU-encoded way-bits can be XOR-ed with the bits transmitted at the

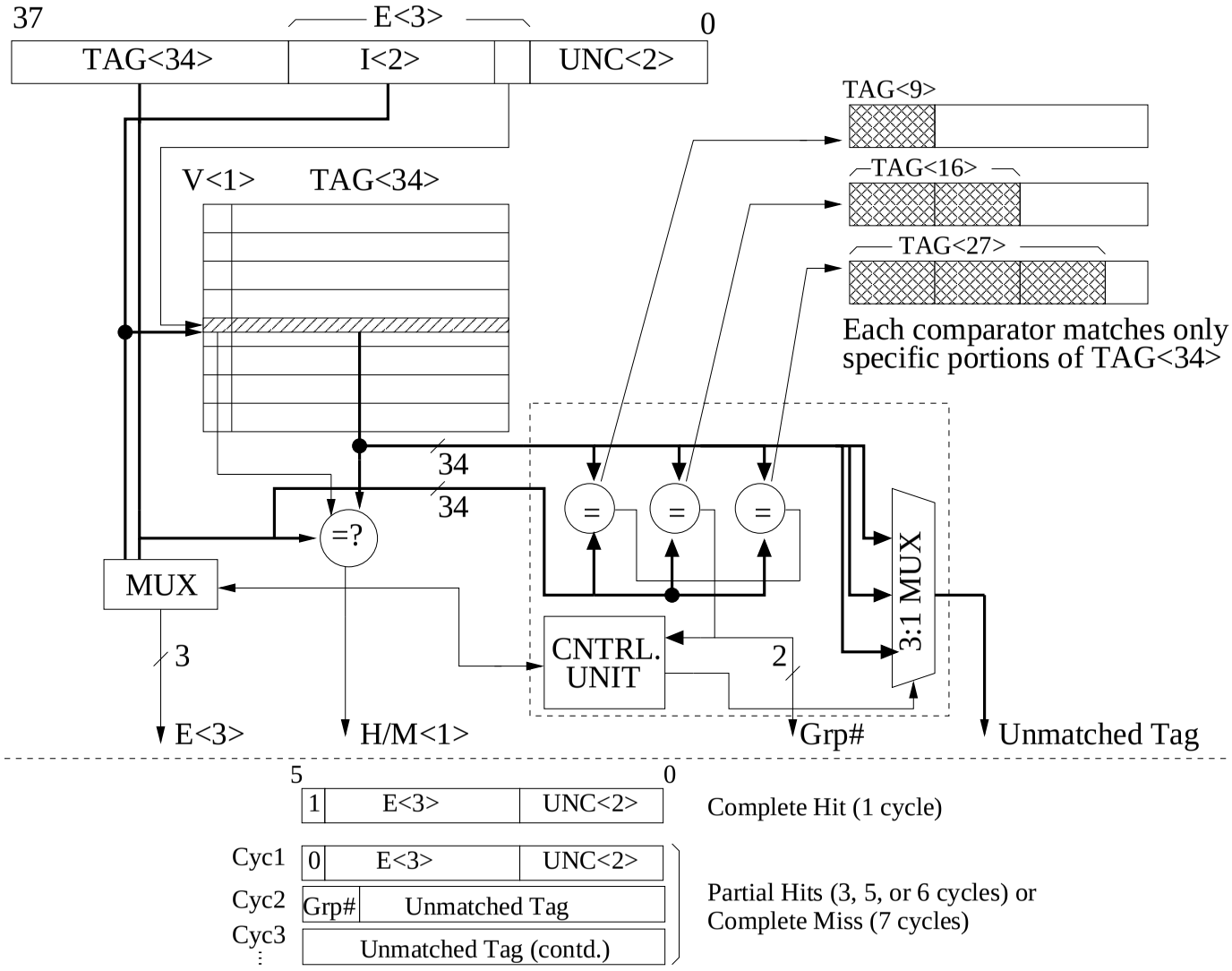


Figure 5.10: **Partial-Match Compression Cache:** Hardware organization for proposed partial-match address compression scheme.

corresponding bit position on the bus in the previous cycle before placed on the bus to reduce self- and coupling-energy. The off-chip and on-chip energy ratios can be slightly improved by 0.011 and 0.009 with way bit encoding, respectively.

In a partial match scheme with *k* partitions in the compressed portion of the address, the starting bit position of each partition starting from the least significant bit (LSB) is $[c \times B - (1+C) - \log_2(E) - U]$-1, where *c* is the number of cycles required to transmit the codeword, *B* is the narrow bus-width, *C* is the number of bits used for partial-match control number to indicate which part has hit, *E* is the number of entries in the compression cache, and *U* is the width of the uncompressed portion.

For k partitions, we assign the k partial-match control numbers in two energy-efficient ways based on the total frequencies of the partitions and the number of 1s in the binary formats of the control numbers. For k partition, our first approach, control A (CA), is to use $\log_2(k+1)$ bits for the control numbers if $k+1$ is power of 2. Otherwise, the first approach is the same as the second approach, control B (CB). In the second approach, we use different number of bits to differentiate the control numbers. If there is only one bit for the control number, the control number is 0. Otherwise, all bits in the binary format of the control numbers are 1s except the LSB bit. As far as the assignment of the control numbers, the main idea is to assign the control numbers which have more 1s to the partitions, which have lower frequency to minimize self and coupling energy. Fig. 5.13 shows the control numbers used for k partitions in binary format. The transmission format for PM is shown in Fig. 5.14. Since the control numbers change less frequent than the tag, the control numbers for the

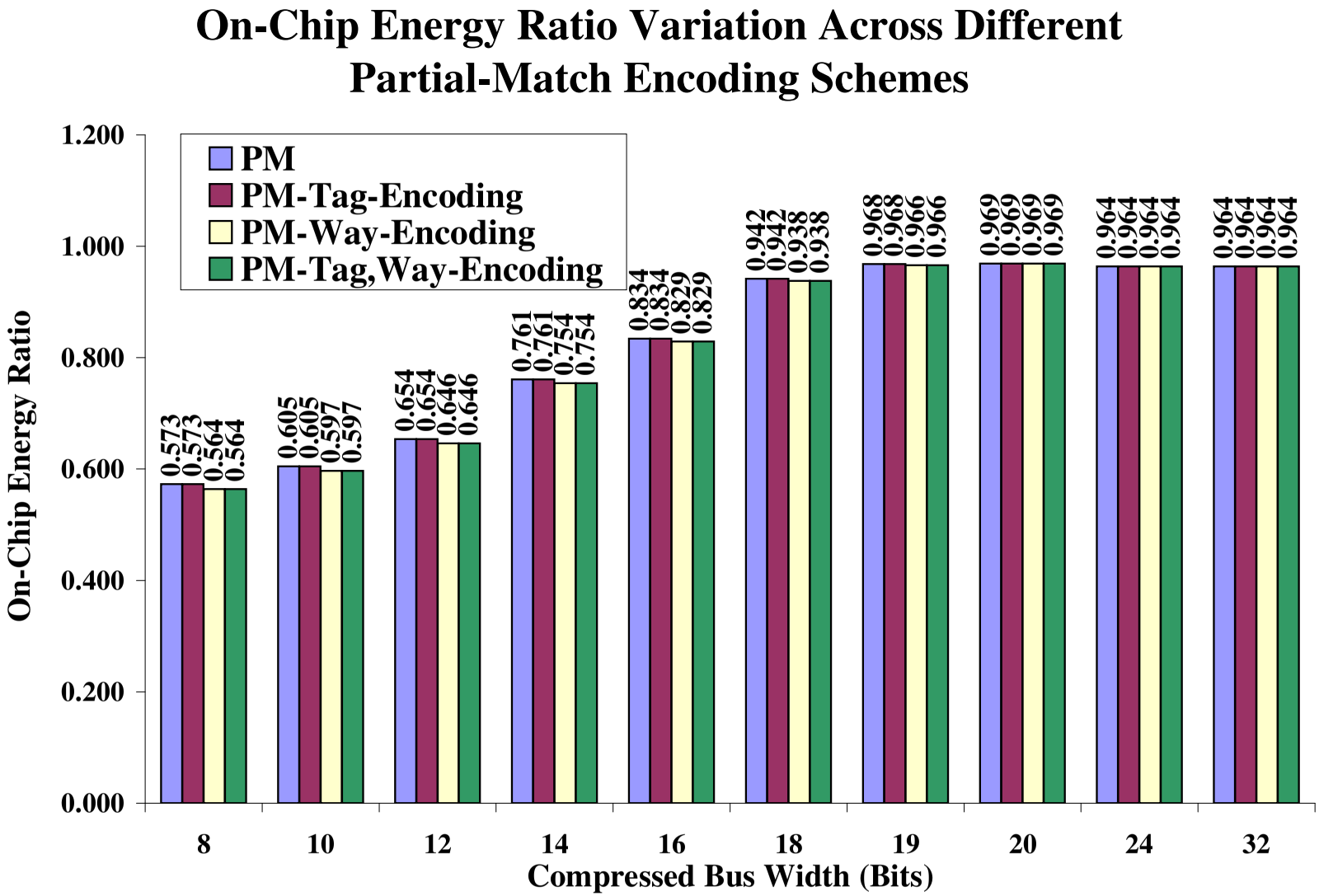


Figure 5.11: **On-Chip Energy Ratio for PM for Different Encoding Schemes.**

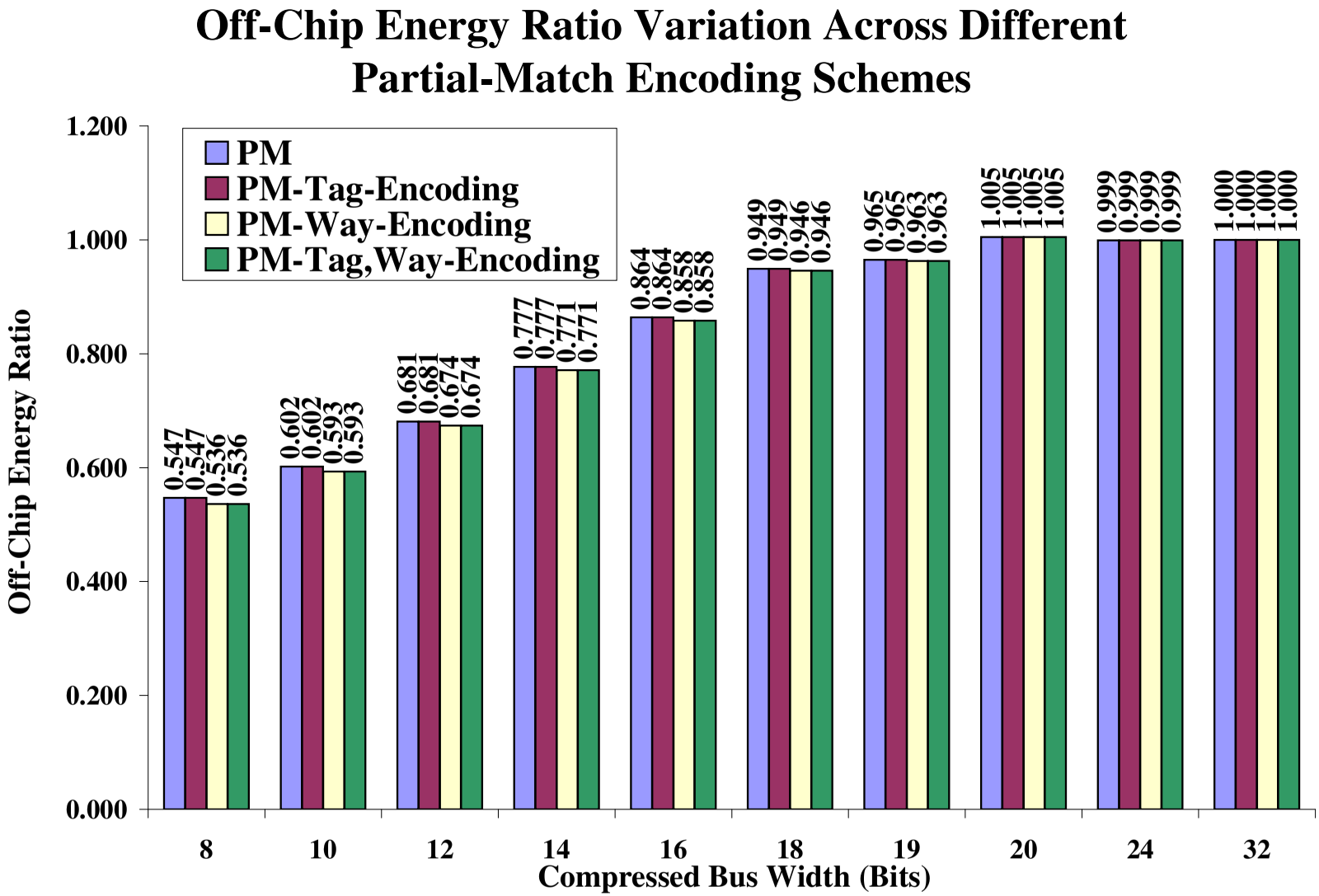


Figure 5.12: **Off-Chip Energy Ratio for PM for Different Encoding Schemes.**

partitions are interspersed alternately with the tag bits in the second cycle starting from MSB to minimize the coupling energy. In addition, before placing the control number on the bus, we XOR each bit of the control number with the bit transmitted at the corresponding bit position on the bus in the previous cycle. Since the higher the frequency of the partition, the more zero-valued bits in the control number, the XOR operation will make the bit pattern of the new control number similar to the one transmitted on the bus in the previous cycle thus reducing both self and coupling energies.

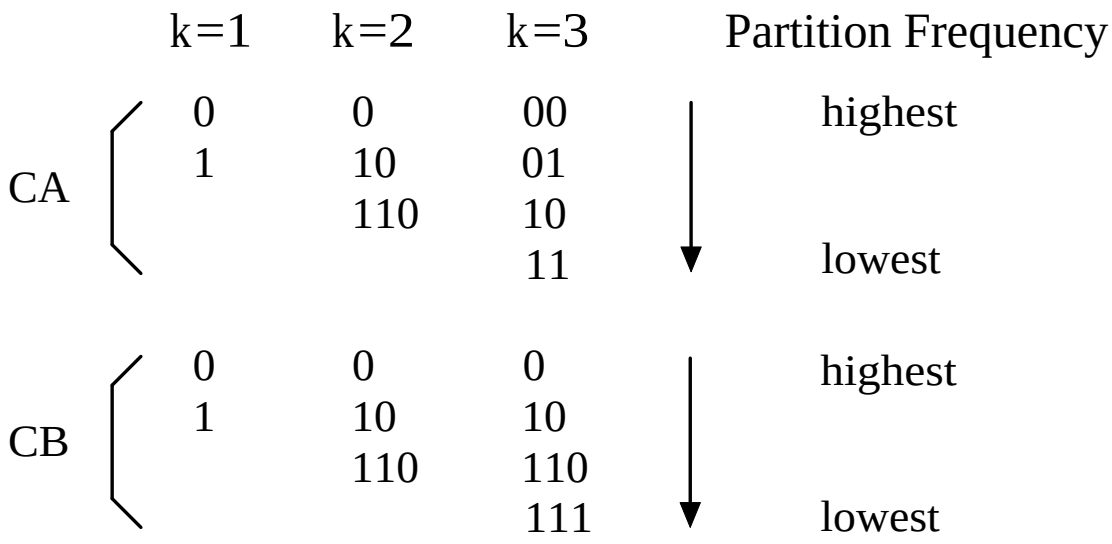


Figure 5.13: **Control Number Format for PM.**

5.8.2 Average miss penalty and average bit penalty

We first collect the individual frequency (IF) for each possible partition ending at the MSB, shown in Fig. 5.15 for n-bit tag field. Fig. 5.16 shows the individual frequency of different partitions for different buses. The complete hit case is where the partition point equals to the

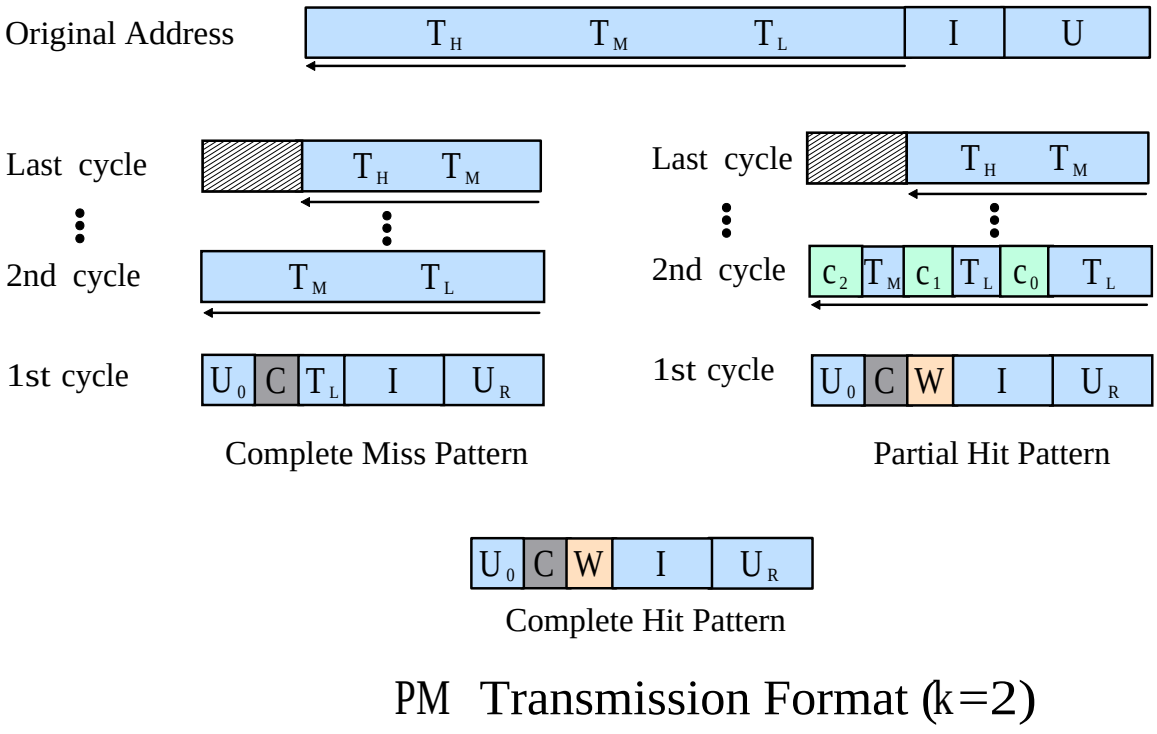


Figure 5.14: **Transmission format for PM.**

number of bits in the tag field. When the bus width increases, the frequency of complete hit increases, which means partial-match compression will be more effective for narrow buses.

Based on the individual frequencies, we use the procedure shown in Fig. 5.17 to generate the total frequency (TF) for the different partitions, which will be used to choose the best partitions for performance optimization design and energy optimization design. Fig. 5.15 shows the total frequency for the j-bit partition starting from i bit, $TF_{i,j}$.

There are two parameters we consider, average miss penalty and average bit penalty, when we choose the best combination of different partition points. In the case of complete tag hit, only one cycle is needed for the transmission, which is the same as BE. Miss penalty (MP) is the extra cycles taken due to partial hit or complete miss. The *average miss penalty* of k

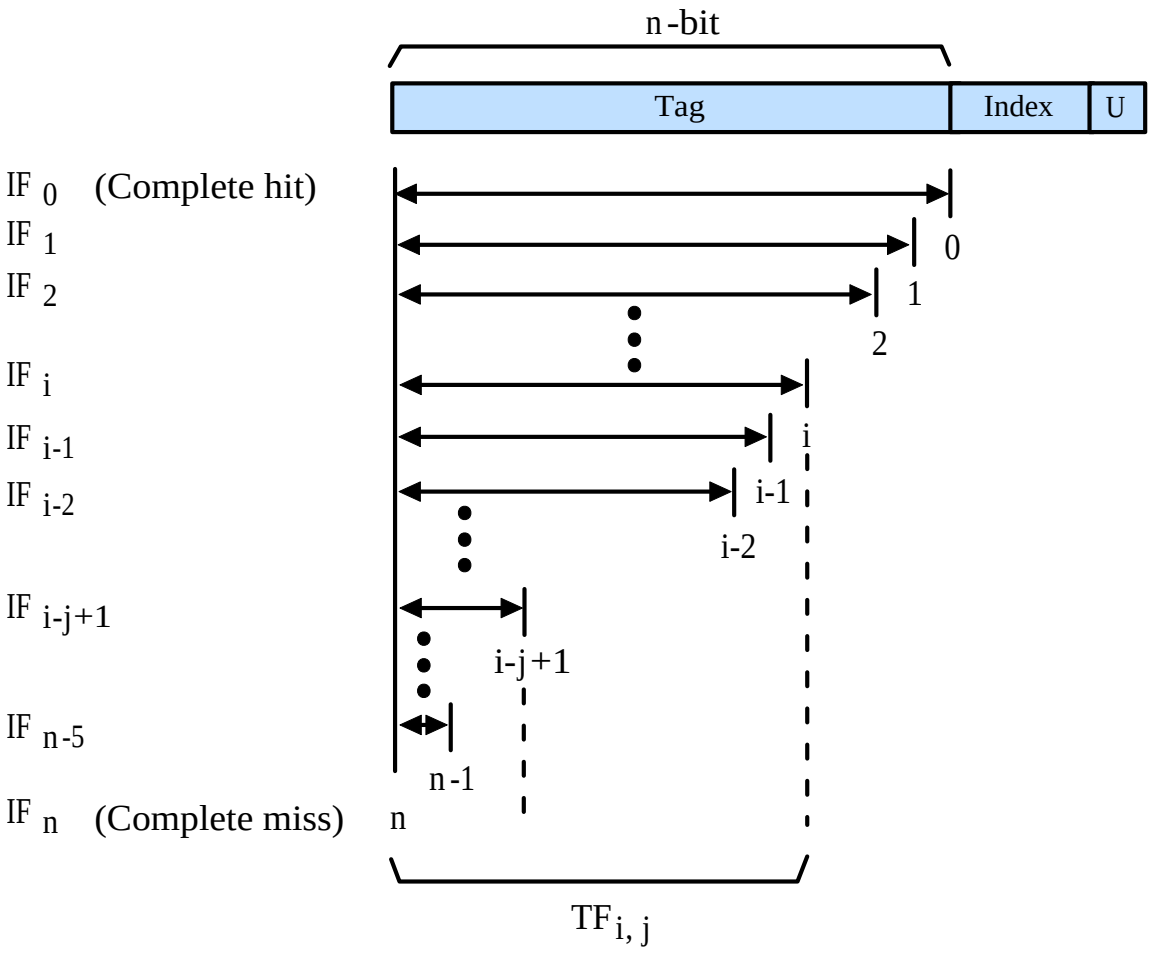


Figure 5.15: **Individual Frequency and Total Frequency.**

partitions for n-bit tag field is as follows:

$$MP = \sum_{i=1}^{k} MP_i \times TF_{i,n-i+1}$$

The bit penalty (BP) is the extra bits transmitted due to partial hit or complete miss including unmatched portion and control bits. Thus, the *average bit penalty* for k partitions is:

$$BP = \sum_{i=1}^{k} BP_i \times TF_{i,n-i+1}$$

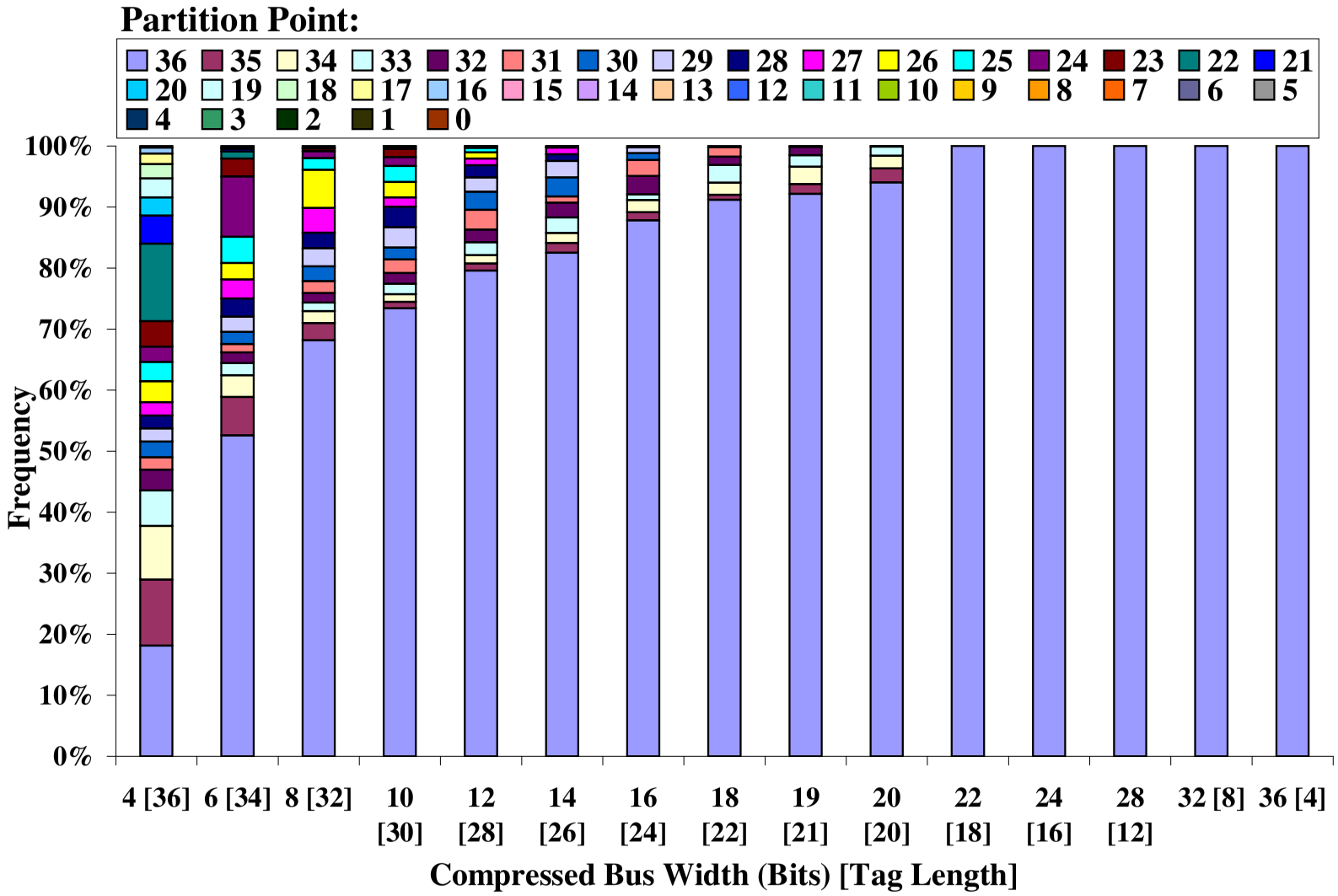


Figure 5.16: **Individual Frequency of Different Partition Points for Different Compressed Buses.**

Procedure TotalFrequency($IndiFrequency[TagLength], TagLength$)

/* To determine all $TotalFrequency[i, j]$ of interest.

$TotalFrequency[i, j] := IndiFrequency[i] + IndiFrequency[i-1] + ... + IndiFrequency[i-j+1]$.

$0 \leq i \leq t, 1 \leq j \leq i$. */

Begin

For $(i = 0, i < TagLength; i++)$ **begin**

1. $TotalFrequency[i, 1] := IndiFrequency[i]$;

2. **For** ($j = 2$; $j < i$; j++) **begin**

   $TotalFrequency[i, j] := IndiFrequency[i] + TotalFrequency[i-1, j-1]$;

   Endfor

Endfor

End /* Procedure TotalFrequency */

Figure 5.17: **Procedure TotalFrequency**

5.8.3 Performance and energy optimized designs

In performance optimized design (PO), the tag field is partitioned to obtain the minimum performance penalty of the address compression. Transmission for any partial match misses always takes full bus cycles to ensure the minimum miss penalty. The miss penalty for complete miss is the same as BE. The best partition points are chosen based on the average miss penalty. We first find out the average miss penalties for all possible combinations of different partitions. Then the combination which gives the minimum average miss penalty will be used for partial-match compression. For the narrowest compressed bus we consider, 8-bit bus, the miss penalty in BE is 4 extra bus cycles, which means the complete miss in partial-match also takes 4 cycles. So the partial match partitions can only take 1, 2, or 3 extra bus cycles for 8-bit compressed bus. For wider buses, the number of extra cycles will be even less. Therefore the number of different combinations for the partitions is limited and the best combination can be easily detected.

Similarly, for energy optimized design (EO), we partition the tag field based on the average bit penalty to obtain the minimum energy consumption. The number of different partitions including the complete miss is equal to the number of bits in the tag field. If we just simply check all possible combinations one by one, the time complexity will be $n!$ for n-bit tag field. We propose the following divide-and-conquer algorithm, minimum average bit penalty algorithm (MABP), for PM energy optimized design. The time complexity for our MABP is $O(n\log(n))$. In MABP, we use recursion to get the minimum average bit penalty and its corresponding combination of the partitions for sub tag fields first and then use procedure

Algorithm MABP(i, j, k, p)

/*MABP(i, j, k, p) perform an optimal partitioning of interval $[i, j]$ k times to minimize bit penalty. Let it return the bit penalty for this optimal k-partition and let the positions of the optimal k partitions be returned in integer array $p[]$. The bit penalty includes the penalties for all k parts in $[i, j]$ plus the partition at $(j+1)$. */

Begin

For $(i = 0; i < TagLength; i++)$ **begin**

1. b := MAXFLOAT;
2. $k1 := floor(k/2)$;
3. $k2 := ceil(k/2) - 1$;
4. $TotalFrequency[i, 1] := IndiFrequency[i]$;

5. **For** $(l = (i+k1); l <= (j-k2); l++)$ **begin**
   If $(k1 = 0)$ **then** $b1 := l * TotalFrequency[l, (l-i+1)]$;
   else $b1$:= MABP($i, (l-1), k1, p1$);

   If $(k2 = 0)$ **then** $b2 : +(j+1) * TotalFrequency[(j+1), (j+1-l)]$;
   else $b2$:= MABP($(l+1), j, k2, p2$);

   If $(k2 = 0)$ **then begin**
   $b = b1 + b2$;
   Concatenate($p, p1, k1, l, p2, k2$);
   EndIf
   Endfor

Endfor

End /* Algorithm MABP */

Figure 5.18: **Algorithm MABP**

Procedure Concatenate($p, p1, k1, l, p2, k2$)

/* .....*/

Begin

1. **For** $(i = 0, i < k1; i++)$
   $p[i] = p1[i];$

2. $p[k1] = l;$

3. **For** $(i = 0, i < k2; i++)$
   $p[k1 + i + 1] = p2[i];$

End /* Procedure Concatenate */

Figure 5.19: **Procedure Concatenate**

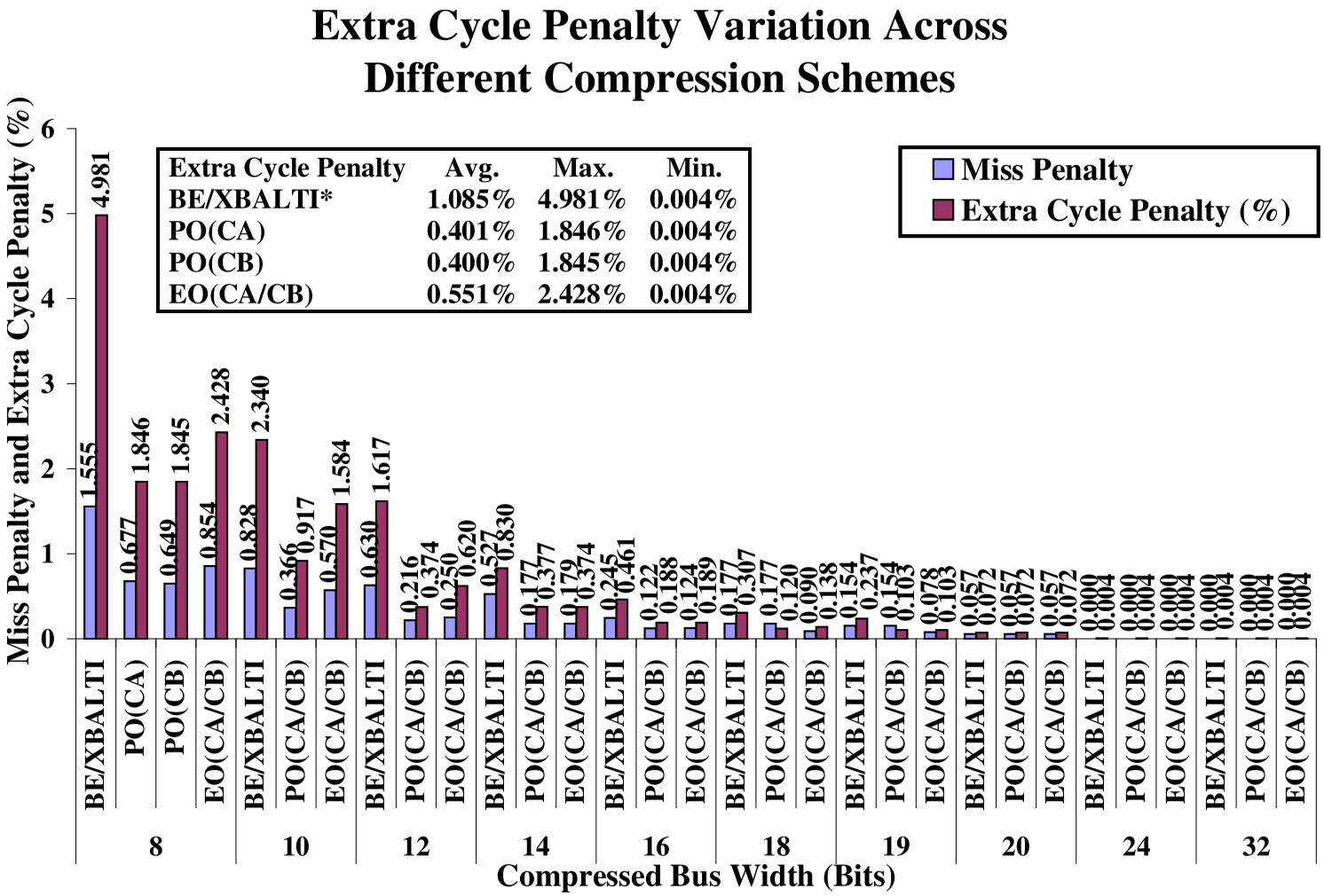


Figure 5.20: **Extra Cycle Penalty Variation Across Different Compression Schemes.**

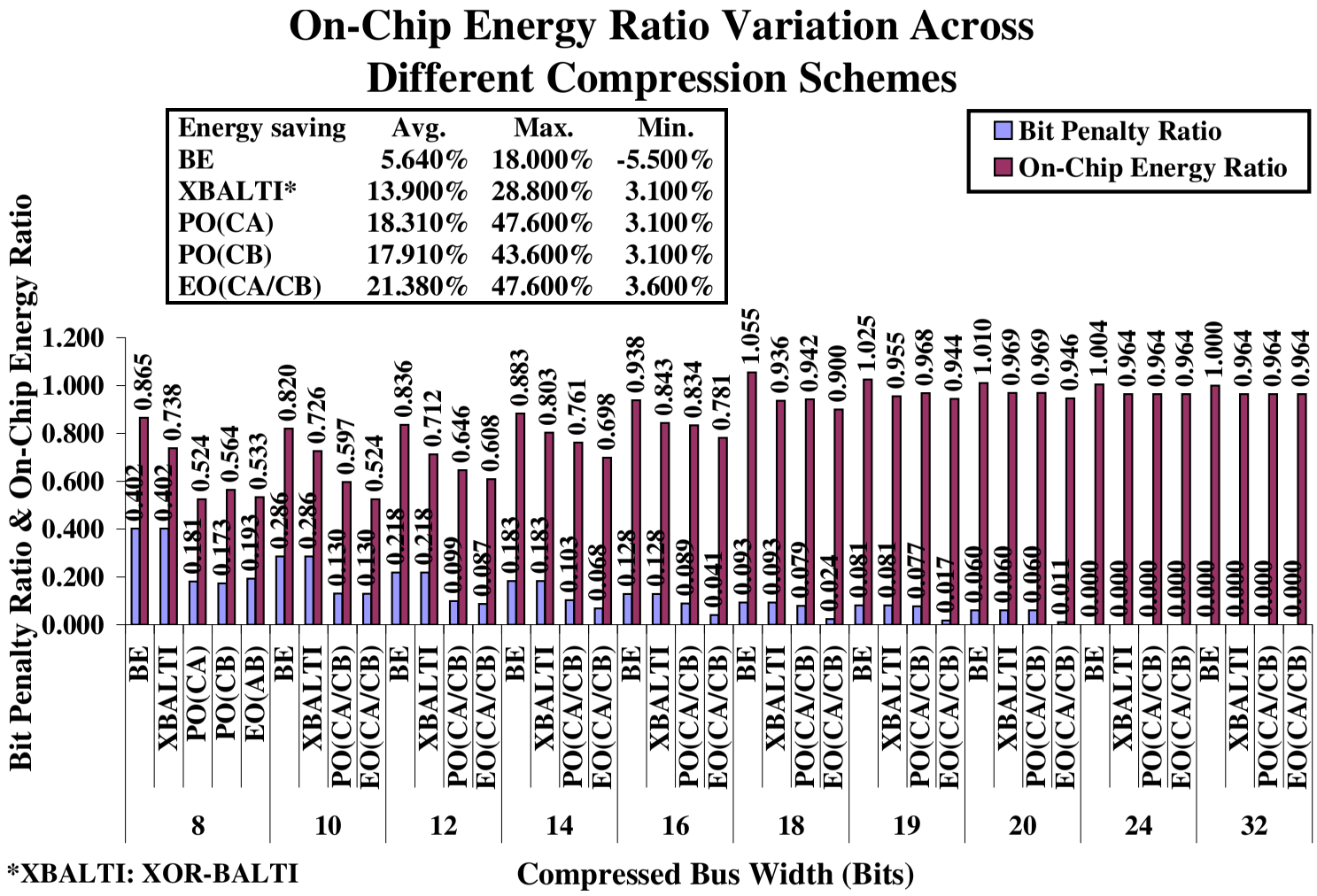


Figure 5.21: **On-Chip Energy Variation Across Different Compression Schemes.**

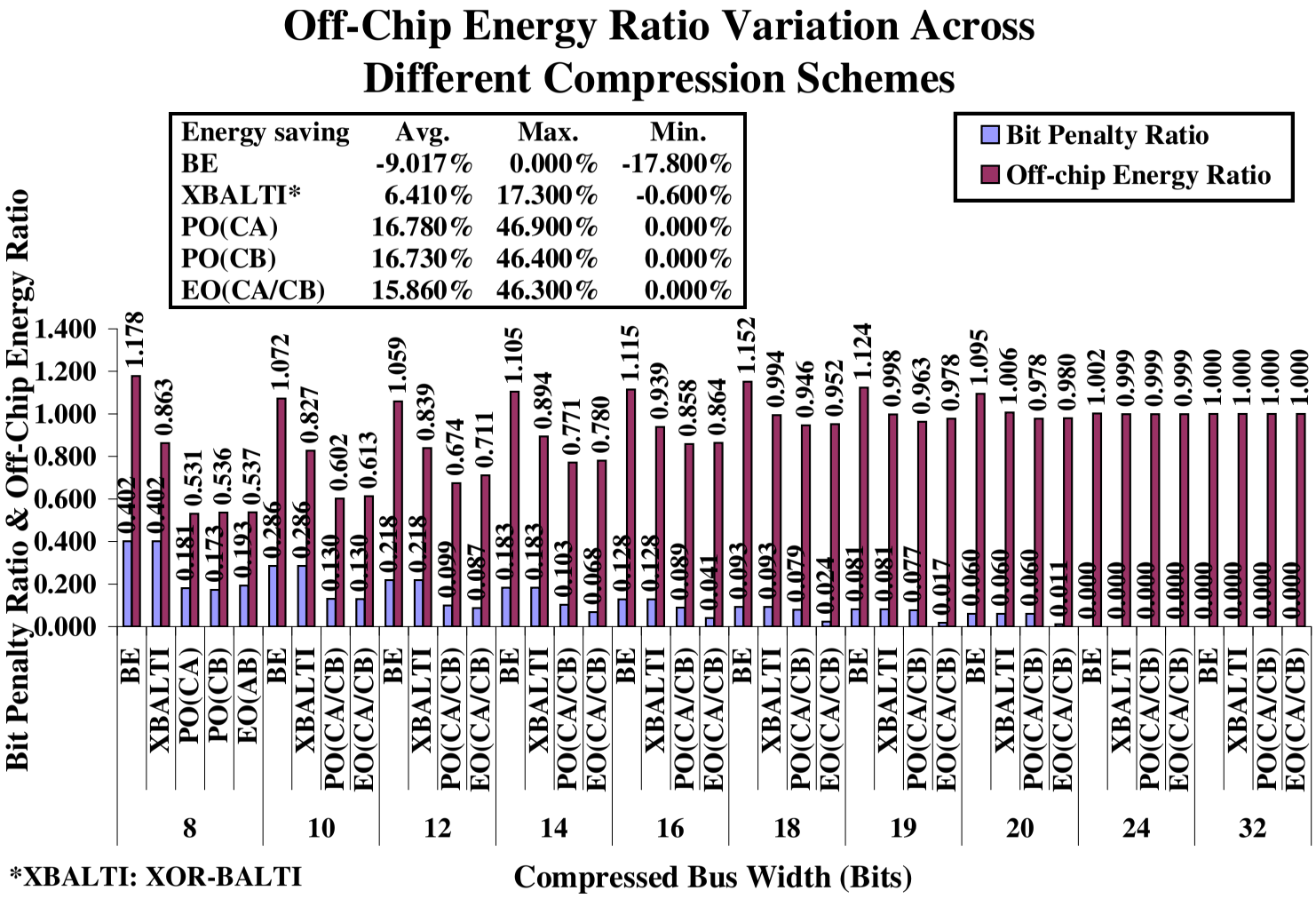


Figure 5.22: **Off-Chip Energy Variation Across Different Compression Schemes.**

Concatenate to generate the optimal solution for the entire tag field.

	Bus Width		
	8	10	12
	P.P.*, C.*	P.P., C.	P.P., C.
PO(CA, CB)	[6,14,22,32],[2,3,4,6] [7,14,21,32],[2,3,4,6]	[9,18,30],[2,3,5]	[11,28],[2,4]
EO(CA,CB)	[13,32],[3,5]	[12,30],[3,4]	[9,28],[2,4]
*P.P.:Partition points and C.:Cycles			

	Bus Width		
	14	16	18
	P.P.*, C.*	P.P., C.	P.P., C.
PO(CA, CB)	[13,26],[2,3]	[15,24],[2,3]	[17,22],[2,3]
EO(CA,CB)	[9,26],[2,3]	[7,24],[2,3]	[5,22],[2,3]

	Bus Width			
	19	20	24	32
	P.P., C.	P.P., C.	P.P., C.	P.P., C.
PO(CA, CB)	[18,21],[2,3]	[20],[2]	[16],[2]	[8],[2]
EO(CA,CB)	[4,21],[2,3]	[3,20],[2,2]	[5,16],[2,2]	[1,8],[2,2]

Table 5.1: **Partition for PM Performance and Energy Optimization.**

Our simulations show that both BE and PM are more effective for narrower compressed buses as far as the energy is concerned. Overall, PM performances much better than BE in terms of performance and energy. Fig. 5.1 lists the partition points and corresponding cycles for both performance and energy optimized designs. As shown in Fig. 5.20, the PM performance and energy optimized compression only cause 0.4% and 0.5% extra cycle penalty, which is much better than 1.085% when BE is applied. For the 8-bit compressed bus, the extra cycle penalty is 1.8% for PM performance optimized compression and 4.9% for BE.

Figs. 5.21 and 5.22 show that 16% and 24% more energy saving for on-chip and off-chip

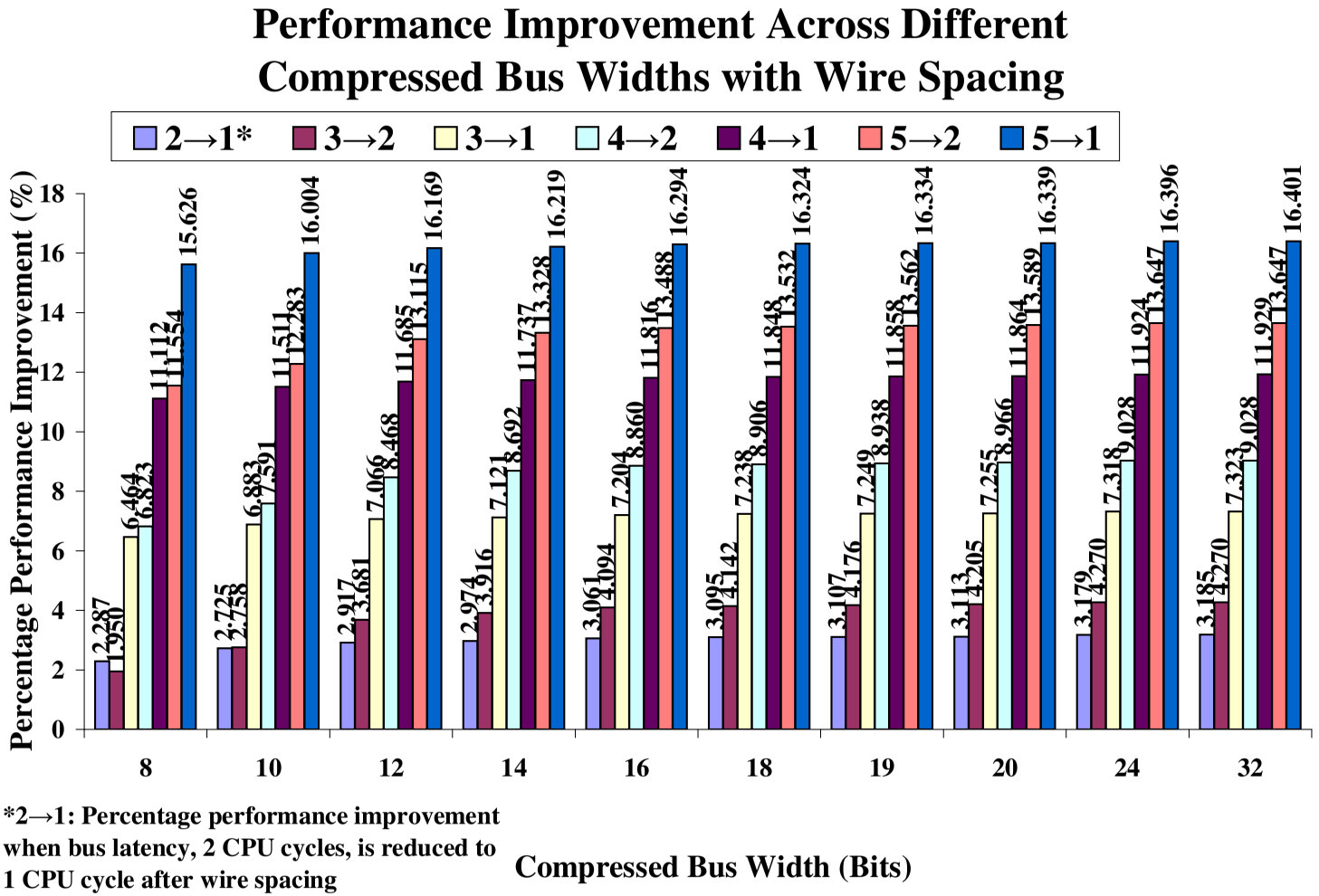


Figure 5.23: **Performance Improvement Across Different Compressed Bus Widths With Wire Spacing.**

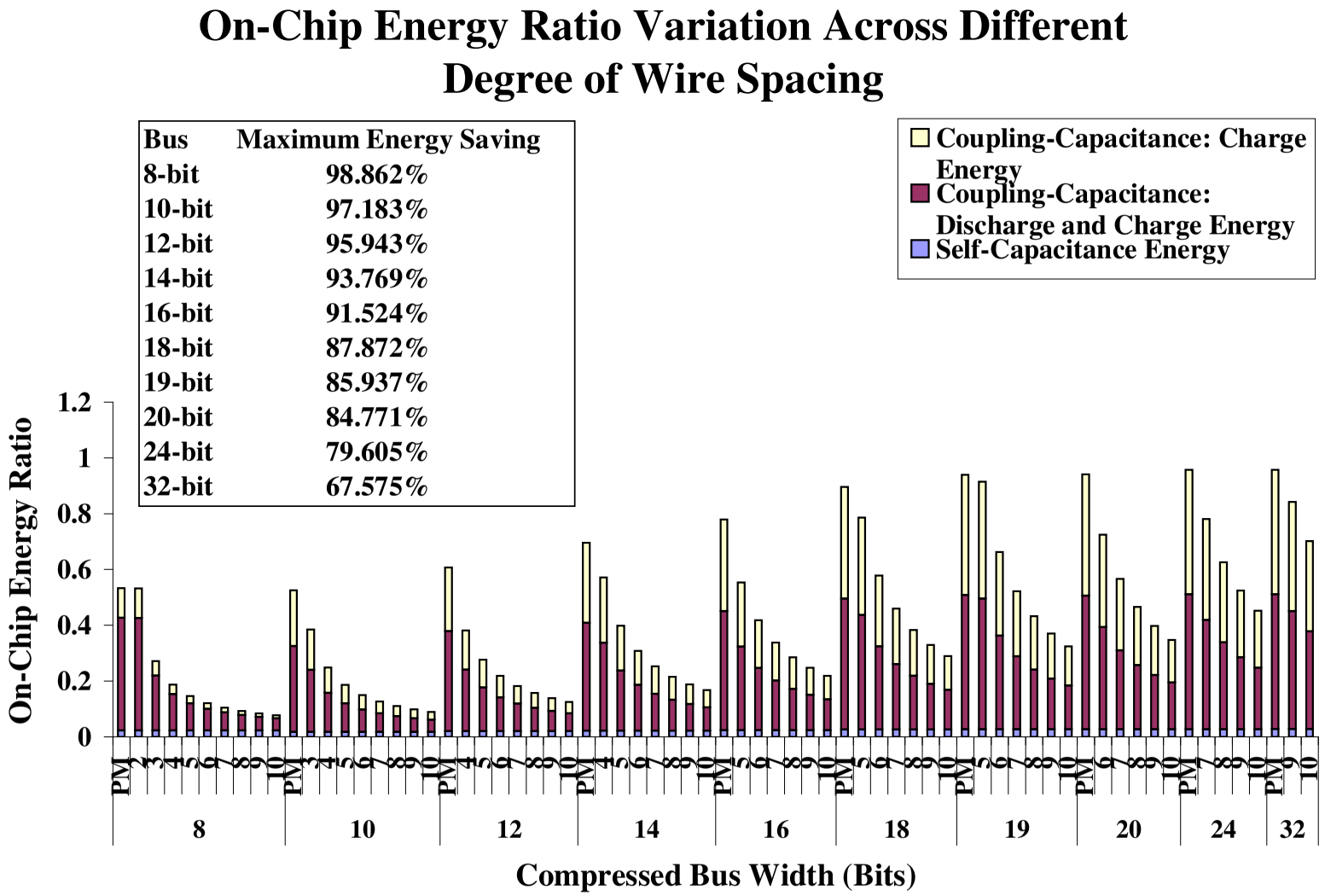


Figure 5.24: **On-Chip Energy Reduction Across Different Compressed Bus Widths With Wire Spacing.**

buses can be obtained with PM energy optimized compression than BE. The bit penalty ratio is the ratio between the bit penalty of the compressed address and the bit penalty of the original address when the compressed bus is used. Also, the average miss penalty and the average bit penalty, which are used to decide the best partitions for performance and energy optimization, correlate very well with the actual performance and the energy dissipation, respectively.

Similar to the study in Sec. 3.9, we also apply different degrees of wire spacing on top of PM to improve performance and on-chip energy further. Fig. 5.23 shows that PM with wire spacing can improve the performance by 2% to 16%. As shown in Fig. 5.24, for 8-bit compressed bus, up to 98% on-chip energy can be saved using PM with wire spacing, which is 51% more than using PM itself. Even for 32-bit compressed bus, 68% on-chip energy reduction can be obtained. On average, 88% on-chip energy can be saved using PM with wire spacing.

5.9 Conclusions

In this work, we have shown that substantial energy reductions are possible by judiciously arranging and/or encoding different bit-fields in compressed addresses thus reducing both self and coupling transitions. Our simulations show that the performance penalty for address compression is very modest: about 0.83-0.004% for compressed address bus widths of 14–36 bits. We proposed six different techniques to reduce energy in compressed address buses and our best scheme results in overall energy reduction of 14.7% for on-chip and 7.0%

for off-chip buses compared to transmission of uncompressed addresses. Address compression alone contributes to only half of this reduction; our proposed techniques contributes the rest. Also, in many cases where address compression failed to yield energy benefits, our schemes were successful in reducing energies. With less than 1% performance penalty for most bus widths and up to 28.8% energy reduction possible by combining address compression schemes with our proposed transmission techniques, excellent energy efficiencies can be obtained for on-chip address buses. Further, energy reductions can be obtained by applying our proposed partial-match compression and wire spacing. With PM, the extra cycle penalty is only 0.5%, which is much better than 1.085% when BE is applied. On average, 21% and 15% energy savings can be obtained for on-chip and off-chip buses using PM, respectively. In addition, with wire spacing, PM can improve the performance by 2% to 16% and provide 88% on-chip energy savings on average.

Chapter 6

Conclusions

In this dissertation, we have made significant strides in the analysis of memory system compression potential and the design of information pattern aware strategies for nanometer-scale address buses to improve performance, cost, and power consumption. This was done in the context of real-world benchmark suites such as SPEC CPU2000 and using execution-driven simulators like Sun Microsystems' Shade and SimpleScalar. We have completed the most comprehensive analysis to date of the potential benefits that address, instruction, and data compression may yield at all levels of the memory system considering a wide variety of factors in Chapter 2. Then, in Chapter 3, we presented a technique called HOC, in which narrow bus widths are used for underutilized buses to reduce cost, novel encoding schemes are employed to reduce power consumption, and concatenation and other methods are applied to mitigate performance penalty. Next, we exploited information and energy redundancy of information transmitted on memory system address buses for performance, power, and cost

improvements. We presented a detailed analysis of the performance, energy, and cost trade-offs possible with two cache-based dynamic address compression schemes in Chapter 4 and a highly energy- and performance-efficient dynamic address compression methodology for nanometer-scale address buses in Chapter 5.

6.1 Key Results

Below we summarize the main contributions of this dissertation.

- We comprehensively analyzed the redundancy in the information (addresses, instructions, and data) stored and exchanged between the processor and the memory system and evaluated the potential of compression in improving performance, power consumption, and cost of the memory system. Analysis on traces obtained with Sun Microsystems' Shade simulator simulating SPARC-V9 executables of eight integer and seven floating-point programs in the SPEC CPU2000 benchmark suite and five programs from the MediaBench suite and analyzed using Markov entropy models, existing compression schemes, and CACTI 3.0 and SimplePower timing, power, and area models showed good potential for compression at all levels of the memory system. Simulations results also showed that, even in current fabrication technologies, well-designed compression schemes can provide overall benefits in performance, power, and cost that outweigh their overheads (extra time, logic, and power for compression and decompression). These benefits will further grow in future technologies since the speed, size, and power consumption of logic (which is used to perform compres-

sion/decompression) are improving and are projected to improve at a much higher rate compared to those of interconnect (which is used to communicate information via buses), both on-chip and, especially, off-chip.

- Minimizing the area/cost and power consumption of communication components (address, instruction, and data buses and associated hardware like I/O pins, pads, and buffers) is becoming important in modern microprocessors. Currently, utilization of buses is not taken into account during design of many bus systems. This may lead to underutilization of many buses during actual operation. We proposed a scheme that exploits the underutilization of address buses to result in a cost-effective and energy-efficient bus system design. This is accomplished by using buses of narrow width, energy-efficient transmission formats, and wire spacing. On average, 16% on-chip energy reduction can be obtained using our best transmission technique, T6, and off-chip energy can be improved by 45% with T4 compared to the baseline transmission format. HOC with wire spacing results in 61% reduction of the wire delay, up to 0.8% to 15% performance improvement, and 60% on-chip energy savings.

- Dynamic address compression schemes that exploit address locality can help reduce both address bus energy and cost simultaneously with only a small performance penalty. We investigated two such schemes and determined their optimal parameters that result in the highest area/cost reductions and least performance penalty for various address buses (both on- and off-chip) in current systems. For addresses compressed with these

schemes, we studied energy reduction of buses in current and future nanometer technology nodes. Results show that using address compression will result in only small performance overheads (less than 1% for a 38-bit bus to 14 bits) and reduce bus energy dissipation by as much as 13% when applied to on-chip buses in current technologies.

- To realize energy-efficient buses in current nanometer-scale technologies, techniques like compression or encoding that exploit information redundancy have been explored. However, available compression techniques for buses do not always ensure energy-efficient transmission of compressed information. We presented various techniques that can be used with existing compression schemes for buses to ensure the best energy-efficiency for compressed information transmission. Our best scheme, applied to a stream of 38-bit addresses issued in a typical microprocessor, yields about 14.3% energy reduction on the average across a wide range of compressed bus widths ranging from 8 to 32 bits. Our proposed techniques especially perform better (up to 27.4% energy reduction is obtained) for narrower bus widths in the range 8–16 bits. Further energy reductions can be obtained by applying our proposed partial-match compression and wire spacing. With PM, extra cycle penalty is only 0.5%, which is much better than 1.085% when BE is applied. On average, 21% and 15% energy savings can be obtained for on-chip and off-chip buses using PM, respectively. In addition, with wire spacing, PM can improve performance by 2% to 16% and provide 88% on-chip energy savings on average.

6.2 Future Work

In the future, we intend to design compression schemes for instruction and data buses by taking into account characteristics of different instruction formats and fields and data types, like character, integer, and floating point. Currently, the schemes developed are meant mainly for communication components, especially, nanometer-scale address buses. We plan to extend and apply most ideas to storage components, like caches, TLB, and main memory.

The increased complexity of computing systems not only involves challenges to system design, but also to testability. Test data volume is increasing dramatically as technology is moving into nanometer regime, which makes testing time longer, consumes more testing power, and increases testing cost [78, 54]. Testing is becoming a critical bottleneck in improving time-to-market. Another fruitful research direction will be to explore test data compression to reduce size of test data and minimize its effect on testing time, power consumption, and cost.

Bibliography

[1] Advanced RISC Machines Ltd (ARM). *An Introduction to Thumb*, March 1995. Available at: http://www.arm.com.

[2] Y. Aghaghiri, F. Fallah, and M. Pedram. Irredundant Address Bus Encoding for Low Power. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 322–327, August 2001.

[3] K. Basu, A. Choudhary, J. Pisharath, and M. Kandemir. Power Protocol: Reducing Power Dissipation on Off-Chip Data Buses. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, pages 345–355, November 2002.

[4] J.C. Becker, A. Park, and M. Farrens. An Analysis of the Information Content of Address Reference Streams. In *Proceedings of the International Conference on Microarchitecture*, pages 19–24, November 1991.

[5] L. Benini, G. De Micheli, E. Macii, and M. Poncino. Selective Instruction Compression for Memory Energy Reduction in Embedded Systems. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 206–211, August 1999.

[6] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano. Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-power Microprocessor-based Systems. In *Proceedings of Great Lakes Symposium on VLSI*, pages 77–82, March 1997.

[7] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu. New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Design. In *Proceedings of the*

IEEE Custom Integrated Circuits Conference, pages 201–204, June 2000.

[8] W.-C. Cheng and M. Pedram. Memory Bus Encoding for Low-power: A Tutorial. In *Proceedings of International Symposium on Quality Electronic Design*, pages 199–204, March 2001.

[9] W.-C. Cheng and M. Pedram. Power Optimal Encoding for a DRAM Address Bus. *IEEE Transactions on VLSI Systems*, 10(2):109–118, April 2002.

[10] D. Citron. Exploiting Low Entropy to Reduce Wire Delay. *Computer Architecture Letters*, 3, January 2004.

[11] D. Citron and L. Rudolph. Creating a Wider Bus using Caching Techniques. In *Proceedings of International Symposium on High Performance Computer Architecture*, pages 90–99, January 1995.

[12] B. Cmelik and D. Keppel. SHADE: A Fast Instruction-set Simulator for Execution Profiling. *ACM SIGMETRICS Performance Evaluation Review*, 22(1):128–137, May 1994.

[13] J. Cong, L. He, C.-K. Koh, and Z. Pan. Global Interconnect Size and Spacing with Consideration of Coupling Capacitance. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 628–633, November 1997.

[14] T. M. Conte, S. Banerjia, S. Y. Larin, K. N. Menezes, and S. W. Sathaye. Instruction Fetch Mechanisms for VLIW Architectures with Compressed Encodings. In *Proceedings of the Annual Symposium on Computer Architecture*, pages 201–211, December 1996.

[15] K.D. Cooper and N. McIntosh. Enhanced Code Compression for Embedded RISC Processors. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 139–149, May 1999.

[16] Standard Performance Evaluation Council. SPEC CPU2000 Benchmark Suite Ver1.2. http://www.specbench.org/cpu2000, 2000.

[17] S. Debray, W. Evans, R. Muth, and B. de Sutter. Compiler Techniques for Code Compaction. *Transactions on Programming Languages and Systems*, 22(2):378–415, March 2000.

[18] R. Desikan, D.C Burger, S.W. Keckler, and T.M. Austin. Sim-alpha: a Validated, Execution-Driven Alpha 21264 Simulator. Technical Report TR-01-23, The University of Texas at Austin, Department of Computer Sciences, 2001.

[19] J. Ernst, W. Evans, C.W. Fraser, S. Lucco, and T.A. Proebsting. Code Compression. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 358–365, June 1997.

[20] M. Farrens and A. Park. Dynamic Base Register Caching: A Technique for Reducing Address Bus Width. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 128–137, May 1991.

[21] P.A. Franaszek and J.T. Robinson. Design and Analysis of Internal Organizations for Compressed Random Access Memories. Technical Report IBM Research Report RC 21146(94535)20OCT98, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, October 1998.

[22] C.W. Fraser, E.W. Myers, and A.L. Wendt. Analyzing and Compressing Assembly Code. *SIGPLAN Notices*, 19(6):117–121, June 1984.

[23] M. Game and A. Booker. Codepack[tm]: Code Compression for PowerPC Processors. IBM White Paper available at: http://www-3.ibm.com/chips/techlib/techlib.nsf/products/CodePack, May 2000.

[24] D.W. Hammerstrom and E.S. Davidson. Information Content of CPU Memory Referencing Behavior. In *Proceedings of the 4th Annual Symposium on Computer Architecture (ISCA-4)*, pages 184–192, march 1977.

[25] J. Henkel and H. Lekatsas. A2BC: Adaptive Address Bus Coding for Low-power Deep Sub-micron Designs. In *Proceedings of Annual ACM/IEEE Design Automation Conference*, pages 744–749, June 2001.

[26] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach, Third edition*. Morgan Kaufmann Publishers Inc., 2003.

[27] J. Hoogerbrugge, L. Augusteijn, J. Trum, and R. van de Wiel. A Code Compression System Based on Pipelined Interpreters. *Software Practice and Experience*, 29(11):1005–1023, September 1999.

[28] ITRS. International Technology Roadmap for Semiconductors, 2001 edition, 2001.

[29] I. Kadayif and M. Kandemir. Instruction Compression and Encoding for Low-power Systems. In *Proceedings of the IEEE International ASIC/SOC Conference (ASIC/SOC'02)*, pages 301–305, September 2002.

[30] K. Kant and R. Iyer. Design and Performance of Compressed Interconnects for High

Performance Servers. In *Proceedings of International Conference on Computer Design*, pages 164–169, October 2003.

[31] H. Kaul, D. Sylvester, and D. Blaauw. Issues in Crosstalk: Active Shielding of Rlc Global Interconnects. In *Proceedings of ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 98–104, 2002.

[32] T.M. Kemp, R.K. Montoye, J.D. Harper, J.D. Palmer, and D.J. Auerbach. A Decompression Core for PowerPC. *IBM Journal of Research and Development*, 42(6):807–811, November 1998.

[33] K.-Y. Khoo and A.N. Willson. Charge Recovery on a Databus. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 185–189, 1995.

[34] K.W. Kim, K.H. Back, N. Shanbhag, C.L. Liu, and S.M. Kang. Coupling-driven Signal Encoding Scheme for Low-power Interface Design. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 318–321, November 2000.

[35] D. Kirovski, J. Kin, and W.H. Mangione-Smith. Procedure Based Program Compression. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, pages 204–213, December 1997.

[36] K.D. Kissell. MIPS16: High-density MIPS for the Embedded Market. http://www.mips.com/Documentation/MIPS16whitepaper.pdf, 1997.

[37] M. Kjelso, M. Gooch, and S. Jones. Empirical Study of Memory-data: Characteristics and Compressibility. *IEE Proceedings on Computers and Digital Techniques*, 145(1):63–67, January 1998.

[38] M. Kozuch and A. Wolfe. Compression of Embedded System Programs. In *Proceedings of International Conference on Computer Design*, pages 270–277, October 1994.

[39] T. Lang, E. Musoll, and J. Cortadella. Extension of the Working-zone Encoding Method to Reduce the Energy on the Microprocessor Data Bus. In *Proceedings of International Conference on Computer Design*, pages 414–419, October 1998.

[40] C. Lee, M. Potkonjak, and W.H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *Proceedings of the Annual Symposium on Computer Architecture*, pages 330–335, December 1997.

[41] J.-S. Lee, W.-K. Hong, and S.-D. Kim. Design and Evaluation of a Selective Compressed Memory System. In *Proceedings of International Conference on Computer Design*, pages 184–191, October 1999.

[42] C. Lefurgy and T. Mudge. Code Compression for DSP. Technical Report CSE-TR-380-98, EECS Department, University of Michigan, Ann Arbor, MI, 1998.

[43] H. Lekatsas, J. Henkel, and W. Wolf. Code Compression for Low Power Embedded System Design. In *Proceedings of Annual ACM/IEEE Design Automation Conference*, pages 294–299, June 2000.

[44] H. Lekatsas and W. Wolf. Random Access Decompression using Binary Arithmetic Coding. In *Prodeedings of Data Compression Conference*, pages 306–315, March 1999.

[45] H. Lekatsas and W. Wolf. SAMC: A Code Compression Algorithm for Embedded Processors. *IEEE Transactions on Computer-aided Design*, 18(12):1689–1701, December

1999.

[46] L. Lev and P. Chao. Down to the wire: Requirements for Nanometer Design Implementation. White Paper, Cadence Design Systems Inc., 2002.

[47] S.Y. Liao, S. Devadas, and K. Keutzer. Code Density Optimization for Embedded DSP Processors Using Data Compression Techniques. In *Proceedings of Conference on Advanced Research in VLSI*, pages 393–399, March 1995.

[48] J. Liu, N.R. Mahapatra, K. Sundaresan, S. Dangeti, and B.V. Venkatrao. Memory System Compression and Its Benefits. In *Proceedings of the 15th Annual IEEE International ASIC/SOC Conference*, pages 41–45, September 2002.

[49] N.R. Mahapatra, J. Liu, and K. Sundaresan. Hardware-Only Compression of Underutilized Address Buses: Design and Performance, Power Consumption and Cost Analysis. In *Proceedings of IEEE International Conference on Computer Design*, pages 234–239, October 2003.

[50] N.R. Mahapatra, J. Liu, K. Sundaresan, S. Dangeti, and B.V. Venkatrao. The Potential of Compression to Improve Memory System Performance, Power Consumption, and Cost. In *Proceedings of IEEE Performance, Computing and Communications Conference*, pages 343–350, April 2003.

[51] E. Musoll, T. Lang, and J. Cortadella. Working-zone Encoding for Reducing the Energy in Microprocessor Address Buses. *IEEE Transactions on VLSI Systems*, 6(4):568–572, December 1998.

[52] A. Park and M. Farrens. Address Compression through Base Register Caching. In

Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture, pages 193–199, November 1990.

[53] J.M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall Inc., 2002.

[54] J. Rajski and J. Tyszer. Test Data Compression and Compaction for Embedded Test of Nanometer Technology Designs. In *Proceedings of International Conference on Computer Design*, pages 331–336, 2003.

[55] Y. Shin and K. Choi. Narrow Bus Encoding for Low Power Systems. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 217–220, January 2000.

[56] P. Shivakumar and N.P. Jouppi. CACTI 3.0: An Integrated Cache Cycle Timing, Power, and Area Model. Technical Report WRL Research Report 2001/2, Compaq Western Research Laboratory, August 2001.

[57] K. Skadron, P.S. Ahuja, M. Martonosi, and D.W. Clark. Selecting a Single, Representative Sample for Accurate Simulation of SPECint Benchmarks. *IEEE Transactions on Computers*, 48(11):1260–1281, November 1999.

[58] P.P. Sotiriadis and A. Chandrakasan. Low Power Bus Coding Techniques Considering Inter-wire Capacitances. In *Proceedings of Custom Integrated Circuits Conference*, pages 414–419, May 2000.

[59] P.P. Sotiriadis and A. Chandrakasan. A Bus Energy Model For Deep Sub-Micron Interconnects. *IEEE Transactions on VLSI Systems*, 10(3):341–350, June 2002.

[60] M.R. Stan and W.P. Burleson. Bus-invert Coding for Low-power I/O. *IEEE Transac-*

tions on VLSI Systems, 3:49–58, March 1995.

[61] M.R. Stan and W.P. Burleson. Low-power Encodings for Global Communication in CMOS VLSI. *IEEE Transactions on VLSI Systems*, 5:444–455, December 1997.

[62] C.L. Su, C.Y. Tsui, and A.M. Despain. Low Power Architecture Design and Compilation Techniques for High-performance Processors. Technical Report ACAL-TR-94-01, Advanced Computer Architecture Laboratory, University of Southern California, 1994.

[63] D.C. Suresh, B. Agarwal, J. Yang, W. Najjar, and L. Bhuyan. Power Efficient Techniques for Off-Chip Data Buses. In *Proceedings of Workshop on Compiler and Architecture Support for Embedded Systems*, pages 267–275, October 2003.

[64] R. B. Tremaine, P.A. Franaszek, J.T. Robinson, C.O. Schulz, T.B. Smith, M.E. Wazlowski, and P.M. Bland. IBM Memory eXpansion Technology (MXT). *IBM Journal of Research and Development*, 45(2):271–285, March 2001.

[65] L. Villa, M. Yang, and K. Asanovic. Dynamic Zero Compression for Cache Energy Reduction. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, pages 214–220, December 2000.

[66] J.L. Wang and R.W. Quong. The Feasibility of Using Compression to Increase Memory System Performance. In *Proceedings of International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 107–113, January 1994.

[67] D.L. Weaver and T. Germond, editors. *The SPARC Architecture Manual, Version 9*. Prentice Hall, 2000.

[68] V. Wen, M. Whitney, Y. Patel, and J.D. Kubiotowicz. Exploiting Prediction to Reduce Power on Buses. In *Proceedings of International Symposium on High Performance Computer Architecture*, pages 2–13, February 2004.

[69] Wayne Wolf. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann Publishers Inc., 2001.

[70] A. Wolfe and A. Channin. Executing Compressed Programs on an Embedded RISC Architecture. In *Proceedings of the Annual Symposium on Computer Architecture*, pages 81–91, December 1992.

[71] Y. Xie, W. Wolf, and H. Lekatsas. Code Compression for VLIW using Variable-to-fixed Coding. In *Proceedings of the International Symposium on System Synthesis*, pages 138–143, October 2002.

[72] J. Yang, Y. Zhang, and R. Gupta. Frequent Value Compression in Data Caches. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, pages 258–265, November 2000.

[73] W. Ye, N. Vijaykrishnan, M. Kandemir, and M.J.Irwin. The Design and Use of Simplepower: a Cycle-accurate Energy Estimation Tool. In *Proceedings of Annual ACM/IEEE Design Automation Conference*, pages 340–345, June 2000.

[74] Y. Yoshida, B. Y. Song, H. Okuhata, T. Onoye, and I. Shirakawa. An Object Code Compression Approach to Embedded Processors. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 265–268, August 1997.

[75] H. Zhang and J.Rabaey. Low-Swing Interconnect Interface Circuits. In *Proceedings*

of International Symposium on Low Power Electronics and Design, pages 161–166, August 1998.

[76] Y. Zhang, R. Y. Chen, W. Ye, and M.J.Irwin. System Level Interconnect Power Modeling. In *IEEE International ASIC/SoC Conference*, pages 289–293, September 1998.

[77] Y. Zhang, J. Lach, K. Skadron, and M.R. Stan. Odd/even Bus Invert with Two-Phase Transfer for Buses with Coupling. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 80–83, August 2002.

[78] Y. Zorian, S. Dey, and M. Rodgers. Test of Future System-on-chips. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 392–398, 2000.