

The Potential of Compression to Improve Memory System Performance, Power Consumption, and Cost*

NIHAR R. MAHAPATRA[†], JIANGJIANG LIU[†], KRISHNAN SUNDARESAN[†],
SRINIVAS DANGETI[‡], AND BALAKRISHNA V. VENKATRAO[‡]

[†]{mahapatr, jliu3, ks48}@cse.buffalo.edu

[‡]{srinivas.dangeti, balakrishna.venkatrao}@eng.sun.com

Abstract

Continuing exponential growth in processor performance, combined with technology, architecture, and application trends, place enormous demands on the memory system to allow information storage and exchange at a high-enough performance (i.e., to provide low latency and high bandwidth access to large amounts of information), at low power, and cost-effectively. This paper comprehensively analyzes the redundancy in the information (addresses, instructions, and data) stored and exchanged between the processor and the memory system and evaluates the potential of compression in improving performance, power consumption, and cost of the memory system. Traces obtained with Sun Microsystems' Shade simulator simulating SPARC executables of nine integer and six floating-point programs in the SPEC CPU2000 benchmark suite and analyzed using Markov entropy models, existing compression schemes, and CACTI 3.0 and SimplePower timing, power, and area models yield impressive results.

1 Introduction

Performance, power consumption, and cost are probably the three most important parameters that drive computer system design today. While their relative importance varies in these systems, all three parameters are recognized as important. Thus, while performance is most important in high-end multiprocessors, performance/cost drives the general-purpose processor market, and power consumption plays a more significant role in embedded and wireless applications.

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 information (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 [8]. 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 [8]. Moreover, in deep-submicron regime, 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 [20]. Consequently, increasingly more fraction of the system power consumption and cost is due to the memory system compared to the computation system [24]. Thus, the memory system is becoming an increasing bottleneck as designers strive towards higher performance, cost-effective, and power-efficient system designs.

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 paper, 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 and access time of storage components), power consumption, and cost. The organization of the remainder of the paper is as follows. Section 2 discusses the advantages of a compressed memory system architecture and its feasibility. Section 3 provides a brief overview of previous work related to the use of compression for buses, code, cache, and main memory. Section 4 describes the simulation environment, analysis tools, and methods we used in our study. Section 5 presents detailed results of our analysis. Finally, we conclude in Section 6.

2 The Case for Compressed Memory System Architectures

In this section, we discuss the opportunities for compression present in the memory system, the benefits of applying compression and the challenges to be overcome, and then finally a useful way of classifying CMS architectures.

2.1 Opportunities for Compression

Compression of some source information consisting of a sequence of symbols is achieved by encoding the more frequent or likely symbols with shorter code words compared to the less frequent or likely symbols. In the memory system, redundancy exists in address, instruction, and data information. Instruction addresses issued by the processor to the L1 cache are typically sequential (very predictable and hence quite redundant), data addresses exhibit locality because of scanning of data arrays in loops, and tag fields of instruction and data caches correspond to blocks that have been recently accessed. Addresses issued by higher levels of the memory system become increasingly unpredictable, because they are caused by misses occurring in the lower level caches, but these addresses also exhibit temporal and spatial locality, although to lesser extents. In instructions,

*This research was supported by the US National Science Foundation under Grant No. 0102830. N.R. Mahapatra and J. Liu are with Dept. of Comp. Sci & Eng., University at Buffalo, The State Univ. of New York, Buffalo, NY 14260, K. Sundaresan with the Dept. of Electrical Eng., University at Buffalo, The State Univ. of New York, Buffalo, NY 14260, and S. Dangeti and B.V. Venkatrao are with the Processor Products Group, Sun Microsystems Inc., Palo Alto, CA 94303.

in addition to temporal and spatial locality, repetitions of instruction sequences, opcodes, registers, immediate constants, correlation between opcodes and registers and between opcodes and immediate constants also constitute redundancy. Finally, data primarily has redundancy present in the values communicated by data buses and stored in registers, data caches, and main memory since not all values are equally likely and values used by a program are typically of small magnitude.

2.2 Benefits and Feasibility 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 memory system, although substantial direct or indirect improvements can be expected in most areas of the system. As an example, by using compression in on-chip or off-chip buses, 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. 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. In storage components like caches, compression can also be used possibly to improve 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. 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.

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. 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.

2.3 CMS Architectures and Degree of Specialization

Depending upon how specialized the data set from which symbol statistics are drawn for compression is, five important classes of CMS architectures, from the most to the least specialized, can be identified as follows. 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. (1) *Block-specific architecture*:

Symbol statistics used to compress a block are drawn from the same block. (2) *Memory-component-specific architecture*: Symbol statistics are drawn from the typical data set of a memory component and are used to compress each block of that component. (3) *Application-program-specific architecture*: Symbol statistics 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. (4) *Application-class-specific architecture*: Symbol statistics are drawn from application programs that belong to the same class. (5) *General architecture*: Symbol statistics are drawn from a broad range of applications and from all memory components that store and communicate the same type of information. It is possible to use different degrees of specialized statistical information to perform compression in different parts of the memory system. 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.

3 Related Work

Previous analytical research on finding the potential of address compression [2], instruction compression [23], program code compression [11], and main memory compression [10] lack a unified analysis of the improvement in the three important metrics, namely, performance, power consumption, and cost attainable with compression. Our earlier work [16] focussed briefly on the overall benefits of compression applied to the memory system. Specific compression schemes for addresses [18, 5], instructions and program code [25, 15, 13, 14, 28, 3], and cache and main memory [12, 26, 22] have been proposed. Current microprocessor cores have also adopted various compression techniques for code [7, 1, 9] and main memory [21]. Finally, various bus encoding techniques for reducing power consumption have been proposed [4] for address, data, and instruction buses.

3.1 Contributions of This Work

To our knowledge, this paper's 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 paper is not to present specific compression schemes but to estimate the extent of compression possible in various memory 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 hence the improvements in performance, power consumption, and cost attainable. We present results related to the compressibility of original, XOR, and offset instruction and data address traces, the effect of compression on cache access time, power consumption, and area, the relationship between compression ratio and bit fields and bit-field groupings, the effect of application class, degree of specialization, encoding and multiplexing, analysis tool, static vs. adaptive compression, multithreading, and the relationship between information content, compression ratio, and power consumption, among other things.

4 Simulation Methodology

In this section, we first discuss the target system and our simulation environment. This is followed by a description of the tools and methods used in our analysis.

4.1 Target System and Simulation Environment

Our target system has a memory hierarchy consisting of 32 integer and 32 floating-point registers, split instruc-

tion 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 our analysis, we modified the cachesim5 cache analyzer in SHADES [6] running on SPARC-V9 platform to collect the real-time traffic (addresses, instructions, and data) for the SPEC CPU2000 benchmark programs. We used benchmarks from the SPEC CPU2000 suite. To capture the characteristics of both integer and floating-point programs, we chose 9 integer and 6 floating-point benchmarks randomly out of 26 in the suite. Integer benchmarks (INT) we used include gcc, gzip, vortex, parser, crafty, twolf, mcf, bzip2, and vpr and floating-point benchmarks (FP) we used were applu, swim, wupwise, lucas, art, and ammp. We used the -O2 optimization flag, which does basic local and global optimization to compile these benchmarks from the SPEC2000 suite. Our methodology for selecting the sampling and warm up windows for these benchmarks is described in greater detail elsewhere [17].

4.2 Analysis Tools

We used various entropy measures and some available compression schemes to estimate the information content or compression ratio possible for our traces. 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. Given a set of symbols s_1, s_2, \dots, s_N and a source in which $M \leq N$ of these symbols actually occur, the measure that characterizes the compressibility of a symbol by its presence or absence in the trace, irrespective of the number of times the symbol repeats in the trace, is called *zero information entropy* and is given by the relation $H = \log_2 M$. If each symbol s_i occurs with probability $p(s_i)$, the *zeroth order Markov entropy* of the source data is given by the relation: $H_0 = -\sum_i p(s_i) \cdot \log(p(s_i))$. In *first order Markov entropy*, 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 are all considered. The first order Markov entropy of a source is given by: $H_1 = -\sum_i [p(s_i) \cdot \sum_j p(s_j|s_i) \cdot \log(p(s_j|s_i))]$.

Using the entropy values measured, the corresponding compression ratio can be computed by taking the ratio of entropy to the size of a symbol (32 for addresses and instructions and 64 for data) in the original uncompressed trace. Thus, 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. The default compression tool we used in some experiments, SAMC, is based on arithmetic coding combined with a precalculated Markov model [14]. We define *average SAMC compression ratio* over n benchmark traces as follows:

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

Gzip is a GNU utility for compression in UNIX systems. It uses Lempel-Ziv (LZ77) dictionary compression algorithm and replaces strings of characters with single codes. We used Gzip 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 similarly as above.

For CMOS technology, power consumption on a bus line is directly related to the number of transitions on it as bits are transmitted one after another over it. We use a methodology similar to the one used in SimplePower [27] 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 bitline of the bus and find the total average probability across all bitlines, which is a measure of the per-input switching activity of the bus in bits [29]. 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* for compressed traces as:

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

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.

4.3 Analysis Methods

We did 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. For instruction and data caches, we performed compression analysis by taking into account their average compressibility. This was done by considering only blocks that were loaded into and replaced during the sampling window of the simulation and writing the values of these blocks randomly in a trace file. A load and the next replacement of a block correspond to a time period during which it is resident in the cache (*cache residency time (CRT)*). The total CRT (TCRT) is the sum of CRTs of all blocks and we select for analysis a subset of the blocks listed in nonincreasing order of their CRTs corresponding to 85% and 95% TCRT. By considering the residency times of blocks, the trace file created reflects the average contents of the cache. Hence the compression ratios obtained would be those expected from a compression scheme that chooses encodings based on average symbol statistics, rather than dynamically as cache contents change. Therefore, the compression ratios we report are, in this sense, not optimistic. The procedure we use for generating these files is detailed in [17]. We estimate improvements possible in performance as reflected by compression ratio and improvements in power consumption

as reflected by the transition ratio for storage and communication components.

5 Results

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. We consider a default memory-component-specific architecture and a multiplexed bus in our simulations. 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 both entropy Markov experiments is 32 bits for address and instruction, 64 bits for data, and 20 bits for tag field in instruction caches. According to various issues we focus on in different subsections next, we always summarize results by averaging over all 15 (9 INT and 6 FP) benchmarks or by showing INT and FP results separately for specific components and we calculate the average compression ratios as mentioned earlier.

5.1 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 1(a) 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 D-cache data field, L1 I-cache tag field, registers, L1 I-cache data field, and main memory. This is to be expected since lot of the data in the data cache consists of small magnitude numbers with many redundant high-order bits and the tag field corresponds to the high-order portion of the address, where redundancy is high. Among communication components, the ordering is data bus, instruction bus, and address bus. A possible explanation for the higher redundancy in the data 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. We also observe that the ordering of the communication components in terms of power savings after compression (from least to most) is as follows: instruction bus, address bus, data bus. As can be seen from Figure 1(a), R_{SAMC} decreases in the same order, suggesting that more compression (i.e., fewer bits transmitted) leads to better power savings.

5.2 Register Compression Analysis

For register compression, we conducted zeroth order Markov analysis of 6 INT and 6 FP over all 32 integer registers and 32 floating-point registers. Figure 1(b) shows the compression ratio for each register. The average integer register compression ratio is 0.12 and floating-point register compression is 0.156. Note that integer register 0 has a hardwired value of 0. This means that integer registers are more compressible than floating-point registers. This is to be expected because floating point data, in general, are relatively less redundant since their fraction field bits are more fully used, where as integer data have more redundancies due to sign extension of their most significant bits. 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 0.26, which implies registers can, on average, be compressed to

about one-fourth of their original size using a very good zeroth-order compression scheme. Although integer registers do contain pointer values that can have large magnitudes, there is still a lot of redundancy present in them for two reasons. First, each pointer typically points to many actual data items, which are normally small magnitude numbers that have a lot of redundancy. Second, pointers point to roughly a similar region in memory as they are dynamically allocated and hence many of their high-order bits will be the same and hence redundant.

5.3 Cache Compression Analysis

Instruction cache compression: As we mentioned earlier, we generated 85% TCRT, 95% TCRT, and 100% TCRT trace files for instruction cache blocks to determine instruction cache compression. It can be seen from Fig. 1(c) that the compression ratios for the three traces are not very different. This means that that an 85% TCRT trace, which is easier to collect is accurate enough for the purpose of determining cache compressibility. We found that the instruction portions of L1 and L2 caches are almost equally compressible, with L1 being slightly more compressible. This may be because the L1 I-cache contains a more frequent symbol set (of instructions) and the L2 cache, in addition to storing the contents of L1, also contains additional symbols (instructions) that are relatively infrequent. On average, zeroth order Markov gave us around 0.26 compression ratio and 0.07 was obtained with first order Markov, which means theoretically we could reduce cache size by 4 to 14 times by applying cache compression or store that much more information in the same area.

Data cache compression: We conducted experiments for blocks stored in data cache in a similar manner as described for instruction caches above. Here also, we report results for 85%, 95%, and 100% TCRT traces. We observe from Fig. 1(c) that in comparison with I-caches, D-caches are more compressible with zeroth order as well as first order methods. Also, contrary to what was observed for I-caches, L2 D-caches are found to be more compressible than L1 D-caches, possibly because of more blocks having zero/uninitialized values in them.

5.4 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 Figure 1(d), we find that the compression potential of cache decreases with increasing cache size. For example, zeroth (first) order Markov compression ratio increases from 0.25 (0.06) to 0.28 (0.09) with the cache size changing from 8 KB to 128 KB. 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.

Compression ratio improves when we increase block size as shown in Figure 1(e). 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. Zeroth (first) order Markov compression ratio decreases from 0.27 (0.08) to 0.23 (0.05) with block size increasing from 8 bytes to 128 bytes. Finally, we found that cache associativity has negligible impact on compression ratio.

5.5 Cache Compression and Cache Access Time, Power Consumption, and Area

To measure the effect of compression on other parameters like access time, power consumption, and area of the

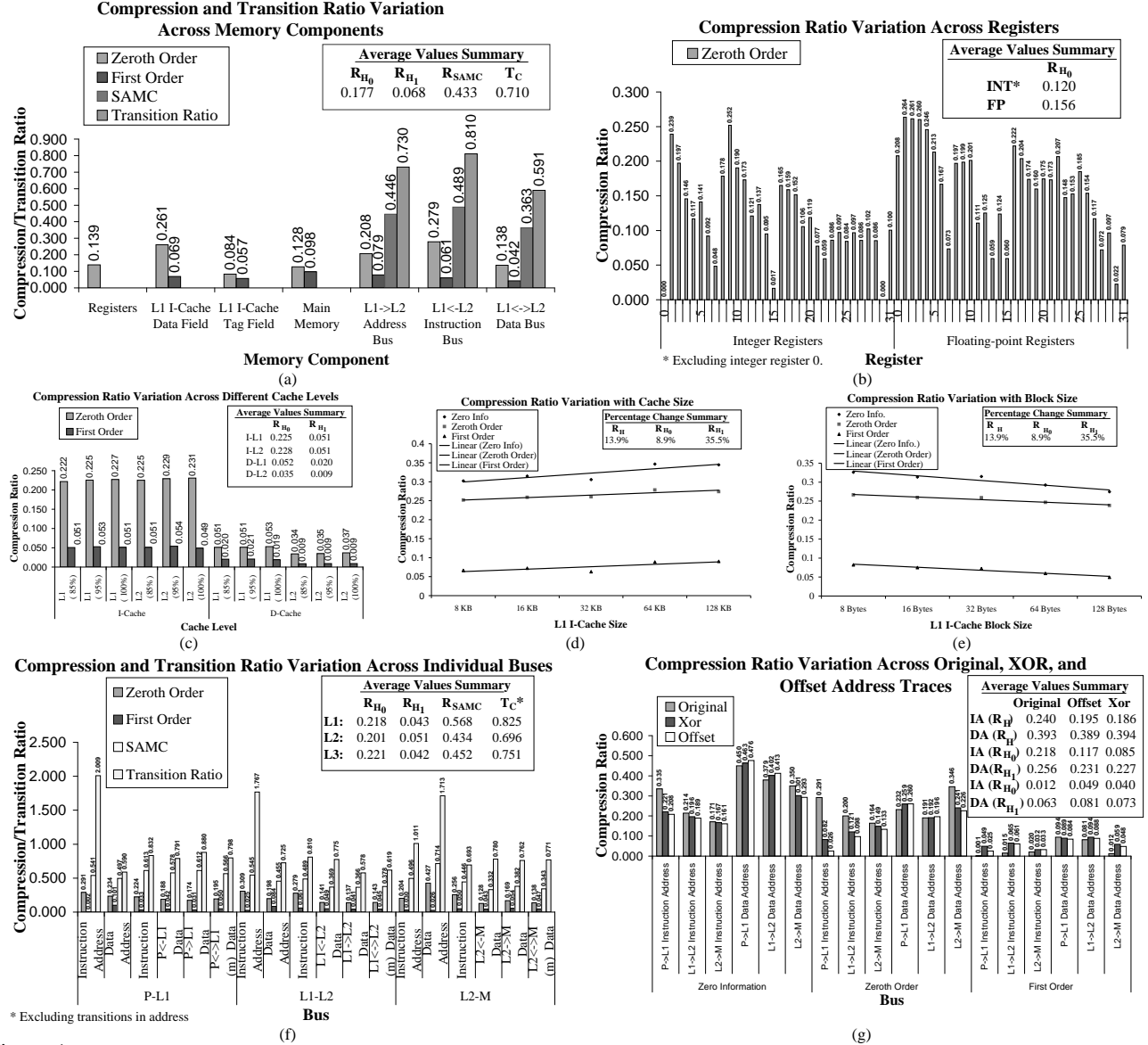


Figure 1: (a) **Overall Memory System Analysis:** Compression ratio variation across memory system components. (b) **Register Compression Analysis:** Average register compression analysis summary for 32 integer registers and 32 floating-point registers across 6 integer and 6 floating-point benchmarks. (c) **Cache Compression Analysis Across Different Levels:** 85%, 95%, and 100% cache block residency compression potential for different cache levels. (d) **Cache Compression and Cache Size:** With increasing cache size, compression ratio deteriorates somewhat. (e) **Cache Compression and Block Size:** With increasing block size, compression ratio improves. Cache associativity has negligible impact on compression ratio. (f) **Compression Ratio in Communication Components:** Compression ratios for zeroth and first order behavior of various buses at different levels of the memory system hierarchy. (g) **Original, XOR, and Offset Address Trace Compression:** Compression ratios for original, XOR, and offset address traces for various address buses.

Cache Type	Compression Method	Access Time		Total Power		Area			
		(ns)	(% redn.)	(nJ)	(% redn.)	Tag (cm ²)	Tag (% redn.)	Data (cm ²)	Data (% redn.)
L1	Uncompressed	1.2790	—	1.6889	—	0.00112	—	0.01163	—
L2	Uncompressed	1.7363	—	3.0679	—	0.00514	—	0.12910	—
L1 I-cache	Zeroth-order	1.2357	3.385	1.5845	6.185	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 1: 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.

tag and data arrays, we used the CACTI 3.0 model [19] for a 0.18 micron SRAM cache implementation. Table 1 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 fewer blocks). In some cases, however, the size of the compressed cache was too small (due to the compression ra-

tio 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 improved by about 32% (27%), power consumption by about 51% (41%), tag area by about 71% (86%), and data area by about 75% (89%) on the average for L1 (L2) levels w.r.t. normal uncompressed caches with the same effective capacity.

5.6 Compression and Transition Ratio Across Individual Buses

Zeroth order and first order redundancies in all buses: Figure 1(f) shows compression and transition ratio results for demultiplexed buses at all three levels. We observe that the R_{H_0} and R_{H_1} 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 address is most compressible. Here again, we find that the power savings for address, instruction, and data buses follows the ordering described in Section 5.1.

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. As seen from Figure 1(g), since instruction addresses occur at some very frequent offsets (typically an instruction word), the zero information and zeroth 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.

5.7 Compression Ratio and Bit Fields

In this experiment, we consider four consecutive bit fields (from high to low order: F3, F2, F1, F0) that are each one-fourth the size of a word for 32-bit addresses and 64-bit data. For 20-bit I-cache tag we consider three fields (F2, F1, F0) of widths 4, 8, and 8 bits respectively. 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). The symbol size used to do compression analysis corresponds to this bit field size. We generated individual bit-field traces and then analyzed each trace by doing zeroth and first order Markov analysis. According to the results shown in Figure 2(a), compression ratio varies across these four fields and the variation differs for each type of traffic. We observe that compressibility improves from low order to high order bit fields, except somewhat in the case of instruction bus traffic. From most to least compressible, the ordering is tag field, address, data, and then instruction.

5.8 Compression Ratio and Bit-Field Groupings

We also investigated how compression ratio varies depending upon grouping of bits fields for compression. From our experiments and results in Figure 2(b), we found that 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, the best compression ratio on all types of traces can be obtained.

5.9 Compression Ratio and Power Savings and Application Class

Since all of our previous results are averaged over all benchmarks, we also wanted to compare the behavior of integer and floating-point benchmarks. The results of this experiment are shown in Figure 2(c). As seen earlier in Sec. 5.2, data in INT registers are more compressible than data in FP registers. On the contrary, 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 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.

5.10 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 2.3. 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. Here, we show separate results for INT and FP. The general case is similar to the application-specific case except that there is only one application class that includes all applications/benchmarks. We observe from results in Fig. 2(d) that with the degree of specialization decreasing, the compression ratio deteriorates. 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.

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

Some experiments above demonstrate that power saving could 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 above three cases and as shown in Fig. 2(e) we found that by using compression and encoding together, we could achieve the best power saving. In fact, the simulated number of transitions in the third method is 12%–39% less than just using encoding and 6%–14% less than just using compression. So if we apply a scheme which combines both compression and encoding, we could

