

# MEMORY SYSTEM COMPRESSION AND ITS BENEFITS

Jiangjiang Liu<sup>†§</sup>, Nihar R. Mahapatra<sup>†§</sup>, Krishnan Sundaresan<sup>‡§</sup>, Srinivas Dangeti<sup>\*</sup>, and Balakrishna V. Venkatrao<sup>\*</sup>

<sup>†</sup> Dept. of Comp. Sci. & Eng.    <sup>‡</sup> Dept. of Electrical Eng.

<sup>§</sup> University at Buffalo, The State University of New York  
Buffalo, NY 14260-2000, USA

<sup>\*</sup> Processor Products Group  
Sun Microsystems, Inc.  
Palo Alto, CA 94303-4900, USA

Email:    <sup>§</sup>{jliu3, mahapatr, ks48}@cse.buffalo.edu  
          <sup>\*</sup>{srinivas.dangeti, balakrishna.venkatrao}@eng.sun.com

*Abstract*— This paper presents an overall analysis of 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. Analysis of traces obtained with Sun Microsystems' Shade simulator simulating SPARC executables of nine integer and six floating-point programs in the SPEC CPU2000 benchmark suite yield impressive results. Well-designed compression schemes may provide benefits in performance, power, and cost that far outweigh their overheads constituting extra time, logic, and power for compression and decompression. This will be more so in the future since the speed, size, and power consumption of logic (which will be used to perform compression/decompression) are improving and are projected to improve at a much higher rate compared to those of interconnect (which will be used to communicate the information), both on-chip and off-chip.

*Keywords*— Address compression, bandwidth, cache, cost-effective design, data compression, entropy, instruction compression, latency, lossless compression, low power, Markov models, memory, register file, storage, traffic.

## I. INTRODUCTION

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.

Increasing levels of device integration, rising clock frequencies, and die area increases combined with greater degrees of parallelism exploited at the bit-, instruction-, thread-, and processor-levels have resulted in an exponential trend for computation system performance enhancement. Due to such dramatic increases in processor system performance, there is an enormous pressure on the memory system to store increasing amounts of information (instructions and data) and communicate this information (addresses, instructions, and data) at a high enough bandwidth and low enough latency to avoid performance bottlenecks.

This research was supported by US National Science Foundation Grant # ACI-0102830. More details on the research reported here appear in previous papers [16] and [17].

To address the above problem, designers have continued to increase the number of I/O buffers, pads, and pins, width of buses, number of registers, number and sizes of caches, and the size of main memory, in addition to improving their design. However, while the number of transistors on a chip can essentially grow as the volume of the die, the number of I/O buffers, pads, and pins and off-chip bus width can grow only as the surface area of the chip. Further, there are more stringent constraints on the clock speed at which external pins can be driven compared to on-chip circuitry. Since interconnect size does not scale as well as on-chip logic size, on- and off-chip buses, especially the latter, have relatively higher capacitances and delays compared to on-chip logic. Also, DRAM bandwidth and latency are improving at a slower rate. All of this contributes to a growing computation-memory system performance gap [7].

The fraction of the processor chip devoted to storage (registers, caches) and communication components (I/O buffers and pads, on-chip buses) is increasing and so also is the number and size of off-chip storage (off-chip caches, main memory) and communication (pins, off-chip buses) components [7]. As noted above, storage components have increased in number and size in order to reduce performance bottlenecks and communication components occupy a greater fraction of the chip area because interconnect size scales relatively poorly compared to logic size. Moreover, in deep-submicron regime, not only individual wire capacitances contribute to power consumption, but more so do interwire capacitances between adjacent 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 of storage components), power consumption, and cost. For our study, we use 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 investigate 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 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 and data buses. Trace analysis performed with Sun Microsystems' Shade simulator simulating SPARC executables of nine integer and six floating-point programs in the SPEC CPU2000 benchmark suite yield impressive results.

The organization of the remainder of the paper is as follows. Section II discusses the advantages of a compressed memory system architecture and its feasibility. Section III provides a brief overview of previous work related to the use of compression to reduce embedded code and memory size and also bus encoding to reduce bus power. Section IV describes the simulation environment, analysis tools, and methods we used in our study. Section V presents detailed results of our analysis. Finally, we conclude in Section VI.

## II. THE CASE FOR MEMORY SYSTEM COMPRESSION

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.

### A. 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, 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. 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).

### B. Benefits and feasibility of compression

By employing compression, the same effective bandwidth can be obtained with a narrower bus and the same effective capacity can be realized using a smaller storage

component. Reduction in the size of these resources will lead to cost and power savings. The reduction in the size of a storage component can also potentially reduce its latency. If the size of these resources is kept unchanged, then compression can provide more performance (bandwidth, storage capacity) for a similar cost.

On the other hand, the use of compression also entails overheads in terms of extra logic, power, and latency for compression and decompression. However, since the size, speed, and power consumption of logic (which will be used to perform compression/decompression) is improving at a faster rate than that of interconnect (which will be used to communicate the information), well-designed compression schemes may provide significant overall benefits.

### C. Architecture of compressed memory systems

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.

## III. RELATED WORK

Previous work in memory system compression has been done both in analyzing compressibility [2], [23], [10], [9] and in the development of specific compression schemes for the memory system. These include schemes for address traffic compression [18] and extension of these schemes to instruction and data traffic compression [4]. Work has been done in code compression [25], [15], [12], [13] and compressed instruction set design [8], [1] for embedded systems. Some research has also been attempted in cache [26], [22], [11] and main memory [21], [6] compression. Finally, various bus encoding schemes have been developed to reduce unnecessary transitions and hence power consumption in buses [3].

## IV. SIMULATION METHODOLOGY

In this section, we describe our target system and the simulation environment including the methodology we used to collect information traces for our analysis and define the metrics we use to quantify the redundancy of the information.

### A. Target system and 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 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. We used a modified version of the cachesim5 cache analyzer in SHADE5 [5] running on a uniprocessor SPARC-V9 platform to collect the real-time traffic for 9 integer and 6 floating point benchmarks in the SPEC CPU2000 suite. We ran the benchmark programs using the reference input sets provided with the SPEC2000 suite, but to limit the execution times of our simulations, we used a methodology similar to the one described by Skadron et al. [19] and simulated all the instructions before the representative segment (warm up window) and traced a representative segment or sampling window of 2 million instructions.

### B. Analysis tools

We used the following measures to quantify the amount of information contained in our trace samples.

#### B.1 Entropy measures

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:

$$H_0 = -\sum_{i=1}^M [p(s_i) \cdot \log(p(s_i))].$$

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:

$$H_1 = -\sum_{i=1}^M [p(s_i) \cdot \sum_{j=1}^M [p(s_j|s_i) \cdot \log(p(s_j|s_i))]].$$

The symbols that we consider while measuring the entropy of any trace correspond to aligned words in the trace (i.e., 32-bit words for addresses and instructions and 64-bit words for data).

#### B.2 Compression ratios

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:

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

### B.3 Transition ratios

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 define *average transition ratio* for compressed traces as:

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

### C. Analysis methods

We calculate the various compression and transition ratio measures described in Section IV-B for address, instruction, and data traffic traces in demultiplexed buses at various levels of the memory system to estimate performance, power, and cost improvements due to compression. For main memory, we calculate compression ratio for the text segment of statically-linked executable code. For cache compression, to keep the analysis simple, we consider only instruction caches. We monitor every block that is both loaded and replaced within the sampling window of the simulation and keep a record of cache residency time (CRT) and block address for each such block. After simulation, we list the blocks in decreasing order of residency times and sum the cache residency times of all blocks to get the total CRT (TCRT). To accurately capture the occurrence of as many unique blocks as possible and to avoid a huge trace file, we consider only blocks upto 95% or 85% TCRT and scale down the number of occurrences of such blocks by that of the last block selected. From the addresses and CRTs of the blocks available in the above analysis, we are able to create similar trace files for analyzing tag field compression. In a similar fashion, for register compression, we consider only register values that are loaded and replaced during the sampling window, although in this case we used the entire trace file for our experiments instead of using only a subset as in the cache analysis above. We calculated only the zeroth order Markov measure for registers because there is not much first order dependency to explore.

## V. RESULTS

Using the simulation setup and procedure described in Section IV, we ran a number of simulations with the following default settings: a memory-component specific architecture, demultiplexed buses, word size for entropy analysis of 32 bits for addresses and instructions, 64 bits for data, and 20 bits for tag field. We summarize results by averaging over all 15 (9 INT and 6 FP) benchmarks and we calculate the average compression and transition ratios as mentioned in Section IV-B.

### A. Overall memory system analysis

We investigated how compression ratio and power consumption vary across memory system components,

namely, register file, cache, main memory, address bus, instruction bus, and data bus. The compression ratio is indicative of the extent to which performance enhancement or cost savings can be realized. Figure 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 tag field, registers (considering its  $R_{H_0}$  value since its  $R_{H_1}$  value is not computed), L1 I-cache data field, and main memory, which is to be expected. 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,  $R_{SAMC}$  decreases in the same order, suggesting that more compression (i.e., fewer bits transmitted) leads to better power savings.

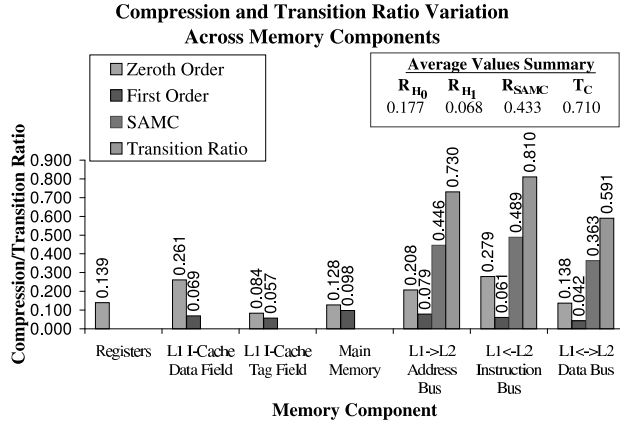


Fig. 1. **Overall Memory System Analysis:** Compression and transition ratio variation across memory system components.

### B. Register and cache 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 2 shows the compression ratio for each register. The average integer register compression ratio is 0.122 and floating-point register compression ratio is 0.156. 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  $R_{H_0}$  value 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.

From Figure 3, we find 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.

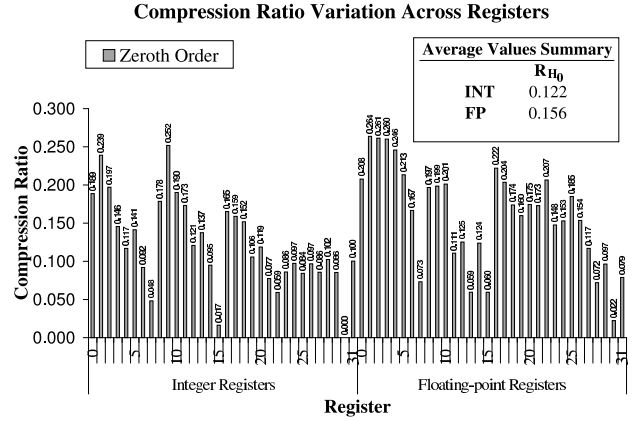


Fig. 2. **Storage Component Analysis (Registers):** Average register compression analysis summary for 32 integer registers and 32 floating-point registers across 6 integer and 6 floating-point benchmarks.

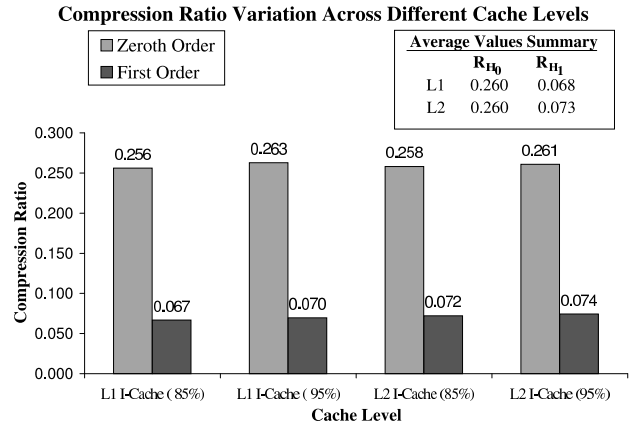


Fig. 3. **Storage Component Analysis (Caches):** 85% and 95% cache block residency compression potential for different cache levels. Compression for L1 cache is better than that for L2 cache.

### C. Compression ratio across individual buses

Figure 4 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 V-A.

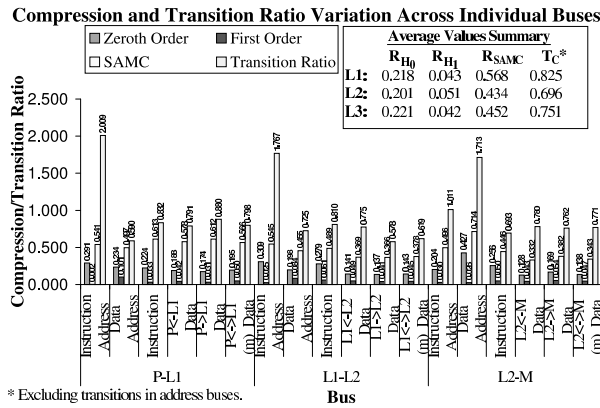


Fig. 4. **Compression and Transition Ratios in Communication Components:** Compression and transition ratios for various buses at different levels of the memory system hierarchy.

## VI. CONCLUSIONS

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) at various levels. Among storage components, we obtained a best-case average value of 0.084 (0.057) for zeroth (first) order Markov entropy. Among communication components, the best-case averages were 0.138 (0.042) for zeroth (first) order entropy. In future studies, we intend to look at other factors that may affect compressibility of information in the memory system such as cache parameters (cache size, block size, degree of associativity), bit-field groupings, and degree of specialization of the compression scheme. Using the results of this and any further study, our future work will be targeted at developing low overhead, scalable address, instruction, and data compression schemes that are suitable for various components of the memory system. The goal is to achieve appropriate cost-performance tradeoffs in accordance with the needs of various application and system requirements and as permitted by technology constraints.

## REFERENCES

- [1] Advanced RISC Machines. *An Introduction to Thumb*, March 1995. Available at: <http://www.arm.com>.
- [2] 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.
- [3] W.-C. Cheng and M. Pedram. Memory Bus Encoding for Low-power: A Tutorial. In *Proceedings of International Symposium on Quality of Electronics Design*, March 2001.
- [4] 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.
- [5] 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.

- [6] 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.
- [7] J.L. Hennessey and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1996.
- [8] K.D. Kissell. MIPS16: High-density MIPS for the Embedded Market. <http://www.mips.com/Documentation/MIPS16whitepaper.pdf>, 1997.
- [9] M. Kjelso, M. Gooch, and S. Jones. Empirical Study of Memory-data: Characteristics and Compressibility. *IEEE Proceedings on Computers and Digital Techniques*, 145(1):63–67, January 1998.
- [10] M. Kozuch and A. Wolfe. Compression of Embedded System Programs. In *Proceedings of International Conference on Computer Design*, 1994.
- [11] 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, 1999.
- [12] C. Lefurgy, P. Bird, I.C. Chen, and T. Mudge. Improving Code Density Using Compression Techniques. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, December 1997.
- [13] H. Lekatsas and W. Wolf. Code Compression for Embedded Systems. In *Proceedings of Annual ACM/IEEE Design Automation Conference*, pages 516–521, June 1998.
- [14] 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.
- [15] 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*, March 1995.
- [16] N.R. Mahapatra, J. Liu, K. Sundaresan, S. Dangeti, and B.V. Venkatrao. An Analysis of the Potential of Compression in Improving Memory System Performance, Power Consumption, and Cost. In *Proceedings of the Second Annual Workshop on Memory Performance Issues (WMPI 2002)*, May 2002.
- [17] N.R. Mahapatra, J. Liu, K. Sundaresan, S. Dangeti, and B.V. Venkatrao. The Performance Advantage of Applying Compression to the Memory System. In *Proceedings of the ACM-SIGPLAN Workshop on Memory System Performance (MSP 2002)*, June 2002.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] 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*, December 2000.
- [23] 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*, January 1994.
- [24] Wayne Wolf. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann Publishers Inc., 2001.
- [25] 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, 1992.
- [26] J. Yang, Y. Zhang, and R. Gupta. Frequent Value Compression in Data Caches. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*, November 2000.