A New Algorithm for Energy-Driven Data Compression in VLIW Embedded Processors

Alberto Macii[#] Enrico Macii[#] Fabrizio Crudo[*] Roberto Zafalon[*]
[#]Politecnico di Torino, Torino - Italy
[*]STMicroelectronics Agrate Brianza (Milano) - Italy

Abstract

This paper presents a new algorithm for on-the-fly data compression in high performance VLIW processors. The algorithm aggressively targets energy minimization of some of the dominant factors in the SoC energy budget (i.e., main memory access and high throughput global bus). Based on a differential technique, both the new algorithm and the HW compression unit have been developed to efficiently manage data compression and decompression into a high performance industrial processor architecture, under strict real time constraints (Lx-ST200: A 4-issue, 6-stages pipelined VLIW processor with on-chip D and I-Cache). The original Data-Cache line is compressed before write-back to main memory and, then, decompressed whenever Cache refill takes place. An extensive experimental strategy has been developed for the specific validation of the target Lx processor. In order to allow public comparison, we also report the results obtained on a MIPS pipelined RISC processor simulated with SimpleScalar. The two platforms have been benchmarked over Ptolemy and MediaBench programs. Energy savings provided by the application of the proposed technique range from 10% to 22% on the Lx-ST200 platform and from 11% to 14% on the MIPS platform.

Keywords: Data compression algorithms, system-level energy optimization, VLIW embedded processors.

1 Introduction

The embedded processor market is rapidly growing and the complexity of embedded applications is increasing even faster. Recently, VLIW architectures have been proposed as an attractive alternative to more conventional CPUs, to balance performance with hardware complexity and scalability [1]. The performance/complexity trade-off is made possible by a sophisticated Instruction Level Parallelism (ILP) compiler infrastructure that aims at scheduling high performance parallel code at compile time rather than at run time.

In this paper, we focus our attention to a specific aspect of the design of embedded system architectures, namely, the speed/energy optimization of the processor-to-memory communication path. In particular, we consider the problem of reducing the amount of memory traffic, and consequently the dissipated energy, when data is exchanged between the processor and the storage sub-system (i.e., L1 cache and main memory) in a high performance VLIW processor architecture, under strict real time constraints. In particular, we pursue our goal by relying on compression of the information being transferred between Cache and main memory.

Until recently, information compression techniques were only targeted towards code size reduction [7][10]. A few of them were adopted with remarkably good results and industrial exploitation (e.g., CodePack [14] and ARM Thumb [5]), but energy optimization was never considered as primary optimization objective, and thus acquired just as a by-product of code size minimization.

Starting with the work presented in [8][9], some selective instruction compression techniques for low-power successfully appeared, bringing to evidence that a key trade-off between code size and total execution energy does exist when dealing with advanced computing embedded platforms. The fundamental principles exploited in low-energy code compression can be further extended to the general case of low-energy data compression, provided that some significant adaptations are applied to the traditional, widespread data compression approaches. In fact, while achieving extremely good compression ratios, the traditional asymmetric algorithms [3][4] usually work on bounded, yet long, input streams and purposely shift the largest portion of the heavy computational effort to be run "off-line", once and for all, at compression time. This enables the use of a much faster and lighter "on-line" decompression step. On the receiving side, the decompression task needs to be performed anytime at the user's terminal, often under severe real-time requirement (e.g., MPEG2, MPEG4 and MP3 [4]).

In our case, an unbounded, although short, input stream of N bits (usually a single Cache line) must be compressed in a shorter string of bits K<N at any Cache

write-back and decompressed at any Cache refill. Both the compression and decompression steps have to be performed on the target embedded system, thus requiring extremely high efficiency (time and energy) on either side. As a matter of fact, it comes out that the traditional compression techniques are largely unsuitable for this scenario since they would provide good compression ratio only at the cost of huge, if not simply unfeasible, system overheads at compression time. On the opposite, our novel differential scheme is extremely effective just for the compression of the small data-lot represented by an individual data Cache-line.

Our major goal is to apply innovative data compression techniques on today's highly integrated VLIW embedded systems, in order to enable further energy savings with no system degradation in terms of both performance and area.

The advantage of using data compression on the Cache-to-Memory path is multiple. In fact:

1. Storing a data-lot in compressed format requires a smaller number of energy-expensive memory accesses on both read and write mode.
2. The average switching activity across the CPU-to-memory communication bus is reduced thanks to a smarter and less-demanding bandwidth allocation.
3. Including the HW assisted compression-decompression unit (CDU) between Data-Cache and main memory allows us to save on the total energy budget while easily buffering and compensating the CDU's extra delay (the CDU is a combinational HW block, and its critical path fits into one processor clock cycle only).

Our data compression algorithm proves to be *simple* to work on a regular Cache line; *differential* to smoothly deal with the very limited data predictability of computing systems (although embedded); *with no overhead*, since embedded systems are often subject to very tight performance and max latency constraints; *effective*, as it provides energy savings ranging from 10% to 22%, depending on the benchmark and the architectural platform to which it is applied.

The remainder of this paper is organized as follows. Previous work on the subject of low-energy data compression is briefly reviewed in Section 2, while the target VLIW processor architecture (Lx-ST200) for which the presented new algorithm has been developed and qualified is summarized in Section 3. Section 4 presents the new **Diff-Lx** differential compression scheme, together with an efficient HW implementation of the compression-decompression unit. Section 5 is devoted to experimental results collected through extensive experimentation on the Lx-ST200 VLIW core. Results from the MIPS RISC processor (SimpleScalar) are provided as well, to prove the generality of the proposed compression algorithm. The paper closes with some final remarks in Section 6.

2 Previous Work

The idea of applying data compression to minimize computational energy derives from code compression. In particular, successful attempts were made to limit energy consumption in embedded systems by reducing memory energy requirements through the compression of I-Cache [8][12] and main program memory [7][9].
The issue of designing fast and inexpensive HW compression units for processor's machine-code has been extensively investigated in [2] and found its way in some industrial products (e.g., [5][14]). A dictionary based instruction compression technique was first proposed in [10]. Further evolutions of code compression algorithms for low-power have been recently developed in [12].

3 Target VLIW Processor: Lx-ST200

In this section we will briefly describe the architecture of our target embedded processor.

Lx is a scalable and customizable VLIW processor architecture, jointly developed by STMicroelectronics and HP-Labs to aggressively target multimedia and signal processing embedded applications [1]. The basic Lx architecture is a 128 bit wide, 4-issues VLIW core, featuring four 32-bit integer ALUs, two 16x32 multipliers, one load/store unit, a branch unit and a 64x32-bit general purpose, multiple ports Register-File (8 Read ports, 4 Write ports). Lx has an in-order 6-stage pipeline and features a simple integer RISC ISA. It also includes an embedded 32 KB direct mapped I-Cache and a 4-ways set associative 32 KB D-Cache with FIFO replacement policy.

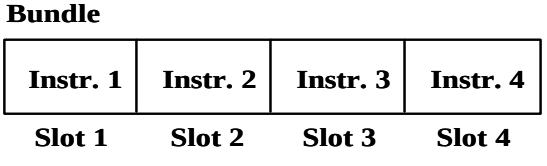

Figure 1. Bundle of 4 parallel operations in Lx-VLIW

For the first generation, realized in 0.25µm HCMOS technology at nominal Vdd=2.5V, the scalable Lx processor runs at 200 MHz (typ) and is planned to span from one to four clusters (i.e., from 4 to 16 instructions issued per cycle). The instruction level parallelism (ILP) is achieved through the execution of 4 explicitly parallel operations (also called syllables) at each cycle; these operations are statically scheduled to constitute the 128-bit Very Long Instruction Word (also called instruction or bundle). Figure 1 shows the bundle of 4 operations.

The Lx's tool-chain includes an aggressive ILP compiler [1], GNU tools and libraries as well as a fast and efficient ISS, which has been used in our experiments by assuming an external main memory of 8MB SDRAM (i.e.: AS4LC4M16SO by Alliance Semiconductors). The length of Lx's basic data word is W=32 bits while the D-Cache line is 8 words wide (i.e.:

32 bytes). Memory blocks used in the D-Cache enable data write of 64-bit blocks and data read of 32-bit blocks at a time (see Figure 2). This structure has been chosen in Lx to increase block replacement speed by exploiting the burst access capability of the main memory.

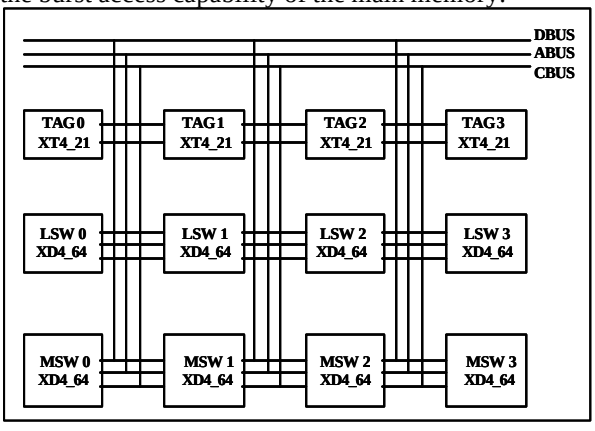

Figure 2. Lx D-Cache (32KB, 4-ways set associative)

4 Hardware-Assisted Differential Data Compression Algorithm

An efficient hardware compression and decompression unit (CDU) for data read and write to the main memory was recently presented in [11]. As we mentioned earlier, the CDU is placed between Data-Cache and the main memory in order to better attack two of the dominant factors in SoC energy budget (i.e., high throughput global bus and main memory access rate), while easily buffering and compensating the CDU extra delay. The CDU proposed in [11], whose high-level block diagram is shown in Figure 4, offers a versatile and customizable template that can accommodate different compression algorithms, which can thus be designed according to the constraints and application domain posed by the processor architectural template of choice. For example, in [11] the MIPS RISC processor was considered as the target architecture, and several compression algorithms were developed with such architecture in mind.

The CDU is a combinational block of minimal hardware complexity, and its critical path fits snugly into one processor's clock cycle only. Hence, compression is performed in one extra clock cycle on Cache write-back, while decompression is on Cache refill. With reference to Figure 4, the Line Compressor (LC) and the Line Decompressor (LD) have been synthesized as multi-level combinational logic circuits starting from Verilog RTL descriptions. They are small in size (around 3000 equivalent gates) and tailored to the differential compression scheme we have devised. This justifies the choice of privileging a fast implementation of the CDU. Indeed, its energy contribution is still negligible with respect to the total system budget (see the energy breakdown of Figure 3). Finally, the Compressed Line Address Table (CLAT) is a content addressable memory (CAM), keeping the address of each Cache block currently written-back to main memory. It provides the way to know the actual format (compressed or not) of a line to be fetched from main memory whenever the Cache Controller issues a refill request.

As mentioned earlier, in this paper we focus on VLIW architectures, and we propose a new differential algorithm that best exploits the characteristics of operation execution in this kind of processors. We call the algorithm **Diff-Lx**; it features remarkably less bus traffic and total energy than those in [11], thanks to a minimum additional overhead in the compression field format.

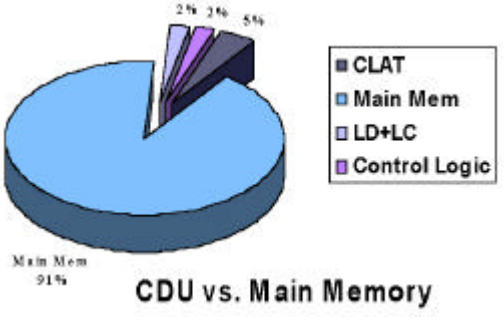

Figure 3. Energy breakdown for Lx-VLIW platform

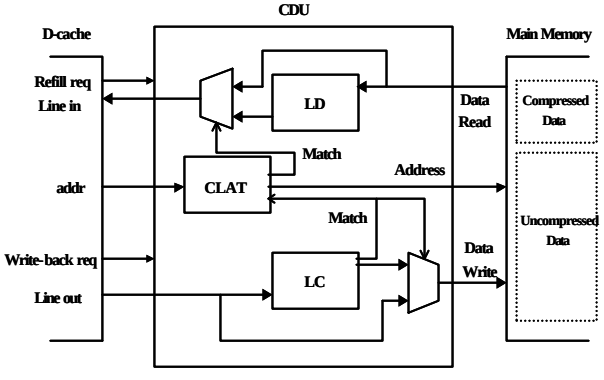

Figure 4. CDU: Compression Decompression Unit

The differential algorithm, which is described next, has been implemented and integrated into the Lx tool-chain, extending in this way the exploration space for next generation low-power embedded systems.

4.1 The Diff-Lx Algorithm

Let's assume that the main memory is byte addressable, the D-Cache line size is L bytes and the basic data word is W bits long. Compression and Decompression are performed on-the-fly in one clock cycle, seamlessly to the processor. The algorithm allows defining the minimum compressed line size S (S<L), as the threshold to discriminate whether or not the current Cache write-back will actually go to main memory in compressed format.

The basic idea behind the **Diff-Lx** differential scheme lies on realizing that for data words appearing in the

same Cache line, some of the bits are common across pairs of adjacent words, either from the LSB (Least Significant Bit) to the MSB (Most Significant Bit) or vice versa. As an example, if data can take values from a limited range, it may well occur that some of the most

significant bits among each pair of words in the Cache line are the same. As a consequence, it would then be possible to store a much shorter line, leveraging on bit's sharing. In particular, given the original Cache line (illustrated, for example, with four words in Figure 5), for each pair of data words W_i and W_j, we look for the maximum number of common bits $\mathbf{cnt_{ij}}$, by scanning the Cache line both from LSB to MSB and vice versa.

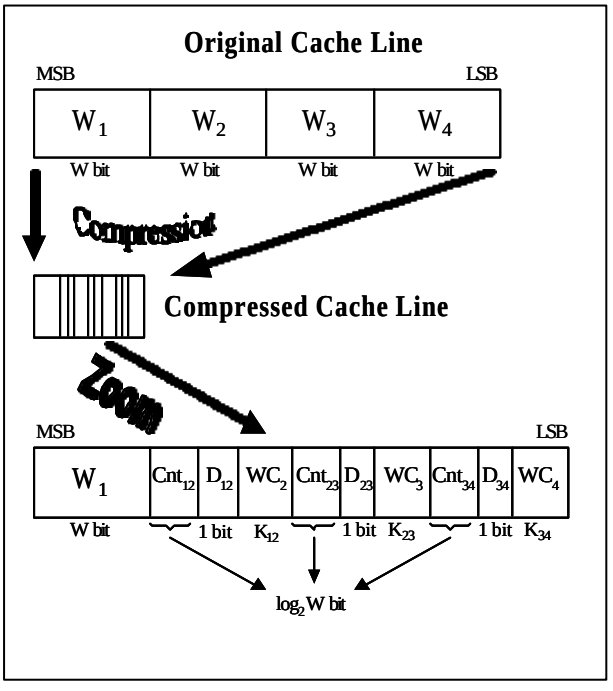

Figure 5. Example of a 4-words Cache Line, before and after the compression, respectively. Please notice that for Lx-VLIW the Cache line is 8 words wide.

In the target compressed line of Figure 5, the field $\mathbf{cnt_{ij}}$ (whose fixed length is $\log_2 W$) is the number of shared bits and $\mathbf{D_{ij}}$ the one bit direction flag. $\mathbf{D_{ij}}$ selects whether LSB or MSB is starting point for the decompression of the cnt_{ij} shared bits. Furthermore, we define $\mathbf{WC_i}$ as the remaining portion of the word $\mathbf{W_i}$, complementary to the shared bits $\mathbf{cnt_{ij}}$. We have $|\mathbf{WC_i}| = \mathbf{K_{ij}} = W - cnt_{ij}$ and $|\mathbf{C_{ij}}| = |cnt_{ij}| + |D_{ij}|$. Now, the Cache-line will be actually compressed on write-back, if and only if the following threshold condition is satisfied:

$$\mathbf{(1)} \quad |W_1| + |C_{12}| + |WC_2| + |C_{23}| + |WC_3| + |C_{34}| + |WC_4| \leq S*8$$

In essence, the wider the agreement between the word's pairs in the original Cache line (i.e., the value of cnt_{ij}), the higher the achievable compression ratio on write-back to main memory.

5 Experimental Set-Up and Results

5.1 Lx-ST200 VLIW Architecture

The proposed **Diff-Lx** algorithm has been applied to the Lx VLIW architecture running a set of benchmark programs taken from the MediaBench (MB) [15] and Ptolemy (PT) [16] suites.

Data and address buses are 32-bit wide, and the size of the D-Cache line is 8 words of 4 bytes each: total length S=32 (bytes). When condition **(1)** applies, Lx-VLIW can actually save up to 22 bytes in transfer and storage to main memory.

The detailed results we have achieved are presented in Table 1 (Mediabench benchmarks) and Table 2 (Ptolemy benchmarks), while Figure 6 and Figure 7 provide a pictorial summary. All the energy figures reported hereafter are based on circuit implementation with STMicroelectronics' 0.25µm HCMOS technology, operating at typical supply VDD=2.5V.

Regarding memory traffic (column *Memory Traffic*), tables report the number of words written/read to/from memory when no compression is used (column *Uncomp*), when the compression is performed (column *Comp*) and the percentage reduction of memory traffic when the CDU is used (column *%*).

Concerning the energy savings (column *Energy*), it is roughly proportional to the memory traffic because it is an indication of the memory access (for the most part) and of the CDU component. Also in this case, values obtained with or without compression and savings are reported.

Analyzing the total execution time for each SW benchmark (column *Execution Time*), we observe that the use of the compression engine does not degrade the processor's performance at all.

5.2 MIPS RISC Architecture

Diff-Lx compression has also been benchmarked on the MIPS RISC processor, using SimpleScalar as simulation environment. This was done with the purpose of proving the robustness of the algorithm.

SimpleScalar was configured to execute the PISA instruction set, derived from MIPS-IV 64 bit [13]. Such a configuration is very commonly used for public domain benchmarking because the MIPS architecture is a well-known single issue, pipelined RISC machine. The configuration of SimpleScalar's functional Cache simulator engine (i.e.: "sim-cache"), is the following: 2KB L1 I-Cache, 4KB D-Cache and 16MB SDRAM main memory.

Data and address buses are 32-bit wide, the data word is W=4 byte wide and the Cache line's length is 4 words (i.e.: 16 bytes). This means that when a Cache refill takes place, if the word is compressed and S=8 (bytes), we save 8 bytes of memory traffic, since the original Cache line's size is L=16 (bytes),

As per the Lx-VLIW, the adopted D-Cache policy was "FIFO replacement". The detailed results for MIPS

are presented in Table 3 and Table 4 again on both Mediabench and Ptolemy benchmarks.

As expected, results are slightly worse for the MIPS environment than for the Lx architecture, as the Lx-Diff algorithm was explicitly thought for the Cache organization of a VLIW core. Yet, the achieved energy savings are satisfactory, indicating that the proposed method can be applied to other architectures than just VLIW.

6 Conclusions

We have presented a new data compression technique for energy efficient high performance VLIW embedded processors. The approach addresses energy minimization of two of the dominant factors in SoCs energy budget (i.e., main memory access and high throughput global bus), by enabling an aggressive energy savings with limited overhead in chip area and no system performance degradation (i.e., execution speed). Based on differential techniques, both the new algorithm and the HW compression unit have been developed to efficiently manage data compression and decompression into a high performance industrial VLIW processor (i.e. STMicroelectronics' Lx-ST200), under strict real time constraints.

An extensive experimental test strategy has been applied for validation, addressing both the Lx-ST200 processor and the MIPS RISC core. The new differential algorithm, called **diff-Lx**, achieves energy savings ranging on average from 10% to 22% on the Lx-ST200 and from 11% to 14% on the MIPS.

7 References

[1] J. Fisher, P. Faraboschi, G. Brown, G. Desoli and F. Homewood, "Lx: a technology platform for customizable vliw embedded processing," in Proceedings of the International Symposium on Computer Architecture, June 2000, pp. 203–213.

[2] S. Bunton, G. Borriello, "Practical Dictionary Management for Hardware Data Compression, Comm. of the ACM, Vol. 35, No. 1, pp. 95-104, 1992.

[3] J. Ziv, A. Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Trans. on Information Theory, Vol. 23, No.3, pp. 337-343, 1977.

[4] Moving Picture Experts Group, MPEG-2 Generic coding of moving pictures and associated audio information, standard document ISO/IEC 13818.

[5] ARM Ltd., "An introduction to Thumb," Mar. 1995.

[6] B. Abali, et al., "Performance of Hardware Compressed Main Memory," HP Journal, pp. 73-81, 2001.

[7] Y. Yoshida, B.-Y. Song, H. Okuhata, T. Onoye, I. Shirakawa, "An Object Code Compression Approach to Embedded Processors," ISLPED-97, pp. 265-268, 1997.

[8] L. Benini, A. Macii, A. Nannarelli, "Cached-Code Compression for Energy Minimization in Embedded Processors," ISLPED-01, pp. 322-327, 2001.

[9] L. Benini, A. Macii, E. Macii, M. Poncino, "Selective Instruction Compression for Memory Energy Reduction in Embedded Systems," ISLPED-99, pp. 206-211, 1999.

[10] C. Lefurgy, P. Bird, I. Chen , T. Mudge, "Improving Code Density Using Compression Techniques," Microarchitecture, 1997. Proceedings, pp. 194-203, 1997.

[11] L. Benini, D. Bruni, A. Macii, E. Macii, "Hardware-Assisted Data Compression for Energy Minimization in Systems with Embedded Processors," DATE-02, pp. 449-453, March 2002.

[12] H. Lekatsas, H. Wolf , "SAMC: A Code Compression Algorithm for Embedded Processors," Trans. On CAD 1999, pp. 1689-1701, Vol. 18, No. 12, Dec. 1999.

[13] D. Burger, T. Austin "The SimpleScalar ToolSet, Version 2.0," Tech. Rep. UCB/ERL No. 1342, Univ. of Wisconsin-Madison, Dept. of CS, 1997.

[14] IBM, "Codepack PowerPC Code Compression Utility," User's Manual Version 4.1.

[15] C. Lee, M. Potkonjak, W. H. Mangione Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," 30th Annual IEEE/ACM International Symposium on Microarchitecture, 1997.

[16] J. Davis II, et al., "Overview of the Ptolemy Project," Tech. Rep. UCB/ERL No. M99/37, Univ. of California, Dept. of EECS, 1999.

Benchmark	Memory Traffic			Energy [nj]			Execution Time		
	Uncomp	Comp	%	Uncomp	Comp	%	Uncomp	Comp	%
JPEGdec	3,27E+06	2,54E+06	22,4	1,09E+08	8,44E+07	22,3	5,29E+09	5,25E+09	0,8
MPEG2dec	3,18E+05	2,98E+05	6,2	1,06E+07	9,90E+06	6,2	3,70E+07	3,67E+07	0,7
Edge_detect	3,60E+05	2,93E+05	18,7	1,20E+07	9,76E+06	18,5	1,78E+07	1,76E+07	1,2
Opendvx	2,63E+07	1,90E+07	27,7	8,75E+08	6,35E+08	27,4	3,76E+09	3,99E+09	-6,1
MatMult	1,19E+06	7,77E+05	34,7	3,97E+07	2,60E+07	34,5	2,88E+06	2,87E+06	0,3
Average			21,9			21,8			-0,6
Dev Std			10,7			10,6			3,1

Table 1. Lx Results on Mediabench benchmark.

Benchmark	Memory Traffic			Energy [nj]			Execution Time		
	Uncomp	Comp	%	Uncomp	Comp	%	Uncomp	Comp	%
Dft	1,56E+05	1,47E+05	5,4	5,14E+06	4,87E+06	5,3	3,30E+08	3,29E+08	0,4
DTMFCodec	6,99E+04	6,95E+04	0,5	2,31E+06	2,30E+06	0,3	1,51E+08	1,52E+08	-0,2
Filterbank	2,35E+05	1,82E+05	22,7	7,79E+06	6,04E+06	22,5	2,30E+08	2,26E+08	1,7
Average			9,5			9,4			0,8
Dev Std			11,6			11,7			1,4

Table 2. Lx Results on Ptolemy benchmark

Benchmark	Memory Traffic			Energy[nj]			Execution Time		
	Uncompr.	Compr.	%	Uncompr.	Compr.	%	Uncompr.	Compr.	%
JPEGdec	1,89E+06	1,75E+06	7,3	1,56E+07	1,38E+07	11,9	3,05E+06	3,01E+06	1,1
MPEG2dec	7,19E+06	7,10E+06	1,3	5,93E+07	5,87E+07	1,1	1,10E+07	1,12E+07	-2,0
Edge_detect	1,49E+07	1,54E+07	-3,6	1,23E+08	1,16E+08	5,3	5,77E+07	5,92E+07	-2,5
MatMult	1,31E+06	1,05E+06	20,0	1,08E+07	6,38E+06	40,8	3,28E+06	3,13E+06	4,4
Opendvx	1,90E+07	1,72E+07	9,8	1,57E+07	1,37E+07	12,5	1,30E+07	1,29E+07	0,7
Average			6,9			14,3			0,3
Dev Std			9,0			15,5			2,8

Table 3. Simple Scalar Results on MediaBench benchmark

Benchmark	Memory Traffic			Energy[nj]			Execution Time		
	Uncompr.	Compr.	%	Uncompr.	Compr.	%	Uncompr.	Compr.	%
Dft	6,40E+05	5,87E+05	8,2	5,28E+06	4,83E+06	8,5	3,39E+06	3,37E+06	0,6
DTMFCodec	2,39E+05	2,17E+05	9,2	1,97E+06	1,71E+06	13,3	7,23E+06	7,22E+06	0,1
Filterbank	1,32E+07	1,24E+07	6,1	1,09E+08	9,74E+07	10,3	5,50E+07	5,48E+07	0,3
Average			7,8			10,7			0,3
Dev Std			1,6			2,4			0,3

Table 4. Simple Scalar Results on Ptolemy benchmark

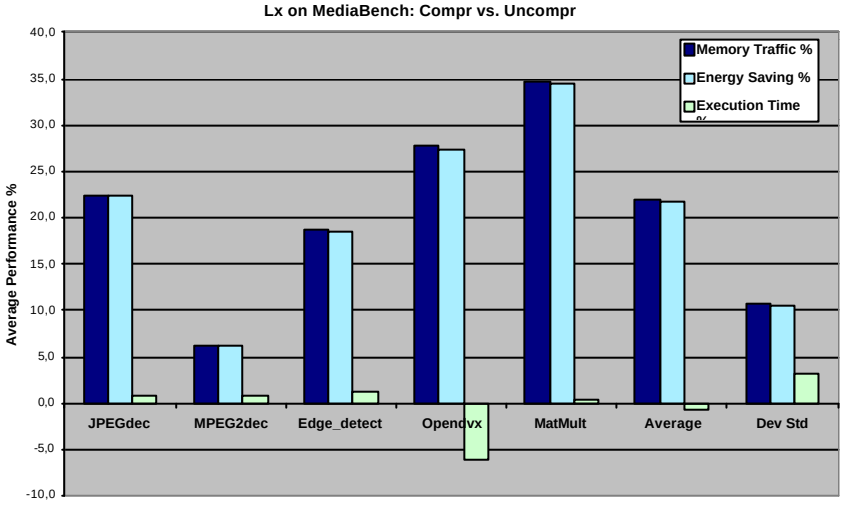

Figure 6.Lx Average Performance on MB Benchmark

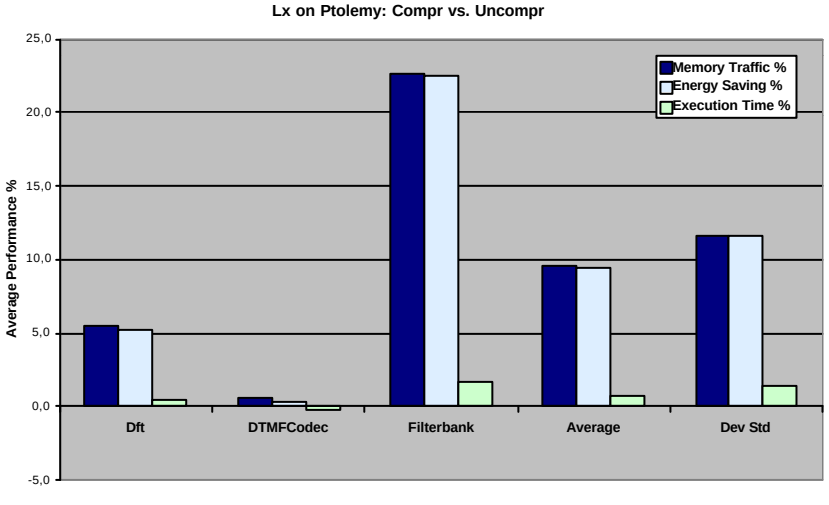

Figure 7.Lx Average Performance on Ptolemy Benchmark