On Power and Performance Tradeoff of L2 Cache Compression

Chandrika Jena, Tim Mason and Tom Chen

Dept. of ECE, Colorado State University, Fort Collins, CO, USA

Abstract—This paper presents the power-performance trade off of three different cache compression algorithms. Cache compression improves performance, since the compressed data increases the effective cache capacity by reducing the cache misses. The unused memory cells can be put into sleep mode to save static power. The increased performance and saved power due to cache compression must be more than the delay and power consumption added due to CODEC(Compressor and DECompressor) block respectively. Among the studied algorithms, power-delay characteristic of Frequent Pattern compression(FPC) is found to be the most suitable for cache compression.

I. INTRODUCTION

In recent years, a fair amount of research has been done into the idea of adding data compression to the memory path of microprocessors[1]. Typically, the use of compression is proposed as a way to store more information in a given amount of memory. Some researchers have proposed increasing the aggregate performance of a microprocessor by using compression to improve the likelihood of a cache hit[2]. Due to addition of CODEC block between the processor and L2 cache, the delay of the critical path increases. For the cache compression to be useful the performance improvement due to increase in cache size must not be offset by the additional delay of the CODEC block.

Another aspect of cache compression is to reduce the overall static power by putting the unused memory cell in sleep [3], [1], [4] mode. In CMOS design when gate is not transitioning, the static power is consumed due to leakage current. With decrease of transistor size [5], leakage current is increasing [6], [7]. To reduce the static power the path between VDD and GND should be disconnected. This can be achieved by adding sleep transistors [8],[9],[10]. Again, power saved by turning off the unused memory cell must not be offset by the power consumed by the CODEC block.

Due to the two reasons described above, it is important to study the power-delay trade off of the cache compression designs. The goal of this paper is to compare the power vs performance characteristics of three different algorithms: Frequent Pattern compression(FPC) [11], Run Length encoding (RLE), Diff_Lx[12].

The remainder of the paper is organized as follows. In Section II cache compression algorithms are reviewed, Section III presents the comparison of all the compression algorithms. Circuit implementation of these algorithms are given in Section IV. Section V discusses the simulation results. Conclusions are presented in Section VI.

II. REVIEW OF THE COMPRESSION ALGORITHMS

A. Modified Frequent Pattern Compression

The original Frequent Pattern Compression (FPC) [11] is specifically designed for the compression of data cache contents. It is based on the observation that some data patterns are frequent and also compressible to a fewer number of bits. For example, many small-value integers can be stored in 4, 8 or 16 bits, but are normally stored in a full 32-bit word. These frequent patterns can be specially treated to store them in a compact form. By encoding the frequent patterns, effective cache size can be increased. In FPC compression/decompression is done on the entire cache line. Each

cache line is divided into 32-bit words (e.g., 16 words for a 64-byte line). Each 32-bit word is encoded as a 3-bit prefix plus data.

Table I shows the pattern that can be encoded using FPC algorithm, and the amount of compression attainable by each compression type. These patterns are selected based on their high frequency in many of our integer and commercial benchmarks. As shown in the Table I; if data does not match one of the first 6 compressible patterns, then the original data is stored with three bit prefix. So it is possible that for the algorithm to expand rather than compress. To overcome this problem, the algorithm was modified. One Bit is added to each cache line to indicate whether or not the cache line is compressed. The added bit is called line bit. In this way, if a given cache line ends up with a negative compression ratio, the penalty is reduced to only one added bit out of the 8192 bits of a cache line(For this experiment 8191 bit cache line is used for all simulation).

Idx	Pattern Encoded	Mantissa Size	Ratio %
000	Zero Run	3 bits (runs up to 8 zeros)	6:32-6:256 (81 %-98 %)
001	4-bit sign-extended	4 bits	7:32 (78 %)
010	One byte sign-extended	8 bits	11:32 (66 %)
011	halfword sign-extended	16 bits	19:32 (41 %)
100	halfword padded with a zero halfword	Nonzero halfword (16 bits)	19:32 (41 %)
101	Two halfwords, each a byte sign-extended	Two bytes (16 bits)	19:32 (41 %)
110	word consisting of repeated bytes	8 bits	11:32 (66 %)
111	Uncompressed word	Word (32 bits)	35:32 (-9 %)

TABLE I
FREQUENT PATTERN COMPRESSION DICTIONARY

B. Run Length Encoding Compression

In an effort to find more ways for improving the compression results, a visual inspection was performed on the data which was uncompressible. This visual inspection revealed many repeated patterns at the sub-word level which led to the Run Length Encoding (RLE) algorithm.

RLE simply looks for repeated bit patterns. For RLE algorithm, two key parameters to control the encoding are identified: atom size and maximum run length. The atom size refers to the bit count of the smallest data division. For example, if RLE is performed on the nibble level, then atom size is 4 bits. Similarly, byte level RLE has 8 bit atoms. The other key parameter is the Maximum Run Length (MRL). Encoding of RLE will require $\log_2(\text{MRL})$ bits per run, plus a "we have a run" bit, and the data atom that is repeated. An example of the encoding with 4 bit atom size and an MRL of 8 atoms is shown in Fig. 1. The first two data atoms in this example (green and yellow) were not compressible. The next several atoms (pink) were all identical and were encoded as a single 8 bit quantity:

The ideal combination of MRL and atom size depends on the input data. For this data, the ideal parameters are determined to be an atom size of 16 bits and a MRL of 32 atoms (length field is 5 bits).

Fig. 1. RLE example with 4-bit atom size and 3-bit encoded MRL

C. Diff-Lx Compression

Diff-Lx was developed by Macii, et. al. It was used for VLIW (Very Long Instruction Word) processors, but the authors reported good results even with smaller word size processors. Also, they report having implemented both the compression and decompression logic in purely combinational logic with a latency of only one clock period.

The algorithm encodes the difference from one word to the next. The basic idea behind the Diff-Lx differential scheme lies in realizing that for data words appearing in the same cache line, some of the bits are common across pairs of adjacent words, either from LSB to MSB or vice versa. The encoding scheme for Diff-Lx is shown in Fig. 2. The first word of a cache line is stored verbatim. Every following word is then stored as count field, a direction bit and a remaining bits field. The count field stores the number of bits in the current word which are the same as the previous word, and the direction bit stores the direction of similarity. The remaining bits field (WCx) stores the dissimilar bits.

Fig. 2. Encoding scheme of Diff-Lx

III. COMPARISON OF COMPRESSION PERFORMANCE OF ALGORITHMS

Each algorithm (FPC, RLE, Diff-Lx) was simulated to determine the compression ratio using various types of input data. In order to quantify the compression results for comparison purposes, a Compression Ratio (CR) was calculated as.

$$\text{Compression Ratio} = 100\% - \text{Bits(out)}/\text{Bits(in)}$$

Using this scale, CR increases as the compressed data footprint decreases. A negative CR indicates that the compressed data is larger than the original (uncompressed) data. The goal is to find the algorithm which gives the highest average CR without worrying about excessively negative results in the worst case.

To evaluate the performance of each compression algorithm, a set of data files was selected in an attempt to emulate data that might be found in the data cache of a modern microprocessor. The files,

Type	Description
Word Document	Microsoft Word™ document with embedded formatting
JPEG Image	JPEG compressed picture.
MatLAB File	MATformat data file from MATLAB.
Windows Executable	Executable program image compiled for Intel x86 architecture.
ANSI Document	Readable text, encoded at 8-bits per character.
Unicode Document	Readable text, encoded at 16-bits per character.
Excel Spreadsheet	Microsoft Excel™ spreadsheet
Windows Library	Library of dynamically linked subroutines, compiled for Intel x86 architecture.

TABLE II
DATA FILES USED FOR COMPRESSION ANALYSIS

summarized in Table II, were selected to be about 2 megabytes in size and the types were chosen represent common end-user applications:

Initial simulation results with original FPC [6] showed a disturbing tendency towards negative compression ratios. Pattern usage histograms showed that most of the data words were falling into the uncompressed bin. This problem of negative CR is overcome by the modified FPC explained in section II-A.

Fig. 3 shows the results of simulating the modified FPC algorithm. Note that the addition of the line bit has been very effective in limiting the degree of negative compression in the JPEG sample data. The average compression ratio was about 10%. Table III shows the CR of all the simulated algorithms.

Fig. 3. Simulation results

Algorithms	Worst	Best	Avarage	sigma
Modified FPC	-0.20%	40.04%	9.99%	14.50%
RLE	-1.78%	14.18%	1.80%	5.09%
Diff-Lx	-0.02%	37.99%	8.48%	13.93%

TABLE III
COMPARISON OF COMPRESSION SIMULATION RESULTS

Running the RLE simulation with atom size 16 and MRL 32 atoms on the set of sample files gave the results show in Fig. 3. Every type of input file showed substantially less compression than with FPC, and the average CR dropped to 1%. The simulation results for Diff-Lx are shown in Fig. 3. For most data files, the results are mediocre. Only the Windows Library and Unicode Document showed good results, pushing the average compression up to 8.5 %.

IV. HARWARE IMPLEMENTATION OF COMPRESSION SCHEME

The three selected compression algorithms were implemented in VHDL and simulated to verify their functionality. The algorithms were simulated with 8191 bits data lines to find the CR To accurately