Thesis for The Degree of Licentiate of Engineering

Reducing Memory Traffic with Approximate Compression

Albin Eldstål-Ahrens

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2020

Reducing Memory Traffic with Approximate Compression

Albin Eldstål-Ahrens

Advisor: Ioannis Sourdis, Prof. at Chalmers University of Technology
Co-Advisor: Pedro Trancoso, Prof. at Chalmers University of Technology
Examiner: Fredrik Dahlgren
Discussion Leader: Georgios I. Goumas, Asst. Prof. at the National Technical University of Athens

Technical Report No 215L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2020.

Abstract

Memory bandwidth is a critical resource in modern systems and has an increasing demand. The large number of on-chip cores and specialized accelerators improves the potential processing throughput but also calls for higher data rates. In addition, new emerging data-intensive applications further increase memory traffic. On the other hand, memory bandwidth is pin limited and power constrained and is therefore more difficult to scale. This thesis proposes lossy memory compression as a means to reduce data volumes by exploiting the ability of applications to tolerate approximations in parts of their datasets. A reduction of off-chip memory traffic leads to reduced memory latency, which in turn improves the performance and energy efficiency of the system. The first part of this thesis introduces Approximate Value Reconstruction (*AVR*), which combines a low-complexity *downsampling* compressor with an LLC design able to co-locate compressed and uncompressed data. Two separate thresholds are employed to limit the error introduced by approximation. For applications that tolerate aggressive approximation in large fractions of their data, AVR reduces memory traffic by up to 70%, execution time by up to 55%, and energy costs by up to 20% introducing up to 1.2% error to the application output. The second part of this thesis proposes Memory Squeeze (*MemSZ*), introducing a parallelized implementation of the more advanced Squeeze (*SZ*) compression method. Furthermore, MemSZ improves on the error limiting capability of AVR by keeping track of life-time accumulated error. An alternate memory compression architecture is also proposed, which utilizes 3D-stacked DRAM as a last-level cache. MemSZ improves execution time, energy and memory traffic over AVR by up to 15%, 9%, and 64%, respectively.

Acknowledgment

The work underlying this thesis would not have been possible without the kind support of my colleagues and friends.

I am very grateful to my advisor Yiannis, for his firm guidance and advice during the ups as well as motivation during the downs. My co-advisor Pedro, thank you for offering balance and lightening the mood. Thanks to my former co-advisor Sally A. McKee, for your invaluable advice on writing, presentation and teaching.

I owe a great deal to my fellow PhD students at CSE. Evangelos taught me everything I know about simulation, and created much of the infrastructure upon which this work is based. His dark sense of humor helps the rest of us feel more well-adjusted. Thank you Ahsen and Alirad, for all your help with the struggle of hardware design. Prajith and Stavros, who have been my guides to the business of being a PhD student. My neighbor Petros, for comedic relief and encouragement. Stefano, who reminds us all that there is also a life outside the 4th floor.

Miquel Pericas and Lars Norén, for their generous help in my eternal quest for processing power. This really wouldn't have been possible without you.

Rolf Snedsböl, the joy of the office. If you are ever allowed to retire, we will all miss you. The CSE administrative staff who let people like me focus on the things we understand, especially Monica Månhammar who deserves the credit of three regular people.

Finally, Lea, my loving partner in crime. We share the struggles of doctoral studies, and this would be infinitely more difficult without you.

Albin Eldstål-Ahrens
Göteborg, June 2020

This work is supported by the Swedish Research Council (contract number 2012-4924) under the ACE project.

List of Publications

Appended publications

This thesis is based on the following publications:

I Albin Eldstål-Damlin, Pedro Trancoso and Ioannis Sourdis
"AVR: Reducing Memory Traffic with Approximate Value Reconstruction"
Proceedings of the 48th International Conference on Parallel Processing (ICPP). 2019.

II Albin Eldstål-Ahrens and Ioannis Sourdis
"MemSZ: Squeezing Memory Traffic with Lossy Compression"
Under submission, ACM Transactions on Architecture and Code Optimization (TACO).

Contents

Abstract iii

Acknowledgement v

List of Publications vii

1 Introduction 1
- 1.1 Problem Statement . . . 2
- 1.2 Related Work . . . 3
  - 1.2.1 Memory Compression . . . 3
  - 1.2.2 Approximate Computing . . . 4
- 1.3 Thesis Objectives . . . 4
- 1.4 Thesis Contributions . . . 5
  - 1.4.1 Approximate Value Reconstruction . . . 7
  - 1.4.2 Memory Squeeze . . . 7
- 1.5 Thesis Outline . . . 8

2 AVR: Reducing Memory Traffic with Approximate Value Reconstruction 9
- 2.1 Introduction . . . 9
- 2.2 Related Work . . . 11
- 2.3 AVR Architecture . . . 12
  - 2.3.1 Memory Blocks . . . 13
  - 2.3.2 Metadata Table . . . 14
  - 2.3.3 Summarizing & Reconstruction . . . 15
  - 2.3.4 Last Level Cache . . . 17
  - 2.3.5 Memory operations . . . 21
- 2.4 Evaluation . . . 22
  - 2.4.1 Experimental setup . . . 23
  - 2.4.2 AVR hardware overhead . . . 24
  - 2.4.3 Experimental Results . . . 24
- 2.5 Conclusions . . . 29

3 MemSZ: Squeezing Memory Traffic with Lossy Compression 31
- 3.1 Introduction . . . 31
- 3.2 Background & Related Work . . . 33
  - 3.2.1 Related Work . . . 33
  - 3.2.2 Background . . . 35

3.2.2.1 AVR: Approximate Value Reconstruction . . . 35
3.2.2.2 SZ Compression . . . 38
3.3 MemSZ Architecture . . . 38
3.3.1 MemSZ parallel lossy compressor . . . 39
3.3.2 Error Limiter . . . 43
3.3.2.1 Metadata Table . . . 44
3.3.3 Last Level Cache . . . 44
3.3.3.1 SRAM LLC on the processor die . . . 45
3.3.3.2 3D-stacked DRAM LLC . . . 45
3.4 Evaluation . . . 46
3.4.1 Experimental Setup . . . 47
3.4.2 Hardware Overhead . . . 49
3.4.3 Experimental Results . . . 49
3.4.3.1 MemSZ with SRAM LLC . . . 50
3.4.3.2 Evaluation of individual MemSZ features . . . 53
3.4.3.3 MemSZ-DC evaluation . . . 55
3.5 Conclusions . . . 55

Bibliography **57**

Chapter 1

Introduction

The performance of computer systems is largely dominated by their memory hierarchy as the gap between computing speed and data transfer speed keeps increasing [1]. Besides the long memory latency, memory bandwidth severely limits performance, energy efficiency and scalability of Chip Multiprocessors (CMPs) [2]. On one hand, the demand for higher memory bandwidth increases. Adding more cores on a chip and using specialized accelerators increases the potential processing throughput and calls for higher data rates. New emerging data-intensive applications further increase the need for large volumes of data to be transferred fast [3–5]. On the other hand, memory bandwidth is pin limited [3, 6] and power constrained [7] and is therefore more difficult to scale [2]. More expensive, 3D-stacked DRAM technologies alleviate the bandwidth problem, but due to power constraints cannot keep up with the increasing demand on data rates either [7].

One way to alleviate the memory bandwidth pressure is to reduce the volume of transferred data using compression. Data can then be transferred between the main memory and the processor chip in a compressed form consuming less bandwidth and reducing energy cost. With a few exceptions, hardware main memory compression is limited to lossless methods. Commercial examples of architectures that use memory compression are graphics processing units (GPUs) [8]. GPUs use application-specific compression, applied to texture and color data [9], and often solve the easy part of the problem, handling read-only data [10]. Current state-of-the-art, lossless memory compression techniques achieve on average a 2:1 to 4:1 compression ratio [11]. However, some classes of applications, e.g., multimedia, scientific, forecasting, may allow for more aggressive compression as they inherently tolerate approximations in parts of their data [12, 13] without introducing significant error.

In the past, the performance of memory subsystems has been improved by exploiting the *approximation tolerance* of certain applications. Load value prediction without fetching the actual requested data has been used for improving memory latency and bandwidth [14–16], but has difficulties capturing irregular data variations. Approximate deduplication of individual cachelines increases cache capacity [17], however, multiple values need to match at cacheline granularity. A form of lossy compression has been applied in approximate computing, but is constrained to reducing precision of single values truncating their least

significant bits [10, 18–20] and therefore achieves limited compression ratio.

This thesis proposes aggressive lossy memory compression as a solution to these shortcomings. Carefully selected portions of in-memory data are marked as approximable. Approximable data are divided into blocks and compressed before being written to off-chip memory, reducing the traffic on the bus.

The first part of the thesis, covered in Paper I, which introduces Approximate Value Reconstruction (*AVR*), which uses *downsampling* and *interpolation* for compression. A hardware compressor is introduced between the on-chip Last Level Cache (LLC) and the main memory controller. The on-chip SRAM LLC is adapted to store compressed data read from memory, facilitating reuse and increasing the effective capacity of the cache. The error introduced by lossy compression is limited by two discrete thresholds, allowing the user to control precision-performance trade-off.

The second part of the thesis, covered in Paper II, describes Memory Squeeze (*MemSZ*), which improves upon the preceding design in three ways. First, it includes the more advanced, more effective *SZ* compressor. Second, it adds another layer of error limiting. The accumulated error introduced by compression is monitored, giving the system the ability to detect highly error-sensitive data blocks and disable compression for them. Third, the LLC replacement policy is altered to reduce redundancy between compressed and uncompressed versions of the same data, increasing the storage efficiency.

Furthermore, MemSZ introduces an alternate memory compression architecture which utilizes 3D-stacked DRAM as an LLC, storing only uncompressed data. Approximable data is thus only compressed in main memory. The larger line size supported by a DRAM cache counteracts several of the challenges faced by AVR, yielding a less complex system while still providing the benefits of reduced off-chip traffic.

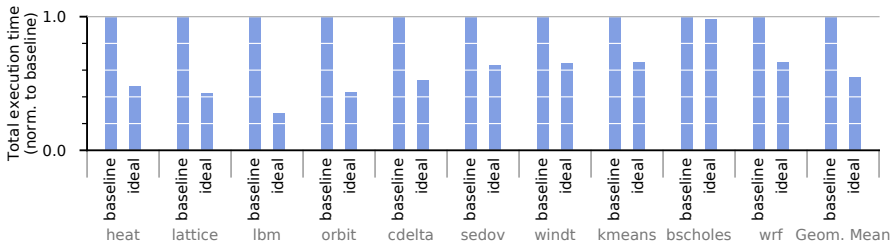
The remaining sections are organized as follows. Section 1.1 describes the problem statement underlying this thesis. Section 1.3 formulates the thesis objectives. Section 1.4 provides a summary of the specific contributions of this thesis. Section 1.5 contains an outline of the remainder of the thesis, which is divided into two parts.

1.1 Problem Statement

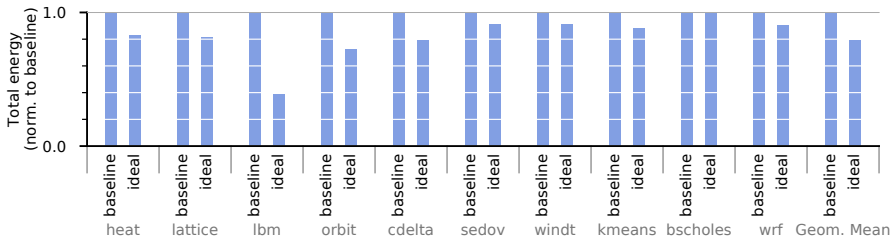
Memory bandwidth is a critical resource in modern systems and has an increasing demand [2]. The large number of on-chip cores and specialized accelerators improves the potential processing throughput but also calls for higher data rates. In addition, new emerging data-intensive applications further increase memory traffic [3–5]. On the other hand, memory bandwidth is pin limited [3,6] and power constrained [7] and is therefore more difficult to scale [2].

The primary drawback of this bandwidth limitation is increased latency of memory operations. Off-chip memory has a high latency compared to on-chip caches, and it increases further when the bus is saturated. Our experiments (Figure 1.1a) show that workloads with high memory intensity can more than double their performance if the memory bandwidth bottleneck is eliminated.

As a direct consequence of extended execution time, energy consumption increases. When processing elements are left idle waiting for memory operations, static power leakage leads to increased overall system consumption.



(a) Execution time



(b) Total system energy

Figure 1.1: Comparison between a baseline system and an ideal system with unlimited memory bandwidth, for a range of applications. Both systems have 8 processor cores at 3.2GHz, an 8MB LLC and 8GB of dual-channel DDR1600.

Furthermore, the main memory itself has been estimated to account for roughly 45% of energy consumption in high-end server systems [21]. A large portion of DRAM energy consumption is leakage and refresh energy, which is also proportional to execution time. In our experiments (Figure 1.1b), total system energy is reduced by up to 60% when the bandwidth bottleneck is removed.

1.2 Related Work

Several techniques have been developed to reduce memory bandwidth consumption. One area of research is *memory compression*, summarized below. Furthermore, this thesis applies *approximate computing*, a paradigm based on trading off computational quality for performance and energy. A summary is provided of relevant designs.

1.2.1 Memory Compression

There is a plethora of lossless memory compression techniques that improve memory capacity and bandwidth utilization. Various compression algorithms are used, such as dictionary-based [22], exploiting frequent patterns or zero-value blocks [23], and more recently similarities of words at the same bit position [11]. However, lossless solutions have limited compression ratio between 2:1 and 4:1. Another aspect is the data placement in memory. Some approaches compact compressed data in memory to improve capacity [24]. Others avoid data compaction, allocating the worst case storage required for the uncompressed data and focus only on memory bandwidth [25, 26]. Finally, managing the metadata

needed for locating and handling the compressed data is also challenging as it may add considerable memory bandwidth overheads [27, 28]. One solution to this is an on-chip cache for metadata [24]. Techniques like Attaché [27] can be applied to further reduce the metadata cost.

The memory compression schemes listed above are lossless. Until recently, lossy compression has been limited to application specific compression, i.e., in GPUs [9], or truncating bits of individual values [10, 18–20], which offered limited compression ratio (2:1 to 4:1). Lossy compression is an example of *Approximate Computing*. Further approximate computing techniques are discussed in the next section.

1.2.2 Approximate Computing

Large classes of applications are inherently tolerant to approximations [12]. This enables a trade-off between the quality of their results and their performance and energy efficiency. This trade-off is exploited by various approximate computing techniques, some of them targeting the aforementioned memory bottlenecks in a lossy manner.

Approximate load value prediction techniques reduce memory latency by providing a predicted value substantially faster than fetching the actual one from memory [14–16]. They may further improve memory bandwidth utilization by not always bringing the actual values at all. Value prediction techniques speculate that the values loaded by the same instruction may be identical or differ by a stride. However, this does not capture any irregular variance of data such as the variance in an image where neighboring pixels may have similar values but may not necessarily differ by a fixed stride. Approximate load value prediction is applied near the core (in parallel with the L1 cache) and is therefore orthogonal to the proposed compression of memory traffic. Another fundamental difference compared to this thesis and in general compared to compression is that load value prediction techniques aim primarily at reducing load latency rather than memory bandwidth because in the end they do fetch the precise values from memory for error checking.

1.3 Thesis Objectives

The objective of this thesis is to reduce traffic between processor and main memory, by exploiting *approximation tolerance*. This is the ability of certain applications to tolerate inaccuracies (approximation) in parts of their data. By reducing traffic, applications can better utilize the limited bandwidth available. For memory-intensive applications, this will lead to reduced memory latency and improved system performance. It is also expected that the improved performance will lead to energy efficiency benefits.

To achieve these goals, this thesis proposes a range of techniques for *lossy memory compression*, i.e. compressing the data transferred between the processor and main memory in a lossy manner. By applying this lossy compression to large blocks, more aggressive compression ratios can be achieved compared to existing (lossless) memory compression systems. The following concrete challenges are discussed:

Large compression block: Compression ratio is directly related to the granularity of compression, the *block size*. Compression at the granularity of individual cache lines limits benefits for two reasons. First, the memory system has a minimum transfer unit which is generally tuned to be the same size as a cache line. As a result, even if a cache line is compressed, a memory request would access a full size line and the benefits of compression would be wasted. Second, compression is based on eliminating information redundancy, and a larger block has a larger potential for such redundancy. For these reasons, a larger block size is desirable.

However, compressing blocks larger than a single cache line leads to additional challenges. A block of multiple cache lines compressed together introduces dependencies between the individual lines. This prevents random access to any single cache line within the compressed block, complicating both memory reads and writes. While a normal memory hierarchy can transfer single cache lines between memory and LLC, a system with multi-line compressed blocks cannot. In order to retrieve a single missing cache line, the entire compressed block it belongs to must be read from main memory. This introduces a traffic overhead compared to a system without compression, as well as latency due to the additional data transferred.

Similarly, a dirty cache line to be evicted from the LLC cannot directly be written back to memory, since the compressed block in memory must be updated in its entirety. In the worst case, eviction of a single cache line requires fetching an entire compressed block (of multiple lines) from memory, decompression on-chip, updating the block, recompression and then finally writeback of the full compressed block.

Low-latency decompression: In a system with memory compression, decompression is on the critical path for memory reads and therefore of crucial importance to system performance. The compression algorithm chosen must therefore offer low decompression latency. However, this potentially conflicts with the need for larger blocks. Depending on the algorithm, processing of larger blocks can lead to increased latency.

Error limiting: Lossy compression deliberately allows inaccuracies in the data. Different applications have different tolerance to such inaccuracy, also depending on which data are approximated. If applied improperly, small approximations in input values can have disproportionately large effects by the time computation completes.

1.4 Thesis Contributions

This thesis describes technical solutions for the use of lossy compression in the memory hierarchy. It tackles the objectives and challenges outlined in Section 1.3 as follows:

Lossy compression: Paper I uses a low-complexity *downsampling* compression method. Individual values which are not compressed with adequate precision are stored along-side the compressed block, allowing for a variable compression ratio. Paper II implements the more advanced Squeeze (SZ) compression algorithm [29]. An additional layer of lossless re-encoding is applied on top of the compression in order to maximize the achieved compression ratio.

Large compression block: In order to achieve higher compression ratios, blocks of 16 consecutive cache lines (1kB) are used. Combined with the lossy compression schemes above, a maximum compression ratio of 16:1 is achieved.

Memory reads: The Last Level Cache is redesigned to keep any combination of compressed blocks and uncompressed cache lines. This allows compressed blocks to be kept on-chip after reading from main memory. Retained blocks are thereby available for reuse, with experiments showing on average 81% of the data from each fetched compressed block used while on-chip. With compression ratios of up to 16:1, the overhead of fetching entire blocks is outweighed by the benefits of compression and reuse.

Memory writes: The empty main memory space between compressed blocks is utilized to temporarily store dirty lines pending recompression. This technique is named *lazy eviction* and allows the system to postpone the overhead of fetching the compressed block and recompressing. Recompression is performed the next time the block is brought on-chip for another reason, potentially combining multiple fetch-and-decompress operations on the same block into one.

Coarse-grained DRAM cache: The challenges above, introduced by compressing blocks of multiple cache lines, are also resolved by the inclusion of a 3D-stacked DRAM last level cache with a line size equal to the compression block size. In such a system, any LLC miss leads to a full-block access in main memory, which means that a compressed block causes no extra overhead. The solution is similarly effective for writebacks since the entire block is kept in the DRAM cache. This eliminates the problem of evicting a single cache line from LLC belonging to a block which is only available in memory.

Low-latency decompression: The downsampling compression used in paper I has a decompression latency of 12 cycles. Paper II presents a heavily parallelized implementation of the SZ compression algorithm, which is able to decompress a full block in at most 18 cycles. In comparison, a single main memory access to one cache line takes between 100 and 400 cycles and an LLC access takes 10-20 cycles. As a result, the total latency of reading out a compressed block and decompressing it is roughly 2 times that of a regular LLC hit, and still far lower than that of an LLC miss.

Error limiting: The approximation error introduced in a block is calculated in parallel with compression. Paper I employs two user-defined error thresholds T_1 and T_2. T_1 controls the maximum tolerated error in any single value. Individual values exceeding T_1 are stored at half-precision alongside the compressed block, resulting in a variable effective compression ratio between 2:1 and 16:1. T_2 controls the maximum average error across an entire block. Blocks exceeding this threshold are left uncompressed, with an exponential back-off mechanism controlling future retries.

Paper II extends this mechanism with an additional error threshold T_3, which limits the total accumulated error across a block throughout the application's lifetime. Blocks exceeding this threshold have compression permanently disabled, to reduce their impact on the application's output quality. In the following section, the contributions of each paper are summarized.

1.4.1 Approximate Value Reconstruction

Paper I describes the design of Approximate Value Reconstruction (*AVR*), a lossy memory compression architecture applicable to a general-purpose multi-processor system.

Concisely, the contributions of paper I are:

- The first use of aggressive lossy memory compression in a general-purpose computing system
- An LLC capable of co-locating compressed and uncompressed data, allowing reuse of compressed blocks
- *Lazy evictions*, avoiding transfer and processing overheads of costly cache evictions
- A two-layer error control mechanism, allowing the user to tune the trade-off between compression ratio and quality degradation

For applications with high approximation tolerance, AVR reduces memory traffic by up to 70%, execution time by up to 55%, and energy costs by up to 20% introducing up to 1.2% error to the application output.

1.4.2 Memory Squeeze

Paper II introduces Memory Squeeze (*MemSZ*), which extends and expands upon AVR with three new features: A more powerful compression method, improved LLC utilization and a more powerful error limiting mechanism. On top of these additions an alternative system organization is proposed, for approximate memory compression in conjunction with a 3D-stacked DRAM cache.

The paper makes the following specific contributions:

- The first low-latency, fully parallelized hardware implementation of the SZ compression algorithm, increasing compression ratio
- A more powerful, error limiting mechanism which complements AVR with the ability to disable compression for individual blocks based on life-time accumulated error
- An improved LLC replacement policy, improving the space utilization by reducing redundancy between compressed blocks and uncompressed cache lines
- A new memory compression architecture, which utilizes a 3D-stacked DRAM cache to reduce the complexity of on-chip support.

MemSZ improves the execution time, energy and memory traffic of AVR by up to 15%, 9%, and 64%, respectively.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Part 2 presents paper I. It introduces *AVR*, an architecture which employs downsampling compression to reduce main memory bus traffic. It also presents an LLC design intended to maintain both compressed and uncompressed data on-chip. AVR takes steps to limit, at execution time, the effects of approximation on the output quality of executed applications.

Part 3 presents paper II. It details *MemSZ*, an extension of AVR with three new features: 1) the more advanced SZ compressor, and an efficient parallelized hardware implementation thereof, 2) a more powerful mechanism for limiting approximation effects on application output, and 3) a modified LLC replacement policy, reducing the redundancy between compressed and uncompressed data. Paper II also presents an alternative architecture, where approximating compression is applied to a system equipped with a 3D-stacked DRAM cache.