Research Statement
Gennady Pekhimenko

gpekhime@cs.cmu.edu

I am interested in doing research that improves both performance and energy efficiency of modern systems, exploring techniques that span the areas of computer architecture, compilers, machine learning, and systems in general. In particular, my future research interests are shaped around the existing trends for the current systems to become more heterogeneous and to operate on massive data sets. My current and past research investigates the potential of hardware-based data compression, which can be viewed as a form of specialized logic, that reduces the complexity and overhead of managing this large amount of data. In this statement, I briefly summarize the contributions of my current and prior research, and then detail my future research plans.

**Data Compression.** One of the most common limiter to computing system performance is usually due to either the lack of capacity of some shared resource such as the main memory or on-chip caches, or the lack of available bandwidth in off-chip or on-chip communication channels. These limitations have become even more significant since many modern applications have to perform computation on massive data sets. One way to utilize the limited shared resources wisely and improve both capacity and bandwidth is through the use of data compression that can represent the data more efficiently – fewer bits to store in memory and send over communication channels. So why don’t we see compression applied everywhere when we look at the whole system stack?

When I started my PhD, I set out to discover why data compression is not more widely applied, and answering this question is one of the major goals of my thesis. As it frequently happens in research, there is no single simple reason. At the high level, there are two fundamental reasons. First, for more than 50 years we have mostly relied on Dennard scaling and Moore’s law to provide large quantities of CMOS-based transistors on-chip to achieve high system performance, so the need for for hardware-based data compression was limited. Unfortunately, this long-lasting trend is coming to an end, and to make things worse, there is no clear candidate technology that could replace CMOS in the next 5-10 years. Second, there are some fundamental challenges that need to be addressed in order to make data compression practical, and these challenges vary depending on where we want to apply it. In my work, I identified those key challenges that prevent hardware-based data compression from being practical across many layers of the system stack, and then proposed solutions to address these challenges for (i) on-chip caches [9, 12], (ii) main memory [11], (iii) off-chip buses and on-chip interconnects [4, 5, 11, 16], and I also investigated (iv) holistic designs where compression can be efficiently applied across multiple levels of the memory hierarchy [11, 12, 16]. My thesis contributions to the area of hardware-based data compression can be categorized in the following way.

- **On-Chip Cache Compression:** First, I propose a simple and fast yet efficient compression algorithm that is suitable for on-chip cache compression. This algorithm solves one of the key challenges for cache compression – achieving low decompression latency, which is on the critical path of the execution, while maintaining high compression ratio. My algorithm is based on the observation that many cache lines have data with low dynamic range, and hence can be represented efficiently using base-delta encoding. I demonstrate the efficiency of the algorithm inspired by this observation (called Base-Delta-Immediate Compression) and the corresponding compressed cache design in my PACT 2012 paper [12]. Finally, in my HPCA 2015 paper [9], I show that compressed block size is a new dimension when making cache replacement decisions that helps to outperform state-of-the-art mechanisms.

- **Main Memory Compression:** I propose a new design for main memory compression, called Linearly Compressed Pages. This mechanism solves a key challenge in realizing data compression in main memory – the disparity on how the data is stored (i.e., page granularity), and how it is accessed (i.e., cache line granularity). This disparity frequently leads to high performance and energy overheads to locate and access the data after compression, because significant amount of compression metadata has to be stored and accessed frequently. I demonstrate the benefits of my approach over prior works on main memory compression in my MICRO 2013 paper [11]. This work won the second place in the ACM Student Research Competition at ASPLOS 2015 [7].

- **Bandwidth Compression:** I show that bandwidth compression, both on-chip and off-chip, can be efficient in providing high effective bandwidth increase in the context of modern CPUs [11, 12], and GPUs [16]. Then, my work finds that there is a new important problem with bandwidth compression that makes compression potentially energy inefficient – the significant increase in the number of bit toggles (i.e., the number of transitions between zeros and ones) that leads to an increase in dynamic energy. This problem was completely overlooked by the prior work on bandwidth compression. My recent works [4, 5] show how dramatic this negative effect can be when data compression is applied in the context of on-chip and off-chip buses in modern GPUs. I propose several potential solutions to this problem in my recent HPCA 2016 paper [5]. This work won the first place in the ACM Student Research Competition at ASPLOS 2015 [8].
Overall, my thesis work shows that there is a significant potential for building better and more efficient systems if we wisely utilize the available hardware resources through data compression by providing higher capacities for on-chip caches and main memory, and higher bandwidth for existing communication channels. My work is done in close collaboration with my industrial partners from Intel, NVIDIA, Microsoft, Qualcomm, and IBM, and with significant involvement of 13 smart undergraduate and graduate students that I mentored during my PhD. In the immediate future, I would like to demonstrate the potential of data compression across the whole system stack such that (i) software can optimize the data allocation to improve compressibility in hardware, (ii) the core can execute directly on the compressed data to avoid compression/decompression overhead and can use the compressed representation for faster computation. In the long term, my goal is to look at new efficient ways to manage and utilize the emerging heterogeneity of modern systems for many important emerging applications such as genome mapping, web search, and online trading. The rest of this statement summarizes my recent and ongoing research, and discusses the directions for my future work.

Recent and Ongoing Research
Hardware-based data compression had some attention in the past, but unfortunately proposed general-purpose designs were not practical. My research journey to solve this issue consists of several major steps.

On-Chip Cache Compression: Base-Delta-Immediate (BDI) Compression Algorithm
In order to make cache compression practical, we have to answer the key question – what is the right compression algorithm for on-chip memory hierarchy? The conventional wisdom before was to aim for the highest compression ratio possible, usually by taking existing algorithms that work by finding common subsets of data and storing them only once (i.e., dictionary-based compression), and then simplifying these algorithms so that they can be implemented in hardware. Instead of following this conventional path, my approach is to prioritize simplicity of the compression algorithm over its efficiency (i.e., compression ratio). To achieve this, I formulate three key principles for the ideal cache compression technique – it should be fast (low decompression latency that is on the execution critical path), simple (no complex or costly hardware changes), and effective in saving storage space. These three principles are typically at odds with each other.

In order to solve the issue, I introduce a new compression technique called Base-Delta-Immediate Compression (BDI) [12] that achieves low decompression latency at high compression ratio and low hardware complexity. It is based on the key observation that many cache lines contain data with low dynamic range: values are similar to each other and/or very small in value, requiring fewer bits to store. For cache lines following this pattern, BDI can represent these values as a small offset (requiring fewer bits) from either a zero base or some other base value. The combined size of the full base and array of small offsets is much smaller than the original cache line.

The resulting BDI compression algorithm has very low decompression latency, i.e., single-cycle decompression, and requires only vector addition, subtraction and comparison operations. Its simplicity and low decompression latency attracted other researchers and catalyzed the research in main memory compression, bandwidth compression, and even memory and cache reliability.

Compressed Cache Management: Exploiting Compressed Size as an Indicator of Future Reuse
Compressed block size can sometimes have unexpected secondary applications. For example, in my recent work [9], I observe that compressed cache block size is sometimes a reliable discriminator between different data structures in an application. At the same time, different data structures have different reuse patterns, hence we can use compressed cache block size to influence cache replacement policy to prefer likely-to-be-reused data. I used this observation to introduce a new Compression-Aware Management Policy that takes into account compressed cache block size along with temporal locality to improve the performance of compressed caches. The performance benefits of this new policy are on-par with the benefits of the initial introduction of data compression for on-chip caches, and the overhead is minimal since compressed size is already part of the compressed block design.

Main Memory Compression: Linearly Compressed Pages (LCP)
Hardware-based data compression is an attractive way to increase main memory capacity, but unfortunately, it requires solving several key challenges. First, unlike on-chip last-level caches, which are managed and accessed at the same granularity (e.g., a 64-byte cache line), main memory is managed and accessed at different granularities. When main memory is compressed, different cache lines within a page can be compressed to different sizes. The main memory address of a cache line is therefore dependent on the sizes of the compressed cache lines that come before it in the page. As a result, the processor (or the memory controller) should explicitly compute the location of a cache line within a compressed main memory page before accessing it. This computation not only increases complexity, but can also lengthen the critical path of accessing the cache line from the main memory and from the physically-addressed cache.
Second, depending on their compressibility, different physical pages are compressed to different sizes. This increases the complexity of the memory management module of the operating system for two reasons: (i) the operating system needs to allow mappings between fixed-size virtual pages and variable-size physical pages and (ii) the operating system must implement mechanisms to efficiently handle fragmentation in memory.

To address these shortcomings, I propose a new approach to compressing pages, called the Linearly Compressed Page (LCP) [11]. The key idea of LCP is to use a fixed size for compressed cache lines within a page (which effectively solves the first challenge), and still enable a page to be compressed even if not all cache lines within the page can be compressed to that fixed size (which enables high compression ratios). LCP is effective in providing high compression ratio for main memory with low overhead, and can also be used in a combined design\(^1\) with cache compression to achieve even higher system performance [11].

Energy-Efficient Bandwidth Compression: Toggle-Awareness

In the context of modern GPUs, where off-chip bandwidth is frequently the major bottleneck, techniques like bandwidth compression can be especially efficient [16]. But there is one problem that was previously overlooked with respect to how data compression affects data transferred over communication channels. I find that the bit toggle count (i.e., the number of times we transition between zeros and ones) can increase dramatically after compression is applied [4] which, in turn, can significantly increase the energy spent while transmitting compressed data. Then, I find simple solutions [5] that can efficiently monitor this negative effect of compression, and only apply compression when it is beneficial for both energy and performance.

Other Research Topics

I have been actively involved in the research projects outside the scope of my thesis.

**Systems.** I work on web search systems for mobile phones where users’ interest in certain trending events can be predicted and efficiently prefetched to extend the phone’s battery life [10]. Previously, I also worked on improving the compile time of existing compilers with machine learning techniques that can predict which optimizations are actually useful for performance [6].

**Main Memory.** In collaboration with Vivek Seshadri, I propose several ways of better utilizing existing DRAM-based main memories: (i) fast bulk data operations like copying and memory initialization using RowClone [13], and (ii) an enhanced virtual memory framework that enables fine-grained memory management [14]. In collaboration with Donghyuk Lee, I work on (i) reducing the latency of existing DRAM memories [3], and (ii) increasing the bandwidth available for the existing (and future) 3D stacking designs [2]. In collaboration with Hasan Hassan, I also work on reducing DRAM latency by exploiting our new observation that many DRAM rows can be accessed significantly faster since they have sufficient amount of charge left [1].

**Bioinformatics.** In collaboration with Hongyi Xin, I work on new filters for alignment in genome read mapping [17], and techniques to find the optimal seeds for a particular read in the genome mapping process [18].

**Approximate Computing.** Together with my collaborators from Georgia Tech, I work on rollback-free value prediction mechanisms for both CPUs [15] and GPUs [19,20].

Future Research

My future research interests are shaped around the existing trends for the current systems to become more heterogeneous and operate on massive data sets. Below I present several representative ideas I would like to work on.

**Execution on Compressed Data.** My thesis work on data compression significantly advanced this subfield of computer architecture, but as it commonly happens, also highlighted some completely new problems and opportunities. One major problem in all prior designs is that compressed data needs to be decompressed somewhere on the way to the core. While it might seem more attractive to always decompress data before execution, since it usually requires less change to the existing system, this internal limitation of performing all operations on the uncompressed data can potentially lead to significant energy waste and suboptimal performance. At the same time, if the final operation performed on the data is relatively simple (e.g., arithmetic comparison), it might be possible to perform these operations on the compressed data itself (this is somewhat similar to the execution on encrypted data in homomorphic encryption).

Let us consider an example where an application performs a simple linear scan through an array searching for a certain value.\(^2\) If this array is already compressed with a certain compression algorithm like BDI, then one simple strategy is to try to represent the searched value in the same base-delta form. If it cannot be represented in this form, then this value is not in this array, and there is no need to do any per-value comparisons. In cases where this representation is possible, we still need to do value comparisons, but more narrow – instead of say 8-byte comparison for the original value, we can do 1–4 byte comparisons between deltas using SIMD (single instruction, multiple data) instructions.

There are several major research questions that I would like to investigate. First, what are the correct abstractions to expose

---

\(^1\)Described in more details in Safari Tech. Report 2012-005.

\(^2\)This scenario can be quite common for existing database queries, search engine queries etc.
the compressed representation to the application? For example, can we combine this idea with some sort of virtual memory extension, so that the user can also see the compressed data directly in another part of the address space or even manipulate that data as long as the representation remains valid? Second, what is the best substrate to implement operations on compressed data? Should we use existing arithmetic instructions or special narrow SIMD-like instructions? What are the corresponding changes in hardware/software needed to support this idea?

**Energy Efficiency Through Heterogeneity.** I strongly believe that the future computational systems should be heterogeneous so we can achieve both performance and energy efficiency. Recent works (e.g., Catapult project from Microsoft) already show how efficient modern accelerators (e.g., FPGAs) can be in speeding up web search, trading operations, domain-specific compression etc. Based on my background in Bioinformatics [17, 18], machine learning [6], and web search [10], I see a significant potential for applying different forms of specialization and acceleration to improve the energy efficiency for many applications from these important areas.

There are several key challenges, which span the areas of computer architecture, programming languages, and design tools, we need to address to realize this potential. First, a powerful computational substrate needs to be defined that is suitable for a particular workload. For every domain-specific problem, I would like to identify the key characteristics of its computational routines and what type of hardware (e.g., CPUs, GPUs, FPGAs etc.) is best to achieve both high performance and high energy efficiency. For example, some of my previous works in the field of Bioinformatics [17, 18] already show that many genome mapping algorithms can be efficiently mapped to modern GPUs, while some algorithms that require higher flexibility would benefit more from mapping to FPGAs.

Second, there is a need for a flexible programming model and corresponding compiler support to convey characteristics about these applications (e.g., the level of parallelism, type of dependencies etc.) to the actual hardware. Currently, the burden of a proper mapping is almost fully on the application developer, which is a very non-trivial task to do requiring both deep understanding of the hardware resources available and applications algorithm. To make things worse, the mapping that was good for one generation of the accelerator, might not be good for the next one, and hence retuning might be needed.

In the short term, I would like to investigate new ways to move most of the complexity from the programmer (i) to the programming models (both domain-specific and generic) and (ii) to the static/dynamic tools such as compilers and runtimes. In the longer term, I would like to find new ways to support the efficient communication, resource sharing, and scheduling between many, potentially very different, architectures within a heterogeneous system.

**New Efficient Representations for Big Data.** Many modern applications, such as machine learning applications, start to operate on data sets that significantly exceed the available main memory. At the same time, these applications do not always require the full precision of the computation as their input data can be already significantly imprecise. In my long term plans, I would like to investigate the potential of partially replacing the accesses to the huge data sets in these applications with the accesses to their much smaller representations or signatures. The key idea is to build a lower-resolution representation of the data set, keep it up-to-date in main memory, and refer to it when information to this data set is missing in the main memory. We then dynamically monitor whether the application meets its desired quality of output, and update the aggressiveness of our speculation accordingly.

**My Research Philosophy**

In order to be successful in my research journey, and more specifically when working on my thesis, I made several early strategic decisions. First, I chose to use my expertise in both systems and architecture, as it was evident from the beginning that the most efficient solutions frequently lay in between main memory. At the same time, these applications do not always require the full precision of the computation as their input data can be already significantly imprecise. In my long term plans, I would like to investigate the potential of partially replacing the accesses to the huge data sets in these applications with the accesses to their much smaller representations or signatures. The key idea is to build a lower-resolution representation of the data set, keep it up-to-date in main memory, and refer to it when information to this data set is missing in the main memory. We then dynamically monitor whether the application meets its desired quality of output, and update the aggressiveness of our speculation accordingly.

And last but not least, I actively recruited and mentored other students to help me on this and other projects. Overall, as of now, I have mentored thirteen students of all levels (undergraduates, masters, and junior PhDs from both CS and ECE) in the span of the last four years. This mentoring not only helped me to progress faster in my research and learn many useful skills on how to manage and work with students of different seniority and background, but also helped students to learn what research is about and frequently even publish top-tier conference papers together. I plan to continue this research strategy in my future career.
References


Research Statement: Gennady Pekhimenko