IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version Oct 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Energy consumption in compact integer vectors: A study case

JOSÉ FUENTES-SEPÚLVEDA[1], SUSANA LADRA[2]
[1]Universidad de Chile, Department of Computer Science, Santiago, Chile (e-mail: jfuentess@dcc.uchile.cl)
[2]Universidade da Coruña, CITIC, Facultad de Informática, A Coruña, Spain (e-mail: susana.ladra@udc.es)

Corresponding author: Susana Ladra (e-mail: susana.ladra@udc.es).

This research has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie [grant agreement No 690941]; from the Ministerio de Ciencia, Innovación y Universidades (PGE and ERDF) [grant numbers TIN2016-77158-C4-3-R; RTC-2017-5908-7]; from Xunta de Galicia (co-founded with ERDF) [grant numbers ED431C 2017/58; ED431G/01]; from postdoctoral grant of Comisión Nacional de Investigación Científica y Tecnológica, CONICYT [grant number 3170534].

ABSTRACT In the field of algorithms and data structures analysis and design, most of the researchers focus only on the space/time trade-off, and little attention has been paid to energy consumption. Moreover, most of the efforts in the field of Green Computing have been devoted to hardware-related issues, being green software in its infancy. Optimizing the usage of computing resources, minimizing power consumption or increasing battery life are some of the goals of this field of research.

As an attempt to address the most recent sustainability challenges, we must incorporate the energy consumption as a first-class constraint when designing new compact data structures. Thus, as a preliminary work to reach that goal, we first need to understand the factors that impact on the energy consumption and their relation with compression. In this work, we study the energy consumption required by several integer vector representations. We execute typical operations over datasets of different nature. We can see that, as commonly believed, energy consumption is highly related to the time required by the process, but not always. We analyze other parameters, such as number of instructions, number of CPU cycles, memory loads, among others.

INDEX TERMS algorithms, compact data structures, data compression, energy consumption, integer vectors

I. INTRODUCTION

We are surrounded by digital information, such as the huge amount of data generated on the Internet and also that we are collecting in our daily lives: human generated data, both consciously (such as emails, tweets, pictures, voice) and unconsciously (clicks, likes, follows, logs, ...), or observed data (biological, astronomical, etc.). When managing large volumes of digital information, data compression has always been considered to be vitally important. Traditionally, data compression focused on obtaining the smallest representation possible, in order to save space and transmission time, thus, providing a good archival method. However, most of the compression techniques require decompressing the data when they need to be accessed, especially when these accesses are not sequential, and thus, limiting the applicability of data compression.

To overcome these issues, compact data structures appeared in the 1990's and rapidly evolved during early years of the current century [1]. They use compression strategies to reduce the size of the stored data, taking advantage of the patterns existing in the data, but with a key difference: data can be directly managed and queried in compressed form, without requiring prior decompression. The main contribution is that they allow larger datasets fit in faster levels of the memory hierarchy than classical representations, thus, dramatically improving processing times. In addition, many compact data structures are equipped with additional information that, within the same compressed space, acts as index and speeds up queries.

Nowadays, multiple compact data structures have been proposed for representing data of different nature, from simple bitvectors or sequences, to other complex types of data, such as permutations, trees, grids, binary relations, graphs, tries, text collections, and others [1]. Compact data structures have now reached a maturity level, with some of them being

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access

used in real systems from other communities, such as in Information Retrieval or Bioinformatics. For instance, some of these compressed data structures are the core of the existing genome assemblers or DNA aligners [2], [3], or have been shown competitive compared with traditional solutions for text or document retrieval [4], [5].

Despite of the benefits that compact data structures can provide, they are not present in most of the internet-connected devices that are part of our daily lives. There are now more of these devices than there are people in the world, and is expected that by 2020 they will outnumber humans by 4-to-1, based on the last forecast by Gartner. Smartphones, smart TVs, wearables, healthcare sensors, etc., are expected to proliferate in the near future, as the development of smart cities, industry 4.0, autonomous cars or unmanned aerial vehicles is becoming a reality. Most of these devices include decision-making algorithms that require collecting, storing and processing large amounts of data using little space, which would seem to be a perfect scenario for compact data structures. Compact data structures have received few attention in these application domains, and this can be related to disregarding energy at the design step of compact data structures. Addressing one of the main biggest limitation of these devices, their battery life, is vital, as they generally operate far from any power supply. Up to today, few researchers in the field of compact data structures expressed any concern about energy consumption [6], and there is no previous work on designing energy-aware compact data structures.

Power management and energy efficiency have been the main focus in Green Computing. Most of the research efforts in this field were devoted to reduce data centers' carbon footprint, or dealing with electrical and computer systems engineering [7]. However, we are still far from being able to develop energy-efficient software due to two main problems: the lack of knowledge and the lack of tools [8]. Despite this fact, green software is gaining importance, especially in the areas of quality and design/construction, followed by requirements [9]. There are also some previous research on rethinking the way query processing was performed to become energy-aware, such as in the case of database management systems [10], [11]. Nevertheless, there is an absence of any science of power management, with a relatively small amount of research in the core areas of computer science [12], [13].

To date, compressing data has been considered before as a technique to reduce energy requirements [14], as it can improve the way data is stored in memory, moving them as close as possible to the processing entity [15]. However, the design of energy-efficient compact data structures is challenging, as compact data structures usually rearrange data to enable fast accesses while being compressed. These rearrangements would probably lead to memory pattern accesses such those that negatively impact energy consumption [16] and even increase temperature [17]. Moreover, accessing compact data structures generally requires more complex computations than their plain counterparts, thus, also increasing energy requirements [15]. In addition, not all compact data structures allow increasing simultaneously time and space performance [1], thus, energy comes into play as a new dimension to be taken into account.

In this paper, we present a case of study in compact integer vectors, trying to get a deeper understanding of the different parameters and factors that impact on energy consumption.

II. BACKGROUND

A. BACKGROUND ON COMPRESSED DATA STRUCTURES

Data compression has always been a popular research area, as the amount of data we manage has been increasing continuously beyond the capabilities of the storage and processing systems [18]. Compression is not only a matter of reducing space disk, but most importantly, reducing transmission and processing times [19]. Thus, as we are now dealing with huge volumes of data of different nature, compression is definitively presented as a real solution to address, in parallel with other approaches, the phenomenon of what is commonly known as Big Data.

Traditional compression techniques are usually designed to achieve the highest compression rate possible. However, there exist numerous scenarios where it is desirable to obtain better processing capabilities over the data than just obtain the most reduced space possible. For instance, there exist text compression techniques that allow for the retrieval of snippets located at random positions in the text without the need for decompressing the whole text [20], [21], which is a very desirable property in many applications, as compression and decompression phases are demanding processes both in terms of memory and time consumption. Moreover, it has been proven that searching string patterns directly over the compressed text is up to 8 times faster than searching over plain text [21], [22]. Of course, these capabilities are usually provided at the expense of some compression ratio.

In addition to the data itself, indexes or additional structures are usually required for supporting processing or querying over the data in efficient time. These data structures can be one or even two orders of magnitude larger than the data [1]. One of the classic examples is the suffix tree that enables efficient sequence analysis over one genome [23]. In the case of human genomes, the data itself can occupy no more than 800 megabytes, whereas the suffix tree requires more than 30 gigabytes. Compact data structures are precisely aimed at addressing these issues. They maintain the data and their additional indexes using less space than the data itself and allowing efficient processing and querying of the data, without the need of decompression.

The beginnings of compact data structures can be dated back to the year 1988, when Jacobson introduced, in his PhD thesis, a new set of data structures, named succinct data structures, which use $\log N + o(\log N)$ bits for representing N different objects [24][1]. Some authors also used the term

[1]We use $\log x$ to mean the base 2 logarithm of x unless specified otherwise.

compressed data structure when proposing new data structures that use $\mathcal{H} + o(\log N)$, $\mathcal{H}$ being the entropy of the data under some compression model [25]–[27]. In the last few years, several other proposals have emerged, boosting the maturity of this field. The term *compact data structures* has recently been adopted by the community [1] to include all these types of data structures, generalizing the term to denote all data structures that use little space and query time, without specifying explicit complexity bounds. There exist multiple compact data structures for different problems: from the most basic needs such as representing arrays supporting reading and writing values at arbitrary positions [28]–[30], bitvectors supporting bit-counting operations [31]–[33], permutations [34], [35], sequences of symbols supporting counting and searching operations [36]–[39], to more complex data, such as trees [28], [40], [41], graphs [42], [43], grids [44]–[46], texts [4], [5], [47], geographical information [48], [49], etc.

B. BACKGROUND ON GREEN COMPUTING

One of the main challenges that information technologies have to face is to strike a balance between their enormous potential for growth and the environmental impact that this is causing to our planet. There is an increasing awareness by the various actors (researchers, governments, industry, etc.) of the importance of developing technological solutions that make an efficient use of energy and other resources, thereby promoting a long-term technological sustainability. Energy savings in data centers, ecological manufacturing of components, extending equipment longevity, etc. are some of the goals of what has been called Green Computing.

Much effort in Green Computing has been devoted to promoting a wise usage of the resources and improving energy consumptions of hardware components or IT infrastructures. However, the industry road map needs to move its efforts to finding solutions in Green Software [50]. Instead of making better chips, the focus will be centered in the different applications developed for smartphones, supercomputers, or even data centers in the cloud. Then, after understanding what these efficient high-level applications require, it will be possible to go down to develop the chips and hardware to support their needs. Research in Green Software is therefore gaining more attention, especially in the areas of quality and design/construction, followed by requirements [9]. Moreover, computing is not defined any more by the needs of traditional PCs or data centers. Today, many people live surrounded by multiple mobile computing devices, such as smartphones, tablets, wearables, smart TVs, smart home products, and other kinds of sensors and IoT devices. In this scenario, energy efficient software is of extreme importance, as it can save battery and reduce the heat generated in the devices, which in turns increases the speed and longevity of these mobile devices. Green IoT has emerged as a field of research to tackle this problem, proposing generally hardware-based approaches, in addition to other efforts that can be categorized as software-based, habitual-based, awareness-based, policy-based, or recycling-based [51]; however, approaches using compact data structure have not been used in this context up to now.

Among the previous work related to energy efficiency, there is little research within the core areas of computer science [12]. Most of this research focuses on online problems related to power management, including energy saving mechanisms based on power-down mechanisms and speed scaling [52]. Regarding energy-efficient algorithm design, some authors have proposed new approaches, such as the reversible computation approach, where inputs can be reconstructed from their outputs [53]. Besides this research, some works exist focusing on finding models or understanding the factors affecting energy consumption [54], [55]. However, there is no research in the particular field of compact data structures; thus, it becomes crucial to study the relationship between the spatial and temporal complexities with the energy dimension including factors such as entropy, among others.

To date, there is a recognized lack of knowledge and tools for the development of energy-efficient software [8], which must firstly be tackled to properly address the ecodesign of compact data structures. This absence of any science of power management was identified almost 10 years ago [12], [13], and there have been no significant advance since then in the field of algorithms and data structures analysis and design. There exist some models proposed for database management systems, software energy optimization for embedded systems [10], [11], [56], or more general energy models for algorithmic engineering [54], [55].

C. GREENING COMPACT DATA STRUCTURES: THE CHALLENGES

The space reduction achieved by compact data structures usually comes at the expense of degrading the locality of reference, and this can significantly damage energy consumption [57]. As an example, one can consider wavelet trees [36], one of the most known and used compact data structures thanks to its multiple applications in the field of information retrieval [58]. Wavelet trees are data structures that represent sequences of symbols in little space and can answer some queries over them efficiently. As most compact data structures, wavelet trees rearrange data in order to obtain good spatial and temporal properties, in addition to enabling fast counting, searching and direct accessing over the sequence. These rearrangements cause a lack of locality of reference, and consequently, memory access patterns that degrade energy consumption.

More concretely, given a sequence S of length n, $S = S_1 S_2 \dots S_n$, composed of symbols from an alphabet $\Sigma = \{s_1, s_2, \dots, s_\sigma\}$, the wavelet tree is a balanced binary tree that divides the alphabet into two parts at each node. Each of these nodes of the tree contains a bitvector, which represents, for each position, if the corresponding symbol belongs to the lower (0) or to the upper (1) partition of the alphabet. More concretely, the recursive construction of the wavelet tree is as follows. The root node of the tree contains a bitvector ($B = b_1 b_2 b_n$) of the same size as the sequence (n), where

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access

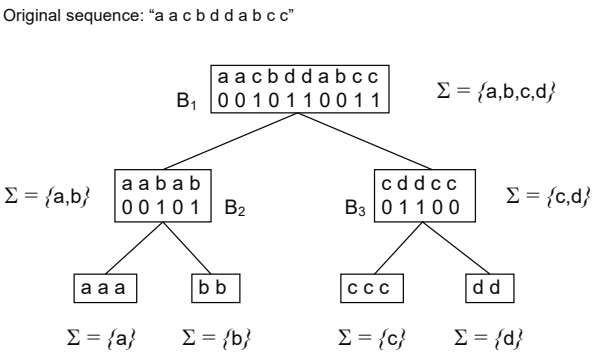


FIGURE 1: Example with a sequence of symbols from the alphabet $\Sigma = \{a, b, c, d\}$ and its corresponding wavelet tree. Text is shown only for clarity, but is not actually stored.

at position i we have $b_i = 0$ in case $S_i \in \{s_1, s_2, \ldots, s_{\sigma/2}\}$, or $b_i = 1$ in case $S_i \in \{s_{\sigma/2+1}, \ldots, s_\sigma\}$. Those symbols marked with a 0 are processed in the left child of the node, and those marked 1 are processed in the right child of the node. Then, for each internal node the procedure is repeated recursively. As the alphabet indexed by a child node is only half of that of its parent, the construction stops when the alphabet is composed of just one symbol.

Figure 1 shows an example of a wavelet tree for the sequence "$aacbddabcc$" over the alphabet $\Sigma = \{a, b, c, d\}$. Only the bitmaps at each internal node are necessary to represent the sequence, as the alphabet corresponding to each node can be implicitly recovered following the path from the root tree to that node. Due to its construction procedure, the wavelet tree has a leaf node for each symbol of the alphabet.

As can be seen in Figure 1, a sequence represented in plain form ($aacbddabcc$), is now rearranged in three different bitmaps following a two-level tree. Thanks to these rearrangements, this representation provides multiple benefits, as it allows counting and searching symbols in the sequence in efficient time and within less space. The plain form is only superior in terms of speed when recovering sequentially some substring of the sequence. As the size of the alphabet grows, the number of levels of the tree increases, and thus, extracting a portion of the original sequence requires multiple accesses to several nodes of the wavelet tree that are not contiguous in main memory, therefore, causing cache misses, movements of data within the memory hierarchy, generating a high energy consumption.

These rearrangements are also present in compact representation of trees, graphs, binary relations and texts. An example of this is the Burrows-Wheeler Transform (BWT) [59], a classical algorithm in the field of data compression, which is used, for instance, in the known compressor bzip2. Given a string of characters, BWT rearranges them with the goal of obtaining long runs of similar characters, improving its compressibility. The BWT is the basis of the FM-Index [60], which is at the core of well-known bioinformatics software, such as the Burrows-Wheeler Aligner (BWA) [3], Bowtie aligner [2], or the SGA Assembler [61]. The FM-index uses a technique for locating patterns in the text called backwards search, which jumps to random positions of the rearranged text to find the occurrences of the searched pattern. Again, this may cause cache misses, movements of data within the memory hierarchy, which penalize energy consumption.

III. STUDY CASE: COMPRESSED INTEGER VECTORS

A. COMPRESSED INTEGER VECTORS

For this first study of energy consumption on compact data structures we evaluate the energy performance of one of the most basic compact data structures, Compressed Integer Vectors (CIV). The CIVs are building-blocks of more complex compact data structures, such as, compressed suffix arrays [66] and inverted indexes [64]. We tested several CIVs present in the literature.

Let $X = x_1, x_2, \ldots, x_n$ be the sequence of n integers to encode. A way to compress X is to use statistical compression, that is, create a vocabulary of the different integers that appear in X, sort them by their frequency, and assign shorter codewords to those values that occur more frequently. In the case of domains where the smallest values are assumed to be more frequent, one can directly encode the numbers with a fixed instantaneous code that gives shorter codewords to smaller numbers. Thus, it is not necessary to maintain a vocabulary of symbols sorted by frequency, which may be prohibitive if the set of distinct numbers is too large. For this scenario, the best-known encoding schemes are unary codes, γ-codes, δ-codes, and Rice codes [62], [67], [68].

There exist some recent fast decodable representations, such as Simple9 [64], Simple16 and PforDelta [65], which achieve fast decoding and little space. The approach used by these techniques is to pack a number of small integers in a computer word, using the number of bits needed by the largest number. For instance, Simple9 packs the integers of the original sequence into words of 32 bits, computing the maximum number of consecutive integers that can be included in a word using the same number of bits for all. For example it can encode 28 1-bit numbers, 14 2-bit numbers, 9 3-bit numbers, and so on. PForDelta can use more than 32 bits (say, 256), and treat the 10% largest numbers as exceptions that are encoded separately. These techniques perform very well in practice.

There exist other compression techniques based on searching for *runs* in the original stream of data, where a run is an interrupted sequence of the same value. When using run-length (RL) encoding, the original sequence is replaced by the representation of its runs, each of them encoded with two integers: the value that is repeated (run value) and the number of times it is repeated (run length).

Specifically, in this work we tested CIVs based on the γ-code and δ-code of Elias [62], the Directly Addressable Codes (DAC) of Brisaboa *et al.* [63], the data structure of Ferragina and Venturini (FV) [30], Simple9 of Anh and Moffat [64], PforDelta of Zukowski *et al.* [65] and a compressed vector based on run-length encoding (RL). To improve the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access

	Space (bits)	$access()$	$next()$	Description
γ-code [62]	$2N + O(\frac{n}{h} \log N)$	$O(h)$	$O(1)$	Sampling step h
δ-code [62]	$N + 2n \log \frac{N}{n} + O(n + \frac{n}{h} \log N)$	$O(h)$	$O(1)$	Sampling step h
DAC [63]	$\lceil N(1 + \frac{1}{b}) \rceil + nb + o(\frac{N}{b})$	$O(\log x_m / b)$	$O(\log x_m / b)$	parameter $b \leq \log x_m$
FV [30]	$N_{fv} + O(\frac{n}{h}(\log N_{fv} + \log(h \log x_m)))$	$O(1)$	$O(1)$	Sampling step h
Simple9 [64]	$N_{s9} + \frac{n}{h} \log N_{s9}$	$O(h)$	$O(1)$	Sampling step h
PforDelta [65]	$N_{pfd} + \frac{n}{h} \log N_{pfd}$	$O(h)$	$O(1)$	Sampling step h
RL	$2r(\log n - \frac{\log r}{2} + 1) + o(r)$	$O(\log \frac{n}{r}) + O(\frac{\log^4 r}{\log n})$	$O(\frac{\log^4 r}{\log n})$	Number of runs, r

TABLE 1: Compressed integer vectors used in the study. Given an integer vector $X = x_1, x_2, \ldots, x_n$, we define $x_m = max\{x_i | 1 \leq i \leq n\}$, and $N = \sum_{i=1}^{n} (\lfloor \log x_i \rfloor + 1)$ the minimum number of bits to represent all the entries of X. For Simple9, PforDelta and FV we define N_{s9}, N_{pfd} and $N_{fv} = \sum_{i=1}^{n} \lfloor \log i \rfloor$ as the number of bits needed to represent the vector X with each technique and without sampling.

random access time of the CIVs based on γ-code, δ-code, Simple9 and PforDelta, we complement them with a sampling array, storing the position of each h-th entry of input vector. The RL structure is composed of an array $H[1..r]$ storing the first value of each run, and a bitvector $B[1..n]$ with rank/select support, marking the beginning of each run. With the objective of reducing the overall space of RL for vectors with long runs, the bitvector B corresponds to the sparse bitvector sdarray of Okanohara and Sadakane [31]. Table 1 summarizes all the tested CIVs. The table shows the space usage of each structure, the time complexity of the operation $access(i)$, which recovers the i-th entry on the vector, and the time complexity of the operation $next(i)$, which recovers the $(i+1)$-th assuming that the i-th entry was already recovered. The operation $access()$ is preferred to random access of the entries of a CIV, meanwhile the operation $next()$ is preferred to a left-to-right scanning of the CIV. In general, the usage of the $next()$ operation in a sequential scanning is faster than using the $access()$ operation, because we can store partial results in local variables, since we already know the next entry to be recovered. Note that, from the definition of the operation $next()$, an operation $access()$ must be applied at the beginning of the left-to-right scanning. For example, we could recover the entry at position $i + 2$ using the expression $next(next(access(i)))$.

B. ACCESS PATTERNS

The aim of this study case is to measure the impact in the energy consumption of the particular data rearrangement of each CIV for the entries of the input vector. For that, we performed two sets of experiments. In the first set, we performed several binary searches over the CIVs. We decided to test binary search because it is one of the most classical algorithms in Computer Science. Additionally, binary search performs up to $\log n$ non-consecutive accesses to a vector of length n, allowing us to study the impact of $access()$ operation. The second set of experiments corresponds to the computation of the sum of m entries of the vector. We tested a random access pattern, choosing the m entries randomly, and a sequential access pattern, choosing the first m entries in increasing positional order. For random access we use the $access()$ operation and for sequential accesses we use the $next()$ operation. The sum operation is one of the most basic and efficient CPU instructions implemented in modern computers. Thus, we argue that the resulting values obtained in Section IV-C are mainly due to the data rearrangement of the CIVs instead of some overhead due to the sum operation.

IV. EXPERIMENTAL EVALUATION

A. EXPERIMENTAL SETUP

In our experiments, we set the sampling steps of the tested CIVs to values that offer a good trade-off between space and access time. Thus, for the CIVs δ-code, γ-code, FV (`fv`) and Simple9 (`s9`), the sampling value was 128, and for PforDelta (`pfd`) the sampling value was 1024. The parameter b of DAC (`dac`) was chosen by an optimization algorithm [63] that computes a different b value for each level of the representation, reducing the final size of the representation.

Depending on the properties of the integer sequence, it is more convenient sometimes to store the differences between two consecutive elements of the vector instead of the elements themselves. To test such situation, we implemented modified versions of the CIVs of Table 1 to store the differences. To deal with negative differences we use the ZigZag encoding[2]. For the rest of the document, we identify the modified versions with the suffix `_zz`.

To measure the energy efficiency of the CIVs, we tested two datasets. The first group corresponds to a sorted integer vector of length 104,857,600, called `sorted`. The elements of the vector were randomly selected from the range $[0..2^{30}]$. This dataset will be used to test the binary search algorithm. The second group corresponds to four datasets from the Pizza&Chili Corpus: *i)* `dblp`, an XML document of publications on Computer Science; *ii)* the repetitive datasets `cere`, the DNA sequencing of the yeast Saccharomyces Cerevisiae; *iii)* `kernel`, a collection of Linux Kernels; and *iv)* `eins`, the English versions up to November of 2006 of the Wikipedia

[2] *ZigZag encoding* maps signed integers to unsigned integers using a "zig-zag" strategy over the positive and negative integers, so that numbers with small absolute values are mapped to small unsigned integers. For example, -1 is encoded as 1, 1 is encoded as 2, -2 is encoded as 3, 2 is encoded as 4, and so on.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access

	cere		kernel		eins		dblp	
bwt	max val:	84	max val:	255	max val:	252	max val:	126
	avg val:	72	avg val:	77	avg val:	91	avg val:	86
	max diff:	-78	max diff:	-223	max diff:	220	max diff:	111
	runs:	9,438,540	runs:	2,327,498	runs:	105,892	runs:	15,060,348
lcp	max val:	32,469	max val:	1,466,191	max val:	935,920	max val:	1,084
	avg val:	2,129	avg val:	113,361	avg val:	41,322	avg val:	44
	max diff:	29,562	max diff:	-1,456,752	max diff:	893,180	max diff:	961
psi	max val:	104,857,600	max val:	104,857,600	max val:	104,857,600	max val:	104,857,600
	avg val:	52,428,800	avg val:	52,428,800	avg val:	52,428,800	avg val:	52,428,800
	max diff:	-104,857,295	max diff:	-104,851,818	max diff:	-104,846,882	max diff:	-104,854,855

TABLE 2: Statistics of the datasets.

article of Albert Einstein. For each dataset, we extracted the first 100 MB of data, equivalent to 104,857,600 symbols. In order to study different types of vectors, we computed the Burrows-Wheeler Transform (`bwt`), the longest common prefix array (`lcp`) and the Ψ function used in compressed suffix arrays (`psi`). The `bwt` vectors have ranges of equal letters, called runs, especially for repetitive datasets; for non-repetitive datasets, such as `dblp`, most of the entries of the `lcp` vectors are small compared to the length of the vector; and the `psi` vectors have ranges of increasing values, especially for repetitive datasets. This second group of datasets will be used to test the random and sequential access patterns of the sum of entries. Table 2 shows some statistics of the datasets and Table 3 shows the space consumption of the CIVs for all the datasets. The results correspond to the expected behaviour of each CIV taking into account the distribution of values of each dataset. `rl` obtains the best compression for those `bwt` datasets with a low number of runs, that is, for those datasets with longer runs, namely `kernel` and `eins`. For `lcp` datasets, the techniques that obtain the best results are `dac` and `dac_zz`. `pfd_zz` is the method achieving the smallest representations for the rest of the datasets, and also the one that obtains the best overall performance.

The experiments were carried out on a Non-uniform memory access (NUMA) machine with two NUMA nodes or packages. Each NUMA node includes a 8-core Intel Xeon CPU (E5-2470) processor clocked at 2.3 GHz. The machine runs Linux 4.9.0-6-amd64, in 64-bit mode. Each core has L1i, L1d and L2 caches of size 32 KB, 32 KB and 256 KB, respectively. All the cores of a NUMA node share a L3 cache of 20 MB. With respect to the associativity of each cache, which may impact in the energy consumption, L1i, L1d and L2 are 8-way set-associative, and L3 is 20-way set-associative. All cache levels implement the write-back policy. Each NUMA node has a 31 GB DDR3 RAM memory, clocked at 1067 MHz. Hyperthreading was enabled, giving a total of 16 logical cores per NUMA node. The algorithms were implemented in C++ and compiled with GCC and -O3 optimization flag. According to the work of Kambadur and Kim [69], the optimization flag -O3 of GCC offers significant energy savings. We measured the running time using the `clock_gettime` function at nanosecond resolution. Each experiment was repeated 10 times and the median is reported.

B. POWER MEASUREMENT

To measure the energy consumption of the CIVs, we used the Intel RAPL (Running Average Power Limit) Interface [70]. The Intel RAPL Interface aggregates the content of several specialized registers, called Model Specific Registers (MSR), to provide an estimation of the energy consumption at cores level (all cores in a processor), package level (all cores in a processor, memory controller, last level cache, among other components) and DRAM memory level. Depending on the processor model of the machine executing the experiments, we may have access to the energy estimation of only the cores level and package level, which is our case. In this work we report the energy estimation of the package level.[3] The Intel RAPL Interface has been used in previous energy-consumption studies and it has provided reasonably accurate measurements [71], [72]. We used the Linux profiler Perf to report the energy estimations of Intel RAPL. Perf also allows us to measure more metrics, such as, CPU cycles, cache hits and cache misses of the L1 cache and the last level cache, number of instructions, etc. We used Perf 4.9.110 in the experiments.

C. RESULTS AND DISCUSSION

Figures 2 shows the running time and the energy consumption from zero up to 24,000 binary searches over the integer vector `sorted`, where each binary search performs up to $\log(104,857,600) \approx 27$ random accesses to the vector. We only report the CIVs whose sizes are smaller than those of their corresponding vectors in plain form. Both the running time and the energy consumption increase with the number of binary searches. When the number of binary searches is zero, the vector is loaded in memory, without accessing their values. Among of the tested CIVs, `dac_zz` is the one using less space but not the most efficient in running time

[3]We replicated part of our experimental in a machine with reduced memory capacity and with access to the three levels of energy estimations (cores, package and DRAM memory). We observed that, on average, the energy consumption of DRAM memory corresponds to 5% for sequential access pattern and to 7% for random access pattern of the total energy consumption. Therefore, for this work, an analysis considering only the package level is valid.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access

	cere			kernel			eins			dblp			
	bwt	psi	lcp	bwt	psi	lcp	bwt	psi	lcp	bwt	psi	lcp	sorted
plain	100.0	400.0	400.0	100.0	400.0	400.0	100.0	400.0	400.0	100.0	400.0	400.0	400.0
δ-code	140.5	424.6	194.5	134.1	424.6	268.4	137.9	424.6	276.4	135.7	424.6	122.2	440.6
γ-code	165.5	633.4	233.9	152.7	633.4	352.1	160.3	633.4	364.9	155.9	633.4	129.0	665.5
dac	87.5	337.5	160.0	100.0	337.5	228.0	100.0	337.5	213.8	87.5	337.5	94.0	350.0
fv	203.1	462.1	249.8	209.2	462.1	321.4	200.5	462.1	327.7	198.3	462.1	197.4	478.1
s9	102.9	403.1	193.4	100.0	403.1	349.2	102.7	403.1	352.7	101.8	403.1	94.9	403.1
rl	44.0	439.1	446.2	11.4	439.1	450.2	0.6	439.1	440.8	68.6	439.1	309.6	439.1
pfd	88.2	399.2	164.1	84.2	399.1	234.5	85.8	399.1	213.6	83.8	399.2	78.8	400.1
δ-code_zz	26.8	63.0	204.5	20.6	58.6	303.9	18.4	56.1	249.3	31.3	64.1	66.5	72.6
γ-code_zz	27.0	52.6	251.7	24.2	47.0	406.6	25.0	43.7	321.0	31.3	54.0	94.0	65.1
dac_zz	35.3	47.8	169.4	30.5	43.8	250.0	28.8	41.4	213.1	39.1	50.3	65.2	59.7
fv_zz	532.4	570.2	657.2	541.8	567.4	747.1	540.7	565.7	704.2	534.6	570.9	563.5	551.5
s9_zz	62.1	72.6	203.0	31.9	44.6	382.5	20.5	34.9	324.5	37.6	49.0	78.7	65.8
pfd_zz	25.9	38.9	169.1	17.4	31.1	253.8	13.8	26.5	216.0	27.2	39.2	65.5	54.8

TABLE 3: Space usage of the compressed vectors for the datasets of Table 2, in megabytes. Smallest values are underlined.

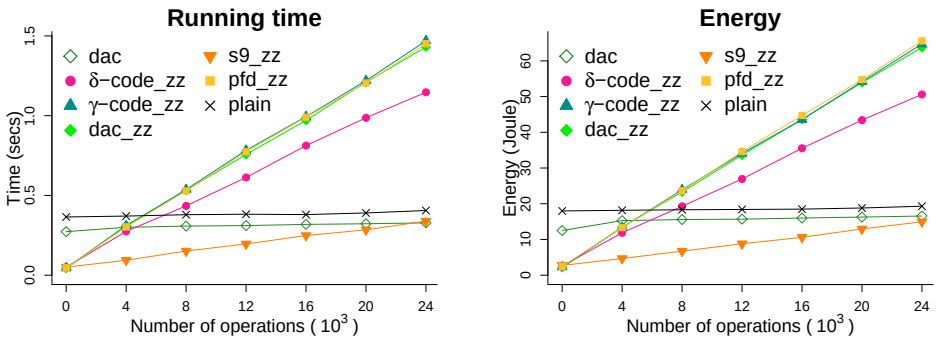


FIGURE 2: Running time and energy consumption of binary search

nor energy consumption. The most time- and energy-efficient approach was the CIV s9_zz, which up to 24,000 binary searches uses less time and energy than the vector in plain form. Thus, as a preliminary conclusion, CIVs represent an energy-efficient alternative for binary search when the number of binary searches executed over a vector is limited: less than 4,000 binary searches for δ-code_zz, γ-code_zz and dac_zz, and less than 24,000 for s9_zz. We provide a deeper discussion of the factors that affect the energy consumption of CIVs in the following paragraphs, when we study the sequential and random access patterns.

Figures 3-6 show several metrics for some of the datasets. We selected three representative cases to discuss our findings: For bwt, psi and lcp vectors, the results of the eins, kernel and dblp datasets are shown, respectively. We omitted experiments for some CIVs that use more space than the vectors in plain format, such as the case of fv for the bwt vector of the dataset cere, among others. To improve the readability of the figures, we only show the five CIVs with better energy consumption for each experiment.

As in Figure 2, when the number of operations is zero, the CIVs are loaded in memory, without accessing their values. Figures 3 and 4 show the results for sequential access pattern. For the bwt vector of eins, in Figure 3a, the structure RL is the most time-efficient up to 20 millions of sequential operations, and the most energy-efficient up to 25 millions of operations. Both the running time and the energy consumption depend directly on the number of instructions, CPU cycles, cache accesses to the different levels of the memory hierarchy, among other factors. The graph of the number of instructions in Figure 3a shows that the CIVs perform more instructions than the plain vector. In general, CIVs perform several internal operations in order to recover an element of the compacted vector, thus, increasing the number of machine instructions. However, the instructions performed by the CIVs need less CPU cycles to be completed than the instructions of the plain vector, as it is shown in the graph of CPU cycles per instruction of Figure 3a. When the number of operations is zero, the CPU cycles per instruction is high since the latency involved in the loading of the CIVs is also high. When the number of operations increases, the cost of loading the data structures is amortized, reducing the average number of cycles per instruction. The fact that CIVs need less CPU cycles per instruction than the plain vector, suggests that the memory accesses involved in some instructions are solved in lower levels of the memory hierarchy. It can be corroborated in the graphs of L1d and L3 cache loads of Figure 3a, where CIVs use intensively the L1d cache compared to the plain vector, meanwhile most of the CIVs have less accesses to the L3 cache in comparison

with the plain vector. The size of the cache memories and their set-associativity have an impact on the running time and the energy consumption of running applications. Cache memories with reduced size and small set-associativity are more time- and energy-efficient than larger memories [73]. Accordingly, the L1d cache memory of the machine used in our experiments is more time- and energy-efficient than the L3 cache memory, since the L1d cache has a size of 32 KB and is 8-way set-associative, meanwhile the L3 cache has a size of 20 MB and is 20-way set-associative. Unlike the cache behavior of the plain vector, CIVs exhibit a higher rate of memory accesses to L1d cache than to L3 cache, which may explain that for more than 25 millions of sequential operations, the plain vector is most time- and energy-efficient. This analysis is also valid for the `psi` vector of the dataset `kernel` (Figure 3b) and the `lcp` vector of the dataset `dblp` (Figure 4a). For the `psi` vector of `kernel`, the most time- and energy-efficient CIV is `s9-zz`, and for the `lcp` vector of `dblp` is `s9`.

From the results we observe that there is a strong correlation between the energy consumption to load the CIVs in memory and the size of the CIVs. However, when the operations are performed, there is not a clear correlation between energy consumption and size. For instance, for the `lcp` vector of the `dblp` dataset, the CIV `s9` is the eighth smallest compact vector and the most efficient from an energy point of view. A similar situation occurs with the `psi` and `bwt` vectors.

Regarding the relation between running time and energy consumption, there exists a strong correlation: the faster the CIV, the more energy efficient. Nevertheless, in some particular situations, the relation is broken. Such situations are marked with vertical lines in Figures 3 and 4. For instance, in Figure 4a, in the range of 26 to 28 millions of operations, the plain vector is faster than the data structure `s9-zz`, but `s9-zz` is more energy-efficient. We leave as future work a more complete study of such particular situations, in order to design energy-efficient CIVs.

Figures 5 and 6 show the random access pattern. As expected, the random access pattern takes more time and energy than the sequential access pattern, since the former does not have the spatial data locality of the latter. Moreover, for some of the techniques, each time that an element is accessed randomly, several values must be decoded up to the desired element, starting from the closest sampled value. The previous explanation of the impact of memory accesses, instructions and CPU cycles per instruction on the running time and energy consumption of the CIVs remains valid for the random access scenario. Among all the tested CIVs, `dac` and `pfd` exhibit a remarkable improvement compared to their behavior in sequential access scenario. In particular, both `dac` and `pfd` improve their behavior in the number of instructions and the number of L1d cache accesses, thus, reducing their relative energy consumption.

From the energy point of view, the compact vectors storing differences are more energy-efficient for sequential access pattern, performing worse for the random access pattern. This is an expected behaviour, as retrieving the original value at a random position of a CIV storing differences requires the use of sampling and the decompression of several positions instead of just one, thus, nullifying the benefits of its fast direct access.

In summary, our experimental study provides evidence than the CIVs are a valid energy-efficient alternative to manipulate integer vectors. For an increasing number of accesses to the elements of a vector, the CIVs reduce their energy-efficiency. Therefore, the number of operations over the CIVs is a factor that must be considered. For future work, this finding will allow us to define favorable scenarios for the usage of CIVs in the design of energy-efficient algorithms.

Our experiments on access patterns revealed that such patterns have an important impact in the energy behavior of the CIVs. If for some applications we could know in advance the most common access patterns, we could select the most suitable CIV for such application, potentially reducing its overall energy consumption. Otherwise, our experiments suggest that CIVs such as `dac` and `s9` are a better alternative, since their energy behavior is less affected with the access patterns. Such CIVs, `dac` and `s9`, could be a good starting point to design energy-efficient compact vectors for general scenarios.

V. CONCLUSIONS AND FUTURE WORK

In this work we studied the energy behavior of several compact integer vectors for typical operations. We performed experiments with different datasets, measuring metrics such as time, energy consumption, number of instructions, memory transfers, among others. Our results suggest that compact integer vectors offer a good alternative for the energy-efficient manipulation of vectors. This work is the first one providing evidence that compact data structures may be considered in the design of energy-efficient algorithms.

As future work, we plan to develop a more low-level framework to better characterize all the factors that impact on the energy consumption. Not only the computer architecture, memory hierarchy and memory access patterns may be taken into account, but other factors, such as temperature inside and outside the computation device, the complexity of the instructions used, among others. This will allow us to propose new energy models that will make possible the design of algorithms that consider energy during their design step.

ACKNOWLEDGMENT

The authors thank Gonzalo Navarro for suggesting this line of research some time ago, and finally creating the link between the authors that made possible this collaboration.

REFERENCES

[1] G. Navarro, Compact Data Structures – A practical approach. Cambridge University Press, 2016, iSBN 978-1-107-15238-0. 570 pages.

[2] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short dna sequences to the human

genome," Genome Biology, vol. 10, no. 3, p. R25, 2009. [Online]. Available: https://doi.org/10.1186/gb-2009-10-3-r25

[3] H. Li and R. Durbin, "Fast and accurate short read alignment with BurrowsâĂŞWheeler transform," Bioinformatics, vol. 25, no. 14, pp. 1754–1760, 2009. [Online]. Available: https://doi.org/10.1093/bioinformatics/btp324

[4] J. S. Culpepper, G. Navarro, S. J. Puglisi, and A. Turpin, "Top-k ranked document search in general text databases," in ESA. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 194–205. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882123.1882145

[5] R. Konow and G. Navarro, "Dual-sorted inverted lists in practice," in Procs. of the 19th International Symposium on String Processing and Information Retrieval (SPIRE), LNCS 7608, 2012, pp. 295–306.

[6] P. Ferragina, "Data structures: Time, i/os, entropy, joules!" in ESA. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–16.

[7] J. Veiga, J. Enes, R. R. ExpÃşsito, and J. TouriÃśo, "Bdev 3.0: Energy efficiency and microarchitectural characterization of big data processing frameworks," Future Generation Computer Systems, vol. 86, pp. 565 – 581, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17325360

[8] G. Pinto and F. Castor, "Energy efficiency: A new concern for application software developers," Commun. ACM, vol. 60, no. 12, pp. 68–75, 2017. [Online]. Available: http://doi.acm.org/10.1145/3154384

[9] C. Calero and M. Piattini, "Puzzling out software sustainability," Sustainable Computing: Informatics and Systems, vol. 16, pp. 117 – 124, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210537916301676

[10] W. Lang, R. Kandhan, and J. M. Patel, "Rethinking query processing for energy efficiency: Slowing down to win the race," IEEE Data Eng. Bull., vol. 34, pp. 12–23, 2011.

[11] C. Bunse, H. Höpfner, S. Klingert, E. Mansour, and S. Roychoudhury, "Energy aware database management," in Energy-Efficient Data Centers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 40–53.

[12] K. Kant, "Toward a science of power management," Computer, vol. 42, no. 9, pp. 99–101, 2009.

[13] P. Ranganathan, "Recipe for efficiency: Principles of power-aware computing," Commun. ACM, vol. 53, no. 4, pp. 60–67, 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721673

[14] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in SenSys. New York, NY, USA: ACM, 2006, pp. 265–278. [Online]. Available: http://doi.acm.org/10.1145/1182807.1182834

[15] P. Larsson, Energy-Efficient Software Guidelines, Intel, 2008.

[16] K. Roy and M. C. Johnson, Software Design for Low Power. Boston, MA: Springer US, 1997, pp. 433–460.

[17] R. Ayoub, K. R. Indukuri, and T. S. Rosing, "Energy efficient proactive thermal management in memory subsystem," in ISLPED, 2010, pp. 195–200.

[18] M. Chen, S. Mao, and Y. Liu, "Big Data: A Survey," Mobile Networks and Applications, vol. 19, no. 2, pp. 171–209, 2014.

[19] D. Salomon, A Concise Introduction to Data Compression. Springer, 2008.

[20] A. Turpin and A. Moffat, "Fast file search using text compression," in Procs. of the 20th Australasian Computer Science Conference (ACSC), 1997, pp. 1–8.

[21] E. Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates, "Fast and flexible word searching on compressed text," ACM Transactions on Information Systems, vol. 18, no. 2, pp. 113–139, 2000.

[22] N. R. Brisaboa, A. Fariña, G. Navarro, and J. R. Paramá, "Lightweight natural language text compression," Information Retrieval, vol. 10, pp. 1–33, 2007.

[23] P. Weiner, "Linear pattern matching algorithm," in Procs. of the 14th Annual IEEE Symposium on Switching and Automata Theory, 1973, pp. 1–11.

[24] G. Jacobson, "Succinct Static Data Structures," Ph.D. dissertation, Carnegie-Mellon, 1989.

[25] P. Ferragina and G. Manzini, "Indexing compressed texts," Journal of the ACM, vol. 52, pp. 552–581, 2005.

[26] A. Gupta, W.-K. Hon, R. Shah, and J. S. Vitter, "Compressed data structures: Dictionaries and data-aware measures," Theoretical Computer Science, vol. 387, pp. 313–331, 2007.

[27] R. Raman, "Encoding data structures," in Procs. of the 9th International Workshop on Algorithms and Computation (WALCOM), LNCS 8973, 2015, pp. 1–7.

[28] G. Jacobson, "Space-efficient static trees and graphs," in Proc. of the 30th Annual Symposium on Foundations of Computer Science (FOCS), ser. SFCS '89, 1989, pp. 549–554.

[29] R. Raman, V. Raman, and S. S. Rao, "Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets," ACM Transactions on Algorithms, vol. 3, p. article 43, 2007.

[30] P. Ferragina and R. Venturini, "A simple storage scheme for strings achieving entropy bounds," Theoret. Comput. Sci, vol. 372, no. 1, pp. 115 – 121, 2007.

[31] D. Okanohara and K. Sadakane, "Practical entropy-compressed rank/select dictionary," in Proceedings of the Meeting on Algorithm Engineering & Expermiments (ALENEX), 2007, pp. 60–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=2791188.2791194

[32] F. Claude and G. Navarro, "Practical rank/select queries over arbitrary sequences," in Procs. of the 15th International Symposium on String Processing and Information Retrieval (SPIRE), LNCS 5280, 2008, pp. 176–187.

[33] S. Gog and M. Petri, "Optimized succinct data structures for massive data," Software Practice and Experience, vol. 44, pp. 1287–1314, 2014.

[34] J. I. Munro, R. Raman, V. Raman, and S. S. Rao, "Succinct representations of permutations and functions," Theoretical Computer Science, vol. 438, pp. 74–88, 2012.

[35] J. Barbay and G. Navarro, "On compressing permutations and adaptive sorting," Theoretical Computer Science, vol. 513, pp. 109–123, 2013.

[36] R. Grossi, A. Gupta, and J. S. Vitter, "High-Order Entropy-Compressed Text Indexes," in Proc. of the 14th annual ACM-SIAM symposium on Discrete algorithms (SODA), vol. 2068, 2003, pp. 841–850. [Online]. Available: http://dl.acm.org/citation.cfm?id=644108.644250

[37] V. Mäkinen and G. Navarro, "Succinct suffix arrays based on run-length encoding," Nordic Journal of Computing, vol. 12, pp. 40–66, 2005.

[38] J. Barbay, F. Claude, T. Gagie, G. Navarro, and Y. Nekrich, "Efficient fully-compressed sequence representations," Algorithmica, vol. 69, pp. 232–268, 2014.

[39] D. Belazzougui and G. Navarro, "Optimal lower and upper bounds for representing sequences," ACM Transactions on Algorithms, vol. 11, p. article 31, 2015.

[40] R. Raman and S. S. Rao, "Succinct representations of ordinal trees," Space-Efficient Data Structures, Streams, and Algorithms, LNCS 8066, pp. 319–332, 2013.

[41] G. Navarro and K. Sadakane, "Fully-functional static and dynamic succinct trees," ACM Transactions on Algorithms, vol. 10, p. article 16, 2014.

[42] A. Farzan and J. I. Munro, "Succinct encoding of arbitrary graphs," Theoretical Computer Science, vol. 513, pp. 38–52, 2013.

[43] F. Li, Q. Zhang, T. Gu, and R. Dong, "Optimal representation for web and social network graphs based on K^2 -tree," IEEE Access, vol. 7, pp. 52 945–52 954, 2019.

[44] T. M. Chan, K. G. Larsen, and M. Patrascu, "Orthogonal range searching on the ram, revisited," in Procs. of the 27th ACM Symposium on Computational Geometry (SoCG), 2011, pp. 1–10.

[45] J. Barbay, F. Claude, and G. Navarro, "Compact binary relation representations with rich functionality," Information and Computation, vol. 232, pp. 19–37, 2013.

[46] N. R. Brisaboa, S. Ladra, and G. Navarro, "Compact representation of web graphs with extended functionality," Information Systems, vol. 39, pp. 152–174, 2014.

[47] D. Arroyuelo, S. González, and M. Oyarzún, "Compressed self-indices supporting conjunctive queries on document collections," in Procs. of the 17th String Processing and Information Retrieval (SPIRE), E. Chavez and S. Lonardi, Eds., 2010, pp. 43–54.

[48] G. de Bernardo, S. Álvarez-García, N. R. Brisaboa, G. Navarro, and Ó. Pedreira, "Compact querieable representations of raster data," in Procs. of the 20th International Symposium on String Processing and Information Retrieval (SPIRE), LNCS 8214, 2013, pp. 96–10.

[49] S. Ladra, J. R. Paramá, and F. Silva-Coira, "Scalable and queryable compressed storage structure for raster data," Information Systems, vol. 72, pp. 179–204, 2017.

[50] M. M. Waldrop, "The chips are down for moore's law," Nature, vol. 530, no. 7589, pp. 144–147, 2016. [Online]. Available: https://doi.org/10.1038/530144a

[51] R. Arshad, S. Zahoor, M. A. Shah, A. Wahid, and H. Yu, "Green iot: An investigation on energy saving practices for 2020 and beyond," IEEE Access, vol. 5, pp. 15 667–15 681, 2017.

[52] S. Albers, "Energy-efficient algorithms," Commun. ACM, vol. 53, no. 5, pp. 86–96, 2010. [Online]. Available: http://doi.acm.org/10.1145/1735223.1735245

[53] E. D. Demaine, J. Lynch, G. J. Mirano, and N. Tyagi, "Energy-efficient algorithms," in ITCS. New York, NY, USA: ACM, 2016, pp. 321–332. [Online]. Available: http://doi.acm.org/10.1145/2840728.2840756

[54] R. Jain, D. Molnar, and Z. Ramzan, "Towards understanding algorithmic factors affecting energy consumption: Switching complexity, randomness, and preliminary experiments," in DIALM-POMC. New York, NY, USA: ACM, 2005, pp. 70–79. [Online]. Available: http://doi.acm.org/10.1145/1080810.1080823

[55] S. Roy, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in ITCS. New York, NY, USA: ACM, 2013, pp. 283–304. [Online]. Available: http://doi.acm.org/10.1145/2422436.2422470

[56] N. Chatterjee, S. Biswas, and P. P. Das, Software Energy Estimation to Improve Power Efficiency of Embedded System. Singapore: Springer Singapore, 2017, pp. 79–92.

[57] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," in Procs. of the 1997 International Symposium on Low Power Electronics and Design (ISLPED), 1997, pp. 202–207.

[58] G. Navarro, "Wavelet trees for all," Journal of Discrete Algorithms, vol. 25, pp. 2–20, 2014.

[59] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Digital SRC Research Report. Issue 124, 18 pages, Tech. Rep., 1994.

[60] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in Procs. of the 41st Annual Symposium on Foundations of Computer Science (FOCS), 2000, pp. 390–398.

[61] J. T. Simpson and R. Durbin, "Efficient construction of an assembly string graph using the FM-index," Bioinformatics, vol. 26, no. 12, pp. i367–i373, 06 2010. [Online]. Available: https://doi.org/10.1093/bioinformatics/btq217

[62] P. Elias, "Universal codeword sets and representations of the integers," IEEE Trans. Inform. Theory, vol. 21, no. 2, pp. 194–203, 1975.

[63] N. R. Brisaboa, S. Ladra, and G. Navarro, "Dacs: Bringing direct access to variable-length codes," Inf. Process. Manage., vol. 49, no. 1, pp. 392–404, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2012.08.003

[64] V. N. Anh and A. Moffat, "Inverted index compression using word-aligned binary codes," Information Retrieval, vol. 8, no. 1, pp. 151–166, 2005. [Online]. Available: https://doi.org/10.1023/B:INRT.0000048490.99518.5c

[65] M. Zukowski, S. Heman, N. Nes, and P. Boncz, "Super-scalar ram-cpu cache compression," in ICDE, 2006, pp. 59–59.

[66] K. Sadakane, "New text indexing functionalities of the compressed suffix arrays," J. Algorithms, vol. 48, no. 2, pp. 294 – 313, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677403000877

[67] I. Witten, A. Moffat, and T. Bell, Managing Gigabytes, 2nd ed. New York: Morgan Kaufmann Publishers, 1999.

[68] D. Solomon, Variable-length codes for data compression. Springer-Verlag, 2007.

[69] M. Kambadur and M. A. Kim, "An experimental survey of energy management across the stack," in OOPSLA. New York, NY, USA: ACM, 2014, pp. 329–344. [Online]. Available: http://doi.acm.org/10.1145/2660193.2660196

[70] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2, 2016, chapter 14.9.

[71] D. Hackenberg, T. Ilsche, R. Schone, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," in ISPASS. Los Alamitos, CA, USA: IEEE Computer Society, 2013, pp. 194–204. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISPASS.2013.6557170

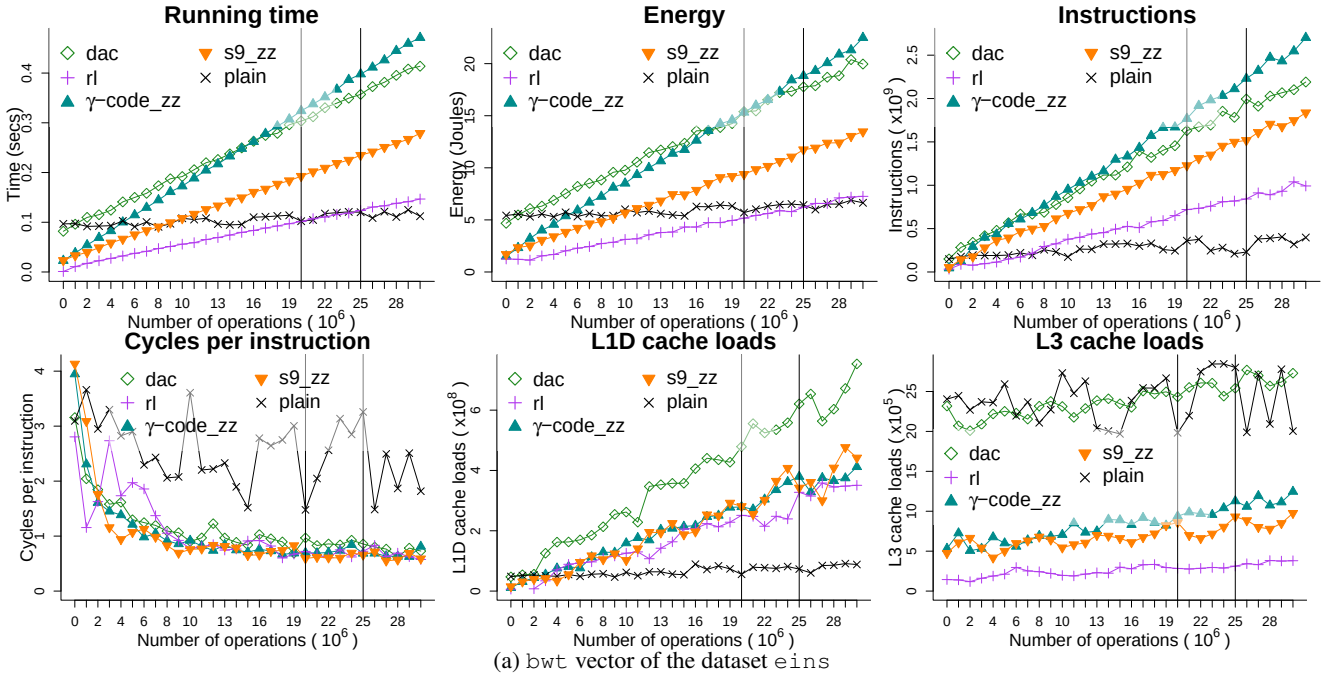
[72] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," ACM Trans. Model. Perform. Eval. Comput. Syst., vol. 3, no. 2, pp. 9:1–9:26, Mar. 2018. [Online]. Available: http://doi.acm.org/10.1145/3177754

[73] J. L. Hennessy and D. A. Patterson, Computer Architecture, Sixth Edition: A Quantitative Approach, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

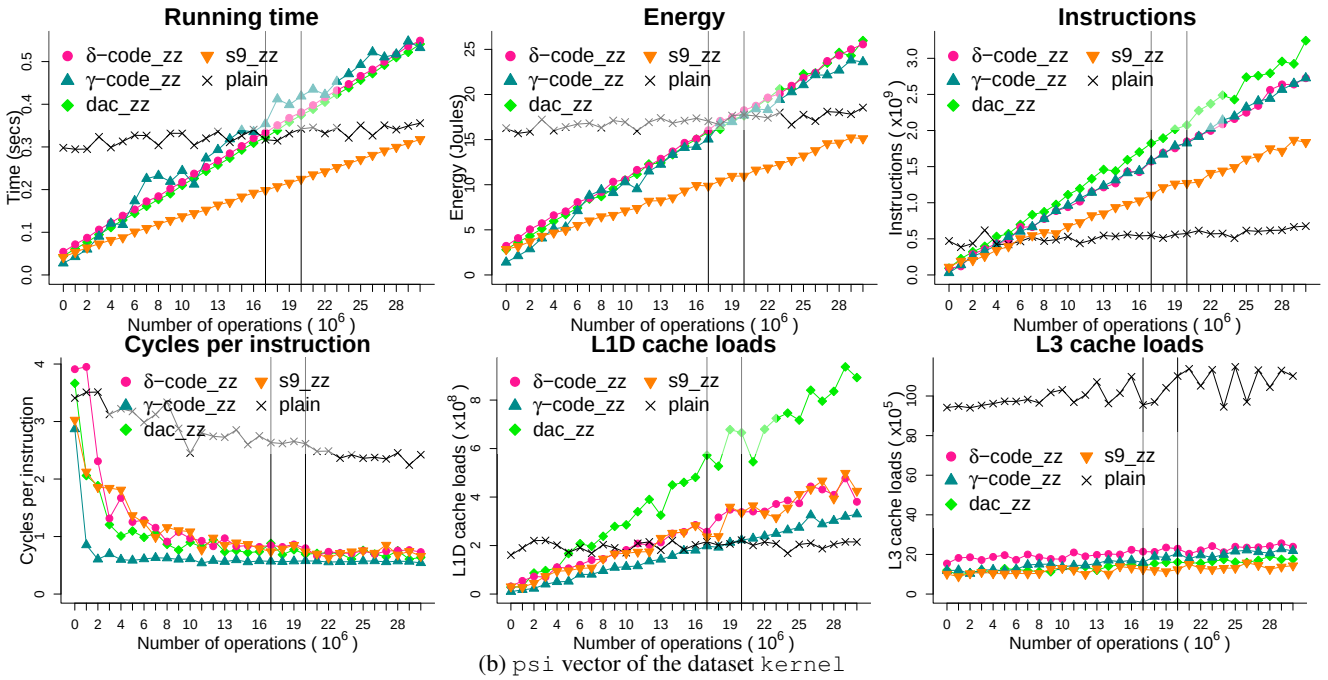
JOSÉ FUENTES-SEPÚLVEDA is a Postdoctoral Researcher at University of Chile under the supervision of Gonzalo Navarro, PhD. He received his degree of Doctor in Computer Science from the University of Concepción in 2016. His research interests include the design and analysis of algorithms and data structures, data compression and parallel algorithms.

SUSANA LADRA is Associate Professor at the University of A Coruña, where she obtained her degree in Computer Science Engineering in 2007 and her Ph.D. degree in Computer Science in 2011 at the same university. She also received her Bachelor in Mathematics from the National Distance Education University (UNED) in 2014. Her fields of interests include the design and analysis of algorithms and data structures, data compression and data mining in the fields of information retrieval and bioinformatics. She has published more than 40 papers in various international journals and conferences and is principal investigator of several national and international research projects.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access



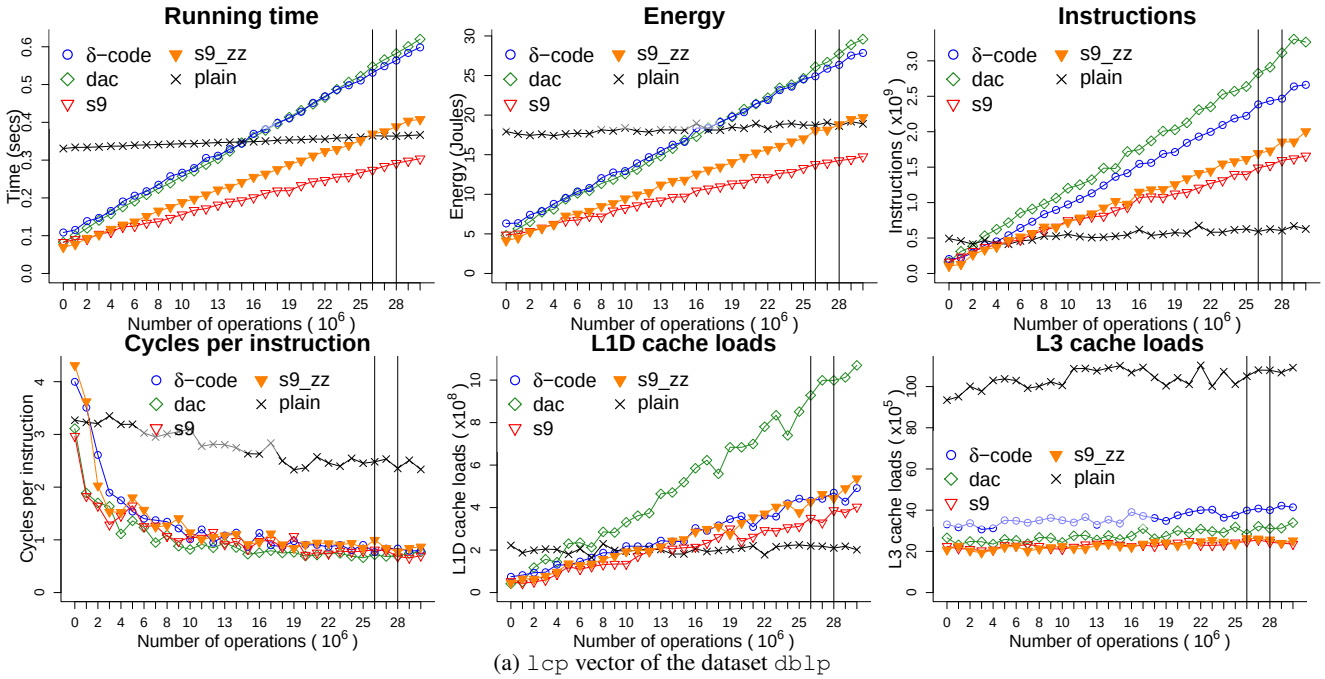
(a) `bwt` vector of the dataset `eins`



(b) `psi` vector of the dataset `kernel`

FIGURE 3: Experiments with sequential access pattern (Part I)

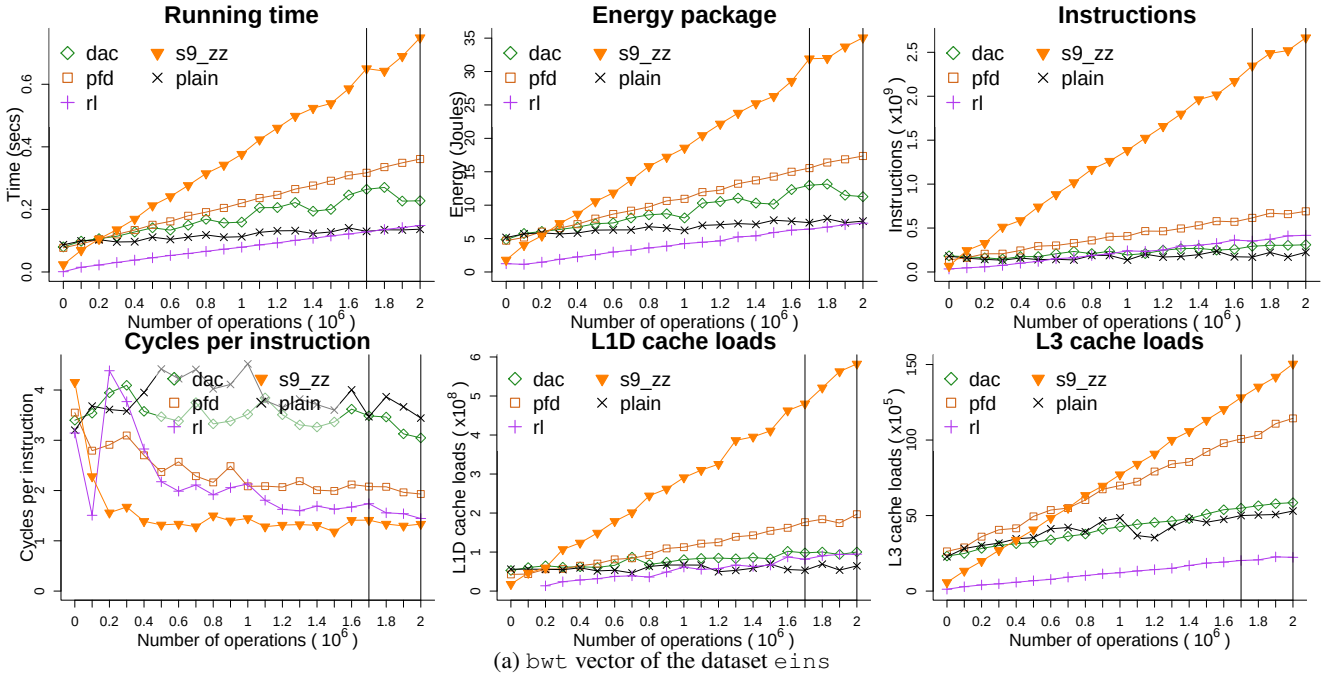
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access



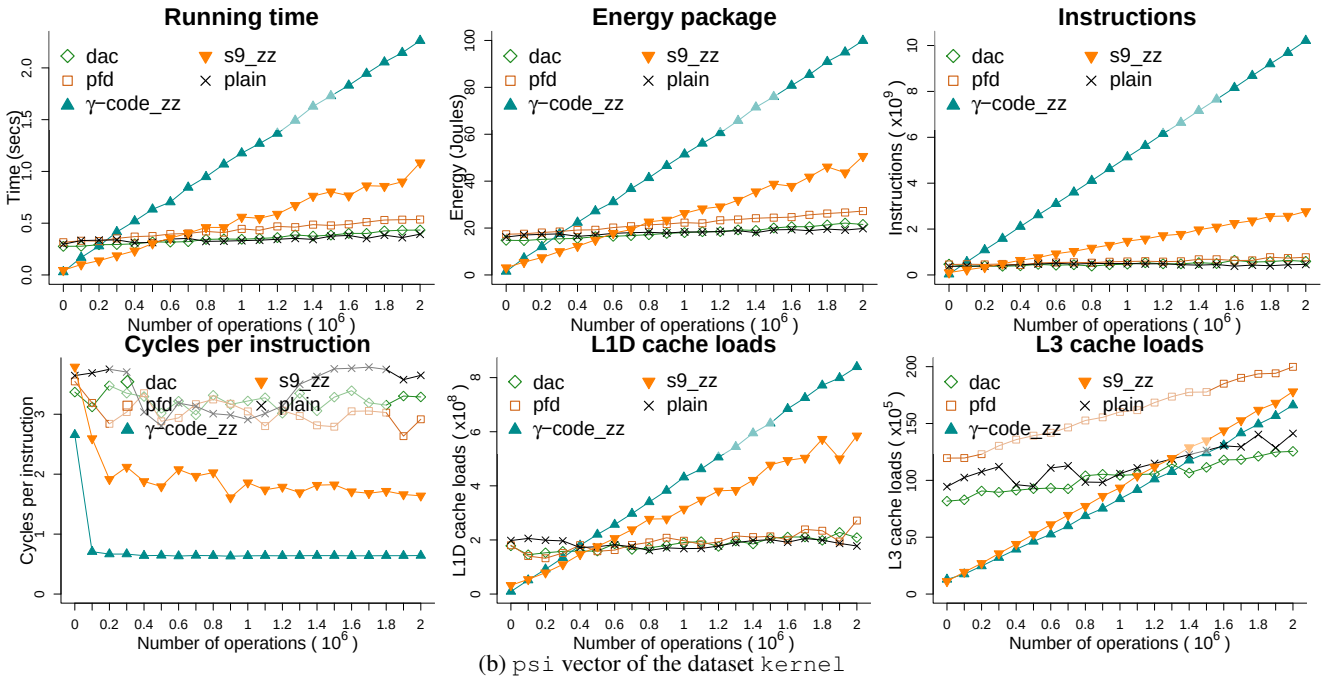
(a) `lcp` vector of the dataset `dblp`

FIGURE 4: Experiments with sequential access pattern (Part II)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access



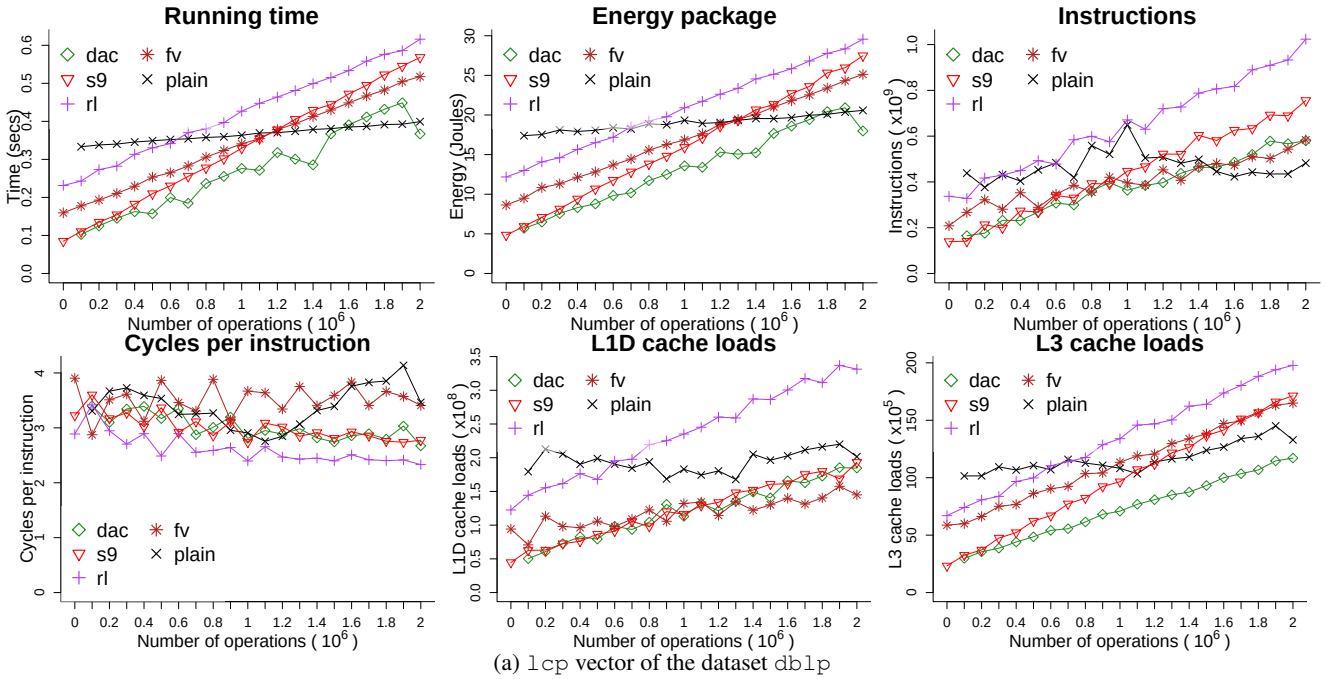
(a) `bwt` vector of the dataset `eins`



(b) `psi` vector of the dataset `kernel`

FIGURE 5: Experiments with random access pattern (Part I)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2949655, IEEE Access



(a) `lcp` vector of the dataset `dblp`

FIGURE 6: Experiments with random access pattern (Part II)