Variable Word Length Word-Aligned Hybrid Compression

Florian Grieskamp
G DATA CyberDefense AG
florian.grieskamp@gdata.de

Roland Kühn
TU Dortmund University
roland.kuehn@cs.tu-dortmund.de

Jens Teubner
TU Dortmund University
jens.teubner@cs.tu-dortmund.de

ABSTRACT

Efficient and fast compression plays a crucial role in processing large bitmap indices in modern database management systems.

The Word-Aligned Hybrid (WAH) compression is a prominent example of a lightweight compression scheme for bitmap indices. WAH is inspired by run-length encoding, but also considers the *word size* of the underlying architecture. The latter is a compromise toward commodity CPUs, where operations below the word granularity perform poorly. With the emergence of novel hardware classes, such compromises may no longer be appropriate. Field-programmable gate arrays (FPGAs) do not even have any meaningful "word size."

In this work, we reconsider strategies for bitmap compression in the light of modern hardware architectures. Rather than tuning compression toward a fixed word size, we propose to tune the word size toward optimal compression. The resulting compression scheme, *Variable Word Length Word-Aligned Hybrid (VWLWAH)*, improves compression rates by almost 75 %, while maintaining line rate performance on FPGAs.

KEYWORDS

leightweight compression, FPGA, data processing on FPGAs, Word Aligned Hybrid compression

ACM Reference Format:
Florian Grieskamp, Roland Kühn, and Jens Teubner. 2020. Variable Word Length Word-Aligned Hybrid Compression. In *Proceedings of DaMoN '20: 16th International Workshop on Data Management on Modern Hardware (DaMoN '20).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Bitmap indices are an important building block in the design of large-scale data warehouses. They appeal with high flexibility, *e.g.*, when used as *join indexes* in star schema settings.

To combat the prohibitive space overhead that would result from naive bitmap index designs, systems use *lightweight compression* schemes, which aim to provide decent compression rates while maintaining *high throughput*. The most well-known incarnation of this idea is the *Word-Aligned Hybrid (WAH)* scheme of Wu *et al.* [17].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DaMoN '20, June 15, 2020, Portland, OR
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

WAH is based on *run-length encoding* for compression. However, rather than treating bit vectors at a bit or byte level for compression, WAH operates at the granularity of *machine words* (usually 32 bits). This *word alignment* is a compromise toward the characteristics of commodity CPUs, where operations on data below the intrinsic machine word size is complicated and slow.

Since the inception of WAH, the hardware landscape has changed significantly. Commodity CPUs—which is what WAH was designed for—have become just one piece in a mix of heterogeneous processing components that may further include graphics processors (GPUs), field-programmable gate arrays (FPGAs), or other processor classes. In such a mix, it is all but clear whether "word alignment" even has a sensible meaning. Different processor classes may favor different word sizes; and some—FPGAs in particular—can deal equally well with *any* word size.

In this work, we aim to understand what the changing hardware landscape means to the design of bitmap indices and their compression schemes. We specifically look at FPGAs, which not only fit well with the typical uses of bitmap indexes but also become increasingly attractive with tighter integrations into mainstream platforms (*e.g.*, in the form of Intel's *HARP* platform [5]). We devise *Variable Word Length Word-Aligned Hybrid (VWLWAH)* as a more flexible variant of WAH that can leverage the opportunities of heterogeneous hardware platforms.

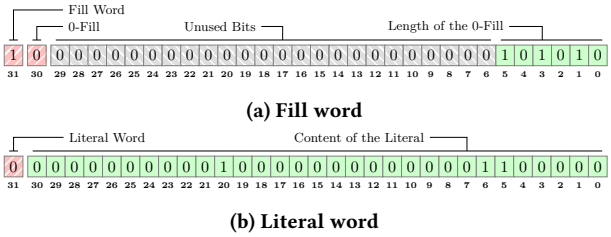
The remainder of the paper is structured as follows. We summarize the basic idea of *Word-Aligned Hybrid* compression and some optimizations of other researchers in Section 2, followed by a description of our idea *VWLWAH* in Section 3, where we also highlight the similarities and differences of our implementation to existing approaches. The theoretical and experimental results regarding space complexity of *VWLWAH* can be found in Section 4. Section 5 contains a description of the library for FPGA-based database acceleration that we are developing at TU Dortmund University. In Section 6, we examine the runtime behaviour of our approach in theory and with an experimental implementation on real hardware. The paper ends with a conclusion and an outlook to possible future enhancements.

2 BACKGROUND

Word-Aligned Hybrid (WAH) was proposed as a compression scheme for bitmap indices by Wu *et al.* [17] and uses a variant of run-length encoding to compress sequences of identical bits within a bit vector.

2.1 WAH Compression

Figure 1 illustrates the WAH compression mechanism, assuming a word size of $w = 32$. In the compressed representation, each word (of size w) can either be a *fill word* (Figure 1a) or a *literal word* (Figure 1b). The role of each word is encoded in its most significant bit (MSB, leftmost bits in Figure 1).



(a) Fill word

(b) Literal word

Figure 1: Layout of a fill and literal word. The fill word represents 42 0-fills with a block size of 31 bit each, which results in a sequence of 1302 consecutive 0-bits.

If the word is a *literal word*, the remaining $w-1$ bits are taken literally from the uncompressed data. *Fill words* encode sequences of all-0 or all-1 bits, and the bit at position $w-2$ encodes whether the fill encodes a sequence of 0s or 1s. In the latter case, the remaining $w-2$ bits are interpreted as a binary encoded integer number, encoding the *length* of the fill (measured in multiples of $w-1$; in Figure 1a, the fill word would encode $(101010_2 = 42) \times 31 = 1302$ 0-bits).

The *word size* w in WAH is chosen based on the characteristics of the underlying machine. All systems that we are aware of choose $w = 32$. Smaller word sizes will underutilize the capabilities of modern CPUs, which tend to favor large word and register sizes. Larger word sizes will deteriorate compression ratios. As can be seen already in the illustration in Figure 1, even long runs of 0s will use only a fraction of the bits that would be available to encode the run length (in Figure 1a, bits 6–29 are all 0, and we colored them in gray for this reason).

A beauty of the WAH scheme is that important operations on bit vectors can be performed *without* decompressing the operand vectors. This includes all the usual bitwise Boolean operations ($\neg$, $\vee$, $\wedge$), and systems derive much of their performance from this potential.

2.2 WAH Variants

The tension between the optimization goals *CPU efficiency* ($\rightsquigarrow$ large word size) and *compression quality* ($\rightsquigarrow$ small word size) has inspired a number of follow-up articles that propose modifications of the original WAH scheme. Their common theme is to make better use of the "unused bits" as we illustrated them in Figure 1a.

Enhanced Word Aligned Hybrid (EWAH) [10] uses the full word size of 32 or 64 bit for literal words, thus a literal word can not be distinguished by the MSB anymore. This implies that every compressed sequence has to start with a special word, called *marker word*, where the MSB specifies whether it is a 0-fill or a 1-fill and the subsequent 16 bits determine the length of the fill. The following bits specify the number of literal words which follow the 0- or 1-fills. That means in return that 17 bits may be wasted, if the first block is a literal word.

Partitioned Word Aligned Hybrid (PWAH) [14] tries to reduce the disadvantage of a fixed word length by partitioning the word in 2, 4 or 8 partitions, where each partition can be a valid fill or literal word. Since the number of representable blocks in a fill word decreases with smaller partitions, two or more fill words can be simply concatenated to represent the number of fills.

In contrast to *PWAH, Variable Length Compression (VLC)* [6] offers more degrees of freedom in the choice of different partition sizes. But in order to maintain the word alignment, words are padded with 0-bits if necessary.

The work of Guzun *et al.* [9] showed that different block sizes may be beneficial for different bit vectors. This led to a compression scheme called *Variable Aligned Length Word-Aligned Hybrid (VAL-WAH)*, which is able to execute bitwise operations on compressed data even if different bit vectors have been compressed with different block sizes. The idea of this is approach is that each block size is restricted to a multiple of a power of two, so that word alignment is ensured.

In addition to the concepts that were discussed above, there are a few more ideas that try to improve the original WAH scheme or even try to improve already existing optimizations to *WAH*. Some of these proposals try to extend the compression in such a way that they try to detect and handle certain patterns that would not compress well under the original *WAH* scheme [2–4, 7, 13]. Other papers discuss the possibility of meta-compression, *i.e.*, to compress the output of already WAH-compressed data[8, 15, 18].

2.3 FPGAs for Database Tasks

From the hardware technology side, FPGAs have emerged a few years ago as a highly promising technology for database acceleration. Early proposals have pioneered the field [12], to be picked up by system makers to develop full-stack database solutions with FPGA accelerations. Examples of this include *Ibex*, which uses FPGAs to pre-filter and pre-process data close to storage [16]; Lisa *et al.* [11] used FPGAs to accelerate scans in in-memory databases; Becher *et al.* [1] leveraged dynamic reconfiguration on FPGAs for full SQL support.

These systems have in common that they use the FPGA to provide processing capabilities along the system's *data path*. This matches particularly well the characteristics of FPGAs, which favor high-throughput, stream-oriented processing modes.

3 VARIABLE WORD LENGTH WORD-ALIGNED HYBRID

WAH and all of its above-mentioned variants stick to the idea of matching the *machine word size* of the underlying hardware, usually 32 or even 64. While such values make sense in the light of commodity CPU hardware, from the perspective of the achievable *compression rate*, smaller values would be much more desirable.

3.1 Word Widths in WAH

To illustrate the effect of the configured word size on the achievable compression rate, we generated bit vectors of size 100 MB and compressed them using WAH using different word width configurations. Figure 2 visualizes the size of the compressed representations that we observed for $w \in \{4, 8, 16, 32\}$.[1]

[1] In the graph, "1-density" refers to the characteristics of the input bit vectors. In a relational database setting, a bitmap index on a relation R that indexes an attribute a with α distinct values ("attribute cardinality") will consist of α bit vectors $\mathbf{x}_1, \ldots \mathbf{x}_{|R|}$ (where $|R|$ is the number of tuples in R). Since every row in R will carry exactly

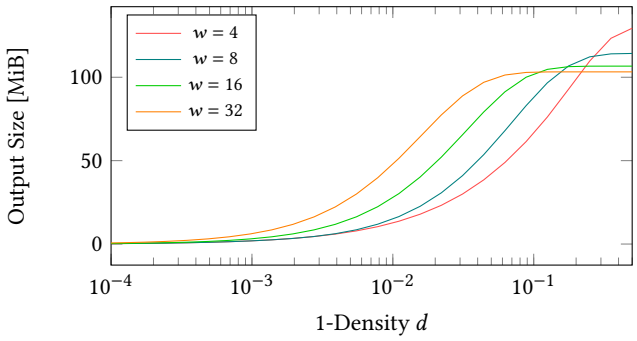


Figure 2: Size of the compressed vectors with varying density for four different word sizes

The left end of the graph illustrates the situation where the input bit vector is extremely sparse. In that case, WAH can efficiently compress very long runs of 0s away, resulting in a very small output data size, irrespective of the word width.

The right end is the other extreme. 0s and 1s mix equally in the input data set, with no runs of 0s or 1s that could be compressed away. This is the situation that would arise when very low-cardinality attributes were indexed with a bitmap index. In that case, WAH compression results in a slight *increase* of the data set size. Intuitively, all input data will be copied into *literal words*, and a 0-bit is attached to each $w-1$-chunk to mark the literal word (cf. Figure 1b). This "compression" penalty, therefore, increases for smaller values of w.

Interesting for practical applications is the range in-between. As we can see in the graph, for moderate 1-densities the choice of the word width can have a significant effect on the size of the compressed data. For some configurations, $w = 4$ compresses up to four times better than the standard value of $w = 32$!

3.2 VWLWAH

In the light of these measurements, we propose to make word width a tuning knob that can be chosen based on data set characteristics. In most situations, values much smaller than 32 will significantly improve the compression ratio of WAH (and we will look at performance in the Section 5).

At some point, very small values for w will have a negative effect on compression ratios, too. Smaller word sizes will also limit the length of fills that can be encoded, counteracting the idea of run-length idea. To remedy the situation, we pick up ideas that were introduced Guzun *et al.* [9] to devise our *Variable Word Length Word-Aligned Hybrid (VWLWAH)* compression scheme.

Figure 3 illustrates how VWLWAH addresses the problem of limited fill ranges in standard WAH for small values of w. As a convention in VWLWAH, successive fill words (here: the second and third words for $w = 7$) are interpreted by *merging* their fill width encodings. In the illustrated example, this enables the encoding of up to 2^{5+5} blocks of 0s with just two encoded 7-bit words.

one a value, all bit vectors together will have exactly $|R|$ bits set to 1. Assuming a uniform data distribution in R, the probability of hitting a 1-bit at an arbitrary bit vector position, the 1-density, will therefore be inversely proportional to the attribute cardinality α.

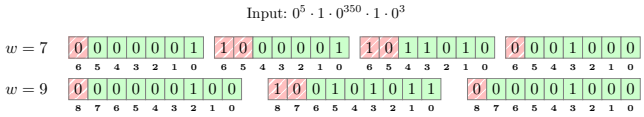


Figure 3: Representation of a compressed bitvector with different word sizes. For a word size of $w = 7$ to fill words are concatented to represent a long fill

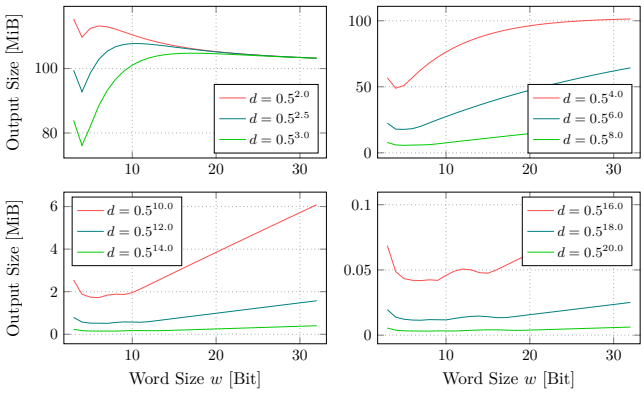


Figure 4: Size of the compressed bitvectors with varying word sizes for different densities.

4 VWLWAH AND COMPRESSION RATIOS

To assess how VWLWAH can indeed improve the compression characteristics of WAH, we used two generators that were proposed in existing studies of the field to produce synthetic datasets (cf. [7, 13, 17]. Since they contain relatively much entropy, *uniformly distributed bitmaps* represent the worst case situation from the perspective of compressibility. *Clustered bitmaps* aim to express situations as they are frequent in data warehouses, where seasonal or other effects let 1-bits accumulate in bitmap indices.

4.1 Uniformly distributed bitmaps

First, we created a model to estimate the number of bits that are necessary for bit vector compression with VWLWAH. With this model, we checked whether the variation of the word size could be beneficial. For uniformly distributed input data, there were three influential factors to the model: The selectivity, that determines the density of the 1-bits in a bit vector `BI`.

$$P(A = a) = P(\texttt{BI}(A = a)_i = 1) = d$$

The additional parameters of the model are the word size w and the number of bits N.

Evaluation. Figure 4 illustrates the effect of the word size on the compressed data size for a fixed set of 1-densities.

As we had already seen earlier (Figure 2), high 1-densities can benefit from a large word size. Our experiments showed that up to a density of $d = 0.5^{2.5}$, which corresponds to $\approx 17.7\%$ of 1-bits in the input vector, a large word size is preferable. For all other densities, it was evident that small word sizes lead to a significant improvement of the compression rate. Even at low densities starting

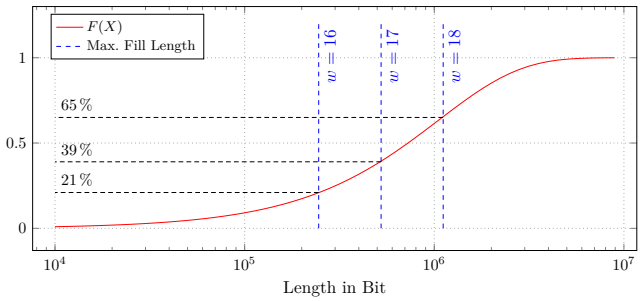


Figure 5: Distribution function of the geometric distribution of d.

from $d = 0.5^{12}$ only slight differences between the different word sizes are recognizable. It can be deduced from this observation that small word sizes, with the exception of high 1-densities should be preferred if the input data seems to be uniformly distributed.

Another remarkable observation is the non-monotonic progression of the size of the compressed vectors at very low densities (Figure 4). This can be explained by a simple experiment:

For a density $d = \frac{1}{2^{20}}$ the expected length of a 0-fill is $2^{20} - 1$. To encode such a fill, a word size of 18 bits is still sufficient, since

$$(2^{18-2} - 1) \cdot (18 - 1) > (2^{20} - 1)$$

If we now look at the cumulative distribution function of the associated geometric distribution (Figure 5), we can see that 65% of the 0-fills can be encoded with a word size of 18 bits. However, for a word size of 16 bits only 21% of the 0-fills can be covered. This results much more often in the need to use two instead of one fill word for a word size of 16 bits. The effect can also be observed in a weakened form with smaller word sizes, where three fill words must be used instead of two. But due to the small word size there are not that many unused bits, which reduces the effect significantly.

In summary, for uniformly distributed bitmaps the best compression was achieved, with the exception of uncompressible data, with word sizes between 4 and 8 bits. Overall, the factor of memory requirement between *VWLWAH* and *WAH* for compressible data is up to 3.98 (with $d \approx 0.55\%$).

4.2 Clustered Bitmaps

Since uniformly distributed bitmaps do not necessarily represent real world behaviour (*i.e.* for clustered bitmaps) we used a Markov process as described by Wu *et al.* [17] to evaluate *VWLWAH* with clustered bitmaps.

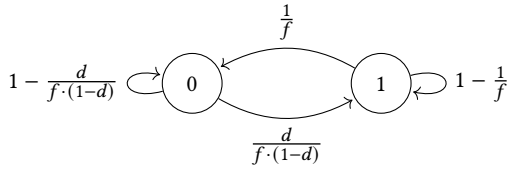


Figure 6: Markov process as described by Wu *et al.* [17].

While d denotes the density, the clustering factor f describes the probability of consecutive 1-bits, with the following rule: $f \geq 1$ and $f \geq \frac{d}{d-1}$

The main focus of this experiment was the influence of the clustering factor f on the compression rate with different word sizes. This is due to the fact that a larger clustering factor promises a higher occurrence of consecutive 1-bits, which should have a beneficial effect on the compression rate, as there are now potentially fewer literal words and a even higher possibility to store 1-bits in a 1-fill. We chose the same values for the density d as in the previous experiment and f as follows: $f \in \{2^n | n \in 0, \ldots 10\}$.

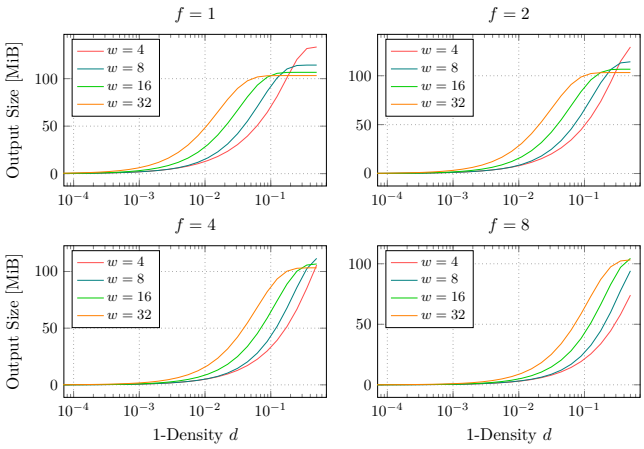


Figure 7: Size of the compressed bitvectors with varying densities and fixed word sizes.

Evaluation. We first evaluated the impact of varying densities on a fixed word size with different clustering factors. Contrary to the general assumption that clustered data is more compressible, a clustering factor of $f = 1$ shows a clearly different picture. This observation can be explained by the fact that two or more consecutive 1-Bits and thus 1-fills are not possible with a clustering factor of $f = 1$. Furthermore a negative effect on smaller word sizes is also noticable, since the possibility of storing two 1-bits in a single literal word is also low.

Apart from this exception, clustered bitmaps show a significant better compressibility (Figure 7). It is also clearly visible that for a word size of $w = 4$ and clustering factors of $f > 4$ the compressed size of bit vectors of any 1-density is smaller than the size of larger word sizes. For higher clustering factors this effect can be also observed with larger word sizes (*e.g.* $w = 8$).

As in Section 4.1 we examined the impact of fixed densities to varying word sizes with different clustering factors. The analysis of the most efficient word sizes shows a very similar result to the corresponding analysis with uniformly distributed data. Interestingly the non-monotonic progress for low 1-densities is also oberservable for data generated by a markov process and occur at nearly the same word sizes. Interestingly different clustering factors for fixed 1-density seem to have no measurable influence on the optimal word size. The word size, which gives the best result depending

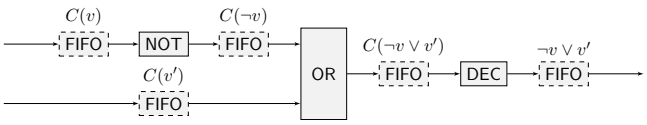


Figure 8: Pipeline that performs an OR-Operation to a negated bitvector v and a bitvector v'.

on the density of the 1-bits, is always within the same range for the test data used here as for the previously performed tests on uniformly distributed data, with minimal deviations.

Despite the fact, that clustered bitmaps typically offer a better compressibility, it can be summarized that a word size of 4 to 8 bit usually offers the best compression ratio for both distributions and low 1-densities.

5 VWLWAH ON FPGAS

VWLWAH has been designed with modern hardware platforms, FPGAs in particular, in mind. The compression scheme is part of a library for FPGA-based database acceleration that we develop at TU Dortmund University.

To leverage the strengths of FPGAs, we set up the FPGA implementation of VWLWAH as a set of *hardware components* that can operate in a streaming manner and with high throughput. The library includes components for VWLWAH compression and decompression, for pipeline management (*e.g.*, FIFO queues), as well as for bitwise Boolean operations that can be evaluated on compressed data.

The library is fully modular. Hardware components can not only be combined to implement more complex bitwise operations on compressed bit vectors. They also integrate well with other streaming operators that support FPGA acceleration in the system's data path. To illustrate, Figure 8 shows an operator composition that would combine two input bit vectors v and v', negating v and bitwise `OR`ing it with v'.

Like *VAL-WAH*, *VWLWAH* also supports the processing of bit vectors of different length. Therefore it is necessary to scale bit vectors. We implemented a scaling unit that can scale bit vectors up and down by a factor that is a power of two.

6 RUNTIME BEHAVIOUR

The use of small word sizes can lead to significant improvements regarding the compression rate. However, due to the fixed word size of current CPUs, the idea of *VWLWAH* promises no advantages for commodity hardware. In contrast, accelerators, such as FPGAs, which do not rely on a fixed word size for data processing could profit massively from VWLWAH.

To demonstrate that potential, we evaluated our VWLWAH implementation on mainstream FPGA hardware and let it compete against a general-purpose CPU. Since "throughput" is much easier to interpret for a single input data stream, we chose 'negation' as the operation to apply to the compressed bit vector in our experiments.

6.1 Theoretical Runtime Evaluation

In order to be able to make more precise statements about the performance of our approach on real hardware, we first determined the theoretical latency and throughput of the negation component. The total latency is composed of the input/output latency Δt_{IO} and the circuit latency Δt_{CL}. The input/output latency Δt_{IO} describes the time period between the moment a signal is applied to a processing component and the moment it is registered by the processing component. With a few exceptions, signals are always processed with a rising clock edge. The circuit latency Δt_{CL}, on the other hand, describes the time that a component requires for processing until output.

The input/output latency Δt_{IO} for the negation component is 1 cycle in the worst case. The circuit latency is also just one cycle, which can be explained by the fact that in literal words only each individual bit of a block has to be negated, which can be done in one cycle. The negation of fill words is even easier, since only the type of fill has to be changed. This results in a latency of 2 cycles in total for the negation component and a throughput of one word per cycle.

6.2 Experimental Runtime Evaluation

For the practical evaluation we used the Zynq Ultrascale+ ZCU102 evaluation kit from Xilinx. In addition to programmable logic (PL), this development board contains a quad-core CPU (PS) from ARM, whereby the communication between both components is carried out via the AXI bus. In order to access the main memory also from the programmable logic, we used an AXI Streaming DMA IP from Xilinx [19]. This made further components necessary to convert the AXI streaming data to *VWLWAH* compatible data.

For the evaluation of the runtime a word size of 32 bit was chosen for *VWLWAH*. This decision may seem unintuitive at first glance, as the memory evaluation in Section 4 would have suggested a small word size, but it is justified by the fact that the AXI bus in our experiments also used a bus width of 32 bits. By using a word size of 32 bits the two conversion components mentioned above can be excluded from becoming a bottleneck. In addition, it allows also a comparison with a CPU-based implementation.

For our experiments, we chose clock frequencies between 100 MHz and 375 MHz for the programmable logic, since higher frequencies caused complications during data transmissions on the AXI bus. The size of the investigated input vectors was varied between 500 and 4000 words. Therefore, our time measurements were started before reading the input data from main memory and ended after processing and complete presence of the output data in the main memory. The generation of the input data was not included in the time measurement.

The results are shown in Figure 9. It can be clearly seen, that there is a linear relationship between input size and processing time for each measured frequency. Based on the measurements we were able determine the parameters $\hat{\beta}_0$ and $\hat{\beta}_1$ for a linear regression. However, it is noticeable that with increasing clock speeds, the practical throughput is lower than the theoretically possible throughput, whereas for low frequencies the theoretical throughput can be achieved (see Figure 9 and Table 1). We explain these deviations by memory access latencies. Therefore we repeated the experiment once again with a word size of 8 bit values. Here, a similarly linear behaviour could be observed, but even at high frequencies only very slight deviations occurred.

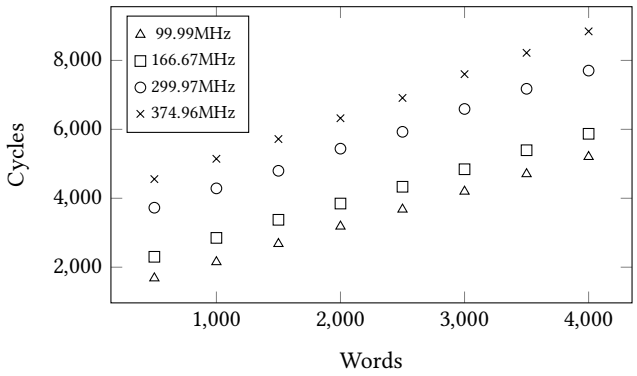


Figure 9: Cycles for different input sizes with different frequencies.

Frequency	Constant $\hat{\beta}_0$	Regression Coefficient $\hat{\beta}_1$	Goodness of Fit R^2
99.99 MHz	1180.0 cy	0.9994 cy	0.9997
124.99 MHz	1423.5 cy	1.0032 cy	0.9999
166.67 MHz	1820.3 cy	1.0139 cy	0.9997
214.29 MHz	2294.7 cy	1.0310 cy	0.9997
249.98 MHz	2593.6 cy	1.0933 cy	0.9991
299.97 MHz	3125.9 cy	1.1467 cy	0.9993
374.96 MHz	3899.6 cy	1.2289 cy	0.9994

Table 1: Parameters for a linear Regression of all measured frequencies and a word size of 32 bit

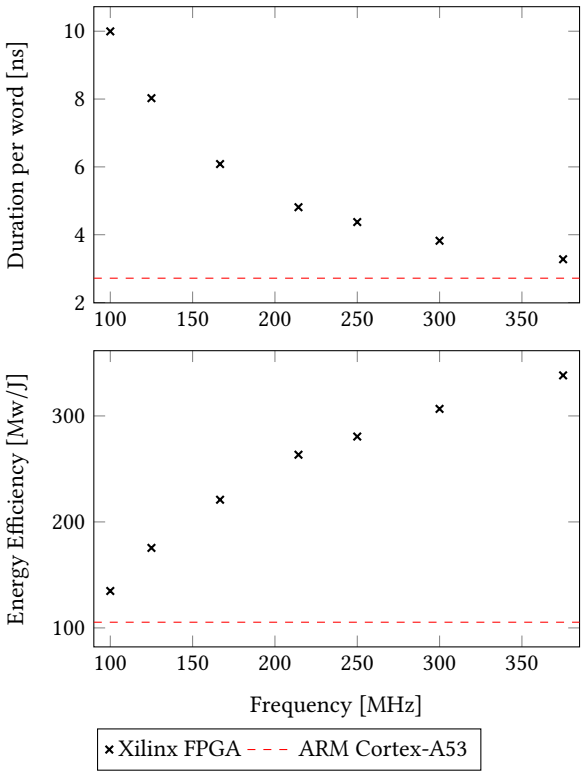


Figure 10: Time needed on the FPGA and on the Odroid C2

6.3 Power Consumption

Another important aspect is the energy consumption. For this reason, we investigated the energy consumption of *VWLWAH* with a word size of 32 bit on a CPU as well as on the FPGA. The power consumption of the FPGA is an estimate of the Vivado suite from Xilinx. For the examination of the power on the CPU side, we used an Odroid C2 with an ARM Cortex-A53 Quad-Core with a fixed clock speed of 1.5 GHz. The power consumption of the entire system was measured with the Odroid SmartMeter.

As can be seen in Figure 10, the CPU can achieve a higher throughput than the implementation on the FPGA, but there is a clear indication that the implementation on the FPGA side has significant saving potential in terms of energy consumption. This point is especially noteworthy, if it is considered that the word size of 32 bits is likely to have a rather positive effect on the throughput and energy consumption of the CPU.

7 CONCLUSION AND OUTLOOK

In this paper we discussed the influence of variable word sizes for WAH based compression techniques. For this purpose, we took the original WAH scheme and combined it with two existing optimizations, analyzed the possible savings and implemented it on a FPGA, since variable word sizes do not seem beneficial in terms of speed on conventional CPUs. Our analysis showed, that a small word size could lead to much better compressibility in most cases and space savings up to 75% compared to a standard word size. Furthermore, we were able to implement a working prototype on a FPGA to demonstrate the feasibility of our approach on non-standard hardware.

A key feature of *VWLWAH* is its simplicity resulting from the strong similarity with the original *WAH* idea, but other optimizations, such as meta-compression, would be possible here as well.

REFERENCES

[1] Andreas Becher, Florian Bauer, Daniel Ziener, and Jürgen Teich. 2014. Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.

[2] Jiahui Chang, Zhen Chen, Wenxun Zheng, Junwei Cao, Yuhao Wen, Guodong Peng, and Wen-Liang Huang. 2015. SPLWAH: A bitmap index compression scheme for searching in archival internet traffic. In *2015 IEEE International Conference on Communications (ICC)*. IEEE, 7089–7094.

[3] Jiahui Chang, Zhen Chen, Wenxun Zheng, Yuhao Wen, Junwei Cao, and Wen-Liang Huang. 2014. PLWAH+: A bitmap index compressing scheme based on PLWAH. In *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 257–258.

[4] Alessandro Colantonio and Roberto Di Pietro. 2010. Concise: Compressed 'n'composable integer set. *Inform. Process. Lett.* 110, 16 (2010), 644–650.

[5] Intel Corporation. 2015. Xeon + FPGA Platform for the Data Center.

[6] Fabian Corrales, David Chiu, and Jason Sawin. 2011. Variable length compression for bitmap indices. In *International Conference on Database and Expert Systems Applications*. Springer, 381–395.

[7] François Deliège and Torben Bach Pedersen. 2010. Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. In *Proceedings of the 13th international conference on Extending Database Technology*. 228–239.

[8] Francesco Fusco, Marc Ph Stoecklin, and Michail Vlachos. 2010. Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1382–1393.

[9] Gheorghi Guzun, Guadalupe Canahuate, David Chiu, and Jason Sawin. 2014. A tunable compression framework for bitmap indices. In *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 484–495.

[10] Daniel Lemire, Owen Kaser, and Kamel Aouiche. 2010. Sorting improves word-aligned bitmap indexes. *Data & Knowledge Engineering* 69, 1 (2010), 3–28.
[11] Nusrat Jahan Lisa, Annett Ungethüm, Dirk Habich, Wolfgang Lehner, Tuan DA Nguyen, and Akash Kumar. 2018. Column Scan Acceleration in Hybrid CPU-FPGA Systems.. In *ADMS@ VLDB*. 22–33.
[12] Rene Mueller, Jens Teubner, and Gustavo Alonso. 2009. Data Processing on FPGAs. *Proceedings of the VLDB Endowment* 2, 1 (Aug. 2009), 910–921.
[13] Andreas Schmidt, Daniel Kimmig, and Mirko Beine. 2011. A proposal of a new compression scheme of medium-sparse bitmaps. (2011).
[14] Sebastiaan J van Schaik and Oege de Moor. 2011. A memory efficient reachability data structure through bit vector compression. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 913–924.
[15] Yuhao Wen, Zhen Chen, Ge Ma, Junwei Cao, Wenxun Zheng, Guodong Peng, Shiwei Li, and Wen-Liang Huang. 2014. SECOMPAX: A bitmap index compression algorithm. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–7.
[16] Louis Woods, Zsolt István, and Gustavo Alonso. 2014. Ibex: an intelligent storage engine with support for advanced SQL offloading. *Proceedings of the VLDB Endowment* 7, 11 (2014), 963–974.
[17] Kesheng Wu, Ekow J Otoo, and Arie Shoshani. 2006. Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems (TODS)* 31, 1 (2006), 1–38.
[18] Yinjun Wu, Zhen Chen, Yuhao Wen, Wenxun Zheng, and Junwei Cao. 2016. Combat: A new bitmap index coding algorithm for big data. *Tsinghua Science and Technology* 21, 2 (2016), 136–145.
[19] Xilinx. 2019. AXI DMA v7.1 LogiCORE IP Product Guide.