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

The Word-Aligned Hybrid (WAH) compression is a prominent example of a lightweight compression scheme for bitmap indices that considers the *word size* of the underlying architecture. This 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.

ACM Reference Format:
Florian Grieskamp, Roland Kühn, and Jens Teubner. 2020. Variable Word Length Word-Aligned Hybrid Compression. In *International Workshop on Data Management on New Hardware (DAMON'20), June 15, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3399666.3399935

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 indices* 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]. 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 CPUs, where operations on data below the intrinsic machine word size are complicated and slow.

Since the inception of WAH, the hardware landscape has changed significantly. Commodity CPUs have become just one piece in a mix of heterogeneous processing components with different or no specific word sizes (e.g. FPGAs).

DAMON'20, June 15, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-8024-9/20/06…$15.00
https://doi.org/10.1145/3399666.3399935

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 indices 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.

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. To achieve this, repeated chunks of 1s or 0s are represented as single *fill words* while incompressible chunks are stored as *literal words*.

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

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". [2–4, 6, 7, 9, 10, 13, 14]. Other papers discuss the possibility of meta-compression, *i.e.*, to compress the output of already WAH-compressed data [8, 15, 18].

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[1, 11, 16].

3 VARIABLE WORD LENGTH WAH

WAH and all variants known to us stick to the idea of matching the *machine word size* of the underlying hardware, usually 32 or even 64 bits. 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.

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

[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 indices an attribute a with α distinct values ("attribute cardinality") will consist of α bit vectors $\mathbf{x}_1, \ldots \mathbf{x}_\alpha$ of length $|R|$. Since every row in R will carry exactly one a value, all bit vectors together will have exactly $|R|$ bits set to 1. Under the premise of a uniform data distribution in R, the proportion of 1-bits in a vector, the 1-density, will be α^{-1}.

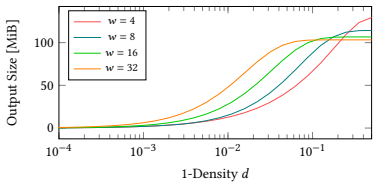

Figure 1: Size of the compressed vectors with varying density for four different word sizes and uniformly distributed values

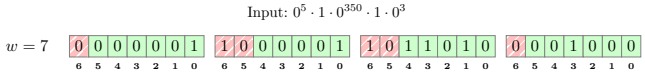

Figure 2: Representation of a compressed bitvector with a word size of $w = 7$. In that case two fill words are concatenated to represent a long 0-fill

The right end of the plot shows the "compression penalty" on incompressible data. While we observe that the compression penalty is higher for smaller word sizes, it is also clear that smaller word sizes enable compression for a wider variety of input data. 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 $w = 32$.

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, $w < 32$ will significantly improve the compression ratio of WAH. 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 encoding. To remedy the situation, we picked up the idea of flexible block sizes that was introduced by Guzun et al. [9] to devise our *Variable Word Length Word-Aligned Hybrid (VWLWAH)* compression scheme.

Figure 2 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 as originally proposed by van Schaik and de Moor [14]. In the illustrated example, this enables the encoding of up to 2^{5+5} blocks of 0s with just two encoded 7-bit words.

4 EXPERIMENTS

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. 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.

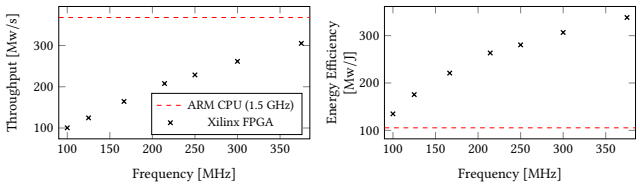

Figure 3: Throughput [million words per second] and energy efficiency [million words per joule] on the Odroid C2 and on the FPGA with different frequencies

For our experiments, the size of the investigated input vectors was varied between 500 and 4000 words. We chose clock frequencies between 100 MHz and 375 MHz for the PL, since higher frequencies caused complications during data transmissions on the AXI bus. Despite the significantly lower frequencies, the FPGA almost reaches the same throughput as the CPU-based implementation.

Another important aspect is energy consumption. For this reason, we investigated the energy consumption of *VWLWAH* 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 monitored an Odroid C2 with an ARM Cortex-A53 Quad-Core (1.5 GHz) with the Odroid SmartMeter.

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 would have suggested a small word size. We expect the word size of 32 bit to result in a best case scenario for the CPU. As can be seen in Figure 3, the CPU can achieve 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 all word sizes below 32 bits are likely to have a rather negative effect on the throughput and energy consumption of the CPU.

5 CONCLUSION AND OUTLOOK

In this paper we discussed the influence of variable word sizes for WAH based compression techniques. For this purpose, we optimized the original WAH scheme, analyzed the possible savings, and implemented it on an FPGA. 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. Our analysis shows that a small word size leads to much better compressibility in most cases and space savings of up to 75% compared to a standard word size. Furthermore, we were able to implement a working prototype on an FPGA to demonstrate the feasibility of our approach. We continue to follow this approach and are currently developing a library for FPGA-based database acceleration at TU Dortmund University.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. The work has received funding from the *Deutsche Forschungsgemeinschaft* (DFG), SFB 876, project C5 (http://sfb876.tu-dortmund.de/).

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.