A Case for Hardware-Supported Sub-Cache Line Accesses

Christopher Schmidt
Markus Dreseler
christopher.schmidt@hpi.de
markus.dreseler@hpi.de
Hasso Plattner Institute
Potsdam, Germany

Berkin Akin
Intel Labs
Hillsboro, Oregon, US
berkin.akin@intel.com

Amithaba Roy*
Google
Mountain View, California, US
amitabha.roy@gmail.com

ABSTRACT

Largely, the performance of main-memory databases is limited by the growing memory gap. To be efficient, a DBMS cannot waste any bandwidth. However, this is exactly the case when 64-byte cache lines are transferred but only parts of these are used. We present a cache simulator that measures this waste, show that current databases waste up to 70% of the available bandwidth, and discuss a new gather instruction with sub-cache line access granularity that could reduce this waste.

CCS CONCEPTS

•Information systems →Data management systems; •Software and its engineering →*Memory management;*

KEYWORDS

Cache Line Utilization, Memory Bandwidth, Near-Data Processing, In-Memory Database Management Systems

ACM Reference format:
Christopher Schmidt, Markus Dreseler, Berkin Akin, and Amithaba Roy. 2018. A Case for Hardware-Supported Sub-Cache Line Accesses. In *Proceedings of 14th International Workshop on Data Management on New Hardware, Houston, TX, USA, June 11, 2018 (DaMoN'18),* 5 pages.
DOI: 10.1145/3211922.3211924

1 INTRODUCTION

For up to half of their execution time, main-memory databases are memory-bound [6, 19]. Different optimizations have been introduced to reduce the memory access cost, including cache-efficient programming [5], compression [1], and dedicated hardware [7]. What is usually taken for granted is the cache line granularity. We cannot read an integer from memory without reading an entire 64-Byte line. An efficient DBMS thus arranges its data in a way where the other loaded bytes are also used. This is the core of the row vs. column store discussion for main-memory databases [2].

Not always can all of the cache line be used. While a column store can scan a sequentially stored column without wasting any bytes of a cache line, it can only do so for the first predicate. For a second predicate, where only some rows are probed, the memory access pattern is random, causing data to be loaded, but not used.

In a discussion at the Dagstuhl seminar *Databases on Modern Hardware* [8], participants agreed that reducing this waste by allowing sub-cache line accesses could be a game changer. With this work, we quantify potential benefits using a waste-tracking cache simulator and show that current databases do indeed waste a significant amount of memory bandwidth. Working towards a solution for this, we outline a more efficient gather instruction, and present the case for sub-cache line access to the hardware community.

2 RELATED WORK

Near Memory Processing is an active area of research, aiming to alleviate the memory bandwidth bottleneck by implementing smarter logic into the memory controller [9, 10, 18] or adding hardware, such as an external integrated circuit mountable to the DIMM [20]. Xi et al. [20] address the memory bandwidth bottleneck of modern column-oriented DBMS by implementing operators on near-data processing hardware accelerators. In particular, they implement a select operator that filters data close to memory, transferring only qualifying data to CPU. Seshadri et al. [18] propose a gather-scatter DRAM *GS-DRAM* that improves spatial locality of non-unit strided accesses. They distribute values of a cache line to different chips within the DRAM module, implement logic in the memory controller for fixed-width strided patterns and access them in a single read/write command. In contrast, we propose a general instruction that is not limited to fixed-width strides.

3 QUANTIFICATION

To measure the cache line utilization (CLU), every single cache slot has to track which bytes have been loaded into a CPU register. This makes hardware counters infeasible. Instead, we built a tool based on the PIN dynamic binary instrumentation framework [15] [1]. PIN instruments a binary by adding hooks for selected instructions, such as loads. We modified the `dcache` tool from the PIN package, which already provides an associative data cache. For each loaded cache line, we keep a 1-Byte bitset, where each bit represents whether the corresponding 8-Byte chunk in the cache line has been used by the CPU. When the cache line is evicted, or when the program terminates, we aggregate the number of used 8-Byte chunks and the number of cache lines loaded. The quotient is the CLU.

*Work done while author was at Intel Labs

DaMoN'18, Houston, TX, USA

DOI: 10.1145/3211922.3211924

[1] Code and instructions for reproduction at https://github.com/hyrise/sub-cache-line-access

The results meet our expectations and we list three limitations mostly for completeness: First, because the actual cache replacement strategies of modern CPUs are unknown [3], we use 4-way associativity and a round robin replacement algorithm. Second, atomic operations on the data structures would be costly and slow down the simulation, thus we currently only support single-threaded workloads. This is an implementation decision, not a fundamental limitation. Third, we only track data loads, not instruction loads. Having only a single cache level is *not* a limitation, because we are interested in data loaded from DRAM into the CPU, no matter in which cache level it ends up.

A second tool that tracks the CLU is Intel Advisor[2]. Comparing both tools, we obtained similar results showing that the discussed limitations do not harm the accuracy of our instrumentation. Different from our tool, advisor requires the user to manually select program loops to be tracked, while our tool can treat the database as a black box. Also, our tool allows for adjustable access granularity.

For a first sanity check, we have build a prototypical row/column store example. With 10, 000 rows of 100 8-Byte integers stored either in row-major or column-major format, we scan one column for a given value. A row store can only use $\frac{1}{8}$ of each cache line. Our tool reports a CLU of 13.10%. For the column store, the CLU is 98.11%.

Using our PIN tool, we have executed the TPC-H benchmark[3] on two research databases, Peloton [16] and Hyrise [13]. We focus on analytical workloads because the proposed instruction works best with operators accessing multiple memory locations (i.e., rows) sequentially. Transactional workloads with point-accesses cannot be optimized with the instruction. As Hyrise is being rewritten and does not yet support subqueries, we show results for a subset of queries. Experiments were executed with a scale factor of 1. The physical system configuration is irrelevant for our instrumentation.

	Q1	Q3	Q5	Q6	Q7	Q9	Q10
Hyrise	78.35	70.00	56.67	74.64	94.13	71.58	90.81
Peloton	49.60	42.50	49.39	28.35	44.74	37.86	40.54

Table 1: Cache line utilization in percent for selected TPC-H Queries with Scaling Factor of 1 based on a for 4-way associative 16 MB cache and a sub-cache line granularity of 8 Byte

Table 1 shows the CLU for these queries. For Hyrise, where data is stored in a column-major format[4], we see that the CLU varies. In the worst case, less than 36 Bytes out of a 64 Byte cache line are used. Peloton, in a row-major configuration, uses, in average, less than 32 Bytes and can have a CLU of only 28%, e.g., in Q6.

This shows a significant waste of memory bandwidth, but does not translate one-to-one in an expected performance benefit of sub-cache line access of 70%. While operators such as column scans on row-major data could be improved to almost 100% efficiency, it will be more difficult for pointer chasing operations (e.g., indexes). Measuring the actual benefit is part of our ongoing work.

[2]https://software.intel.com/en-us/intel-advisor-2017-user-guide-linux
[3]http://www.tpc.org/tpch/
[4]Hybrid data layout for the new version of Hyrise is still work-in-progress.

4 PROPOSED INSTRUCTION

The interface most similar to what we are suggesting is the gather instruction in AVX-512. It takes a base address to the location in memory and an index list containing the offsets to the actual values. It then gathers values from different memory locations into a single 512-bit register. This can already improve the performance of a scan by up to a factor 10 [12]. Its shortcoming is that the data is first transferred on a cache line granularity to the CPU. Only there, the values are extracted and combined. If we could get the memory hardware to assemble the requested values into a cache line, only relevant data would be send over the memory bus, see Figure 1.

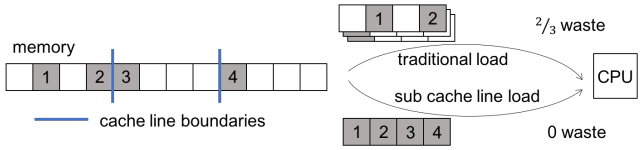

Figure 1: Waste in memory bandwidth for a traditional load compared with a sub cache line load

The proposed sub-cache line gather (SCG) instruction has a similar interface, but uses sub-cache line memory accesses when assembling the data, instead of bringing in every cache line containing elements. Although off-the-shelf DRAM modules provide full 64-Byte data per access, we are motivated by recent memory technologies such as High Bandwidth Memory and Hybrid Memory Cube, which can provide sub-cache line memory accesses of 32 and 16 Byte granularity [14, 17]. Also, techniques such as sub-ranking or burst-chop can enable smaller granularity memory accesses [4].

Scalar sub-cache line memory accesses lead to partially valid lines in caches complicating the cache design. However, SCG forms full cache lines via several sub-cache line accesses before inserting the data which eliminates the partial line issue. After assembling the full cache line, SCG can target an AVX register or a pre-allocated memory region. Latter is useful to exploit temporal locality, setting the destination to L2/L3 using cache allocation technology [11].

A limitation of this approach is that the read data is a non-cache-coherent copy of the data stored in memory. Modifications do not get propagated to other copies of the data. Because we see the use case mainly for read operators, we do not consider this to be an issue. For use cases that require a write operation a symmetric scatter instruction could be added.

5 SUMMARY AND FUTURE WORK

For two research databases, we have shown that they underutilize the data stored in the cache lines loaded into the CPU. Some TPC-H queries waste as much as 70% of the available bandwidth. We have proposed a gather instruction that assembles data with a sub-cache line granularity before it is sent over the memory bus.

As a next step, we will work on simulating this proposed instruction. Once this becomes available, preferably via a compiler intrinsic, we can prototypically implement first database operators with sub-cache line access. This should allow us to not only quantify the waste, but also gains expected for SCG.

Accepted at DaMoN.
The final authenticated version is available online at https://dx.doi.org/10.1145/3211922.3211924

While we have only discussed databases in this paper, we believe that other types of memory-bound programs can also profit, especially when future compilers identify scattered memory reads and automatically emit the appropriate gather instruction.

REFERENCES

[1] Daniel Abadi, Samuel Madden, and Miguel Ferreira. 2006. Integrating Compression and Execution in Column-oriented Database Systems. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD '06)*. ACM, New York, NY, USA, 671–682. DOI: http://dx.doi.org/10.1145/1142473.1142548

[2] Daniel J. Abadi, Samuel R. Madden, and Nabil Hachem. 2008. Column-stores vs. Row-stores: How Different Are They Really?. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. ACM, New York, NY, USA, 967–980. DOI: http://dx.doi.org/10.1145/1376616.1376712

[3] A. Abel and J. Reineke. 2014. Reverse engineering of cache replacement policies in Intel microprocessors and their evaluation. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 141–142. DOI: http://dx.doi.org/10.1109/ISPASS.2014.6844475

[4] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi. 2009. Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs. *IEEE Computer Architecture Letters* 8, 1 (Jan 2009), 5–8. DOI: http://dx.doi.org/10.1109/L-CA.2008.13

[5] Anastasia Ailamaki, David J. DeWitt, Mark D. Hill, and Marios Skounakis. 2001. Weaving Relations for Cache Performance. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 169–180. http://dl.acm.org/citation.cfm?id=645927.672367

[6] Anastasia Ailamaki, David J. DeWitt, Mark D. Hill, and David A. Wood. 1999. DBMSs on a Modern Processor: Where Does Time Go?. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 266–277. http://dl.acm.org/citation.cfm?id=645925.671662

[7] K. Aingaran, S. Jairath, G. Konstadinidis, S. Leung, P. Loewenstein, C. McAllister, S. Phillips, Z. Radovic, R. Sivaramakrishnan, D. Smentek, and T. Wicki. 2015. M7: Oracle's Next-Generation Sparc Processor. *IEEE Micro* 35, 2 (Mar 2015), 36–45. DOI: http://dx.doi.org/10.1109/MM.2015.35

[8] Gustavo Alonso, Michaela Blott, and Jens Teubner. 2017. Databases on Future Hardware (Dagstuhl Seminar 17101). *Dagstuhl Reports* 7, 3 (2017), 1–18. DOI: http://dx.doi.org/10.4230/DagRep.7.3.1

[9] Marco A. Z. Alves, Paulo C. Santos, Francis B. Moreira, Matthias Diener, and Luigi Carro. 2015. Saving Memory Movements Through Vector Processing in the DRAM. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '15)*. IEEE Press, Piscataway, NJ, USA, 117–126. http://dl.acm.org/citation.cfm?id=2830689.2830705

[10] J. Carter, W. Hsieh, L. Stoller, M. Swanson, Lixin Zhang, E. Brunvand, A. Davis, Chen-Chi Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. 1999. Impulse - Building a Smarter Memory Controller.. In *Proceedings Fifth International Symposium on High-Performance Computer Architecture*. IEEE Comp Soc, 70–79. DOI: http://dx.doi.org/10.1109/HPCA.1999.744334

[11] Intel Corporation. 2015. *Improving Real-Time Performance by Utilizing Cache Allocation Technology White Paper.* Technical Report. Intel Corporation. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf

[12] Markus Dreseler, Jan Kossmann, and Johannes Frohnhofen. 2018. Using AVX-512 Mask Instructions to Double the Performance of Chained Table Scans. *Joint Workshop of HardBD (International Workshop on Big Data Management on Emerging Hardware) and Active (Workshop on Data Management on Virtualized Active Systems), in conjunction with ICDE* (Apr 2018), to appear.

[13] Martin Grund, Jens Krüger, Hasso Plattner, Alexander Zeier, Philippe Cudre-Mauroux, and Samuel Madden. 2010. HYRISE: A Main Memory Hybrid Storage Engine. *Proc. VLDB Endow.* 4, 2 (Nov. 2010), 105–116. DOI: http://dx.doi.org/10.14778/1921071.1921077

[14] JEDEC. 2015. *JEDEC Standard JESD235A: High Bandwidth Memory (HBM) DRAM.* JEDEC Solid State Technology Association, Virginia, USA.

[15] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05)*. ACM, New York, NY, USA, 190–200. DOI: http://dx.doi.org/10.1145/1065010.1065034

[16] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *CIDR 2017, Conference on Innovative Data Systems Research*. http://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf

[17] J. T. Pawlowski. 2011. Hybrid memory cube (HMC). In *2011 IEEE Hot Chips 23 Symposium (HCS)*. 1–24. DOI: http://dx.doi.org/10.1109/HOTCHIPS.2011.7477494

[18] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. 2015. Gather-scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided

Accepted at DaMoN.
The final authenticated version is available online at https://dx.doi.org/10.1145/3211922.3211924

Accesses. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. ACM, New York, NY, USA, 267–280. DOI: http://dx.doi.org/10.1145/2830772.2830820

[19] Utku Sirin, Pinar Tözün, Danica Porobic, and Anastasia Ailamaki. 2016. Micro-architectural Analysis of In-memory OLTP. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 387–402. DOI: http://dx.doi.org/10.1145/2882903.2882916

[20] Sam Likun Xi, Oreoluwa Babarinsa, Manos Athanassoulis, and Stratos Idreos. 2015. Beyond the Wall: Near-Data Processing for Databases. In *Proceedings of the 11th International Workshop on Data Management on New Hardware (DaMoN'15)*. ACM, New York, NY, USA, Article 2, 10 pages. DOI: http://dx.doi.org/10.1145/2771937.2771945

Accepted at DaMoN.
The final authenticated version is available online at https://dx.doi.org/10.1145/3211922.3211924