Advanced Architectures: Simulator release for the emerging memory technologies

D4.4 v1.0

WP4 – Advanced Architectures: D4.4 Simulator release for the emerging memory technologies

Dissemination Level: Confidential

Lead Editor: Adrián Cristal

Date: 17/10/2013

Status: Final

Description of Work:

T4.4. A simulation platform for the emerging memory technologies: Memory has always been a critical point in the performance of the database systems, and having faster or larger memory can significantly improve the performance of entire database systems. Emerging Non-Volatile (NV) memory technologies have a promising advantage over the existing software solutions, since it eliminates the necessity for additional slow permanent storage

(e.g. hard disks). However, currently there is no analytical approach to know in advance what will be the performance of a DBMS with a certain type of memory technology. This task will provide a simulation platform that will allow us to experimentally estimate a performance advantage of using a certain memory technology with unmodified DBMSs. We will explore two approaches for making the simulator: (1) at the OS-level, providing a kernel module that extends the ramfs, and (2) a stand-alone architectural simulator written in software. The first choice has the advantage of providing lower overhead and higher execution speed; however, it cannot simulate very fast NV memories. The second choice is more general, supports all types of NV memories, but it has lower speed of execution. For this, we will analyse two popular simulators the GEM5, targeting a low-power ARM type architecture, and PTLSim modelling the AMD x86_64 architecture. During the first three months, we will select one of these simulators as the baseline, and for the rest of the project, we will modify them extensively so that they could model the memory hierarchy using emerging memory technologies. By the end of the project, the simulator will run selected slices of execution from the AXLE application workloads.

A first version of the modified simulator will be released for the partners at month 12, and will be made publicly available at month 24.

Contributors:

Adrián Cristal
Adrià Armejach
Nehir Sonmez
Nikola Markovic

Internal Reviewer(s):

2ndQ: Martin Marques
PV: Olivier Marchesini

Version History

Version	Date	Authors	Sections Affected
0.1	17/10/2013	AA,AC,NS,NM	Initial version
0.2	21/10/2013	Nehir Sonmez	Added content
0.3	25/10/2013	Adrià Armejach	Changes due to comments from PV review.
	31/10/2013	Adrià Armejach	Changes due to comments from 2ndQ review.
1.0	01/11/2013		Final version.

Table of Contents

List of Figures

List of Tables

1. Summary

This deliverable describes the work done during the first year regarding the simulation platform for emerging memory technologies. We explain the decisions taken when choosing an appropriate simulation platform, and how our choice will provide a suitable platform to explore unconventional heterogeneous architectures and memory hierarchies. Later, we explain the process we followed to simulate PostgreSQL, our database management system of choice, running an industry standard workload (TPC-H). We introduce our simulation methodology, which consists of simulating a modern system architecture, including out-of-order cores, 3 levels of cache hierarchies, and a configurable DRAM memory subsystem. Our results include execution stack diagrams for the executed TPC-H queries that clearly show where execution time is spend from the hardware's perspective, and allow us to draw meaningful conclusions from our simulations. We conclude with a summary of future directions of research using emerging memory technologies and 3D stacking, which we anticipate will be of interest given the initial results obtained using our first version of the simulator.

2. Analysis / Description of work done

In this section we describe the process that we followed to select an appropriate simulation platform to simulate DataBase Management System (DBMS) workloads. We then describe the necessary steps to run our DBMS of choice, i.e. PostgreSQL, inside the simulator. We conclude with a set of results that characterize the PostgreSQL execution from a hardware perspective, showing where time is spent (e.g CPU, branching, memory). These results will guide our research in the months to come regarding emerging memory technologies and advanced hardware architectures that might include specialized accelerators and novel manufacturing technologies like 3D stacking.

2.1 Selection of an appropriate simulation platform

Our simulator selection process was based on the fact that a DBMS with relatively long running workload executions had to be efficiently and effectively simulated. We looked at the TPC-H workload [TPCH], which is a decision support benchmark that consists of a set of business oriented queries designed to have a high degree of complexity and give answers to relevant business questions. These queries can run from a few seconds to a few tens of seconds even for small data sets. Furthermore, depending on the query plan, very complex behaviour could be observed, necessitating the simulation of large and detailed program segments.

DBMS workloads have very distinct characteristics when compared to traditional High Performance Computing (HPC) benchmarks. Simulations of HPC workloads are usually conducted by performing extremely detailed simulation of code regions, which would run for

only one second on real hardware, but that could run for several hours if executed inside a simulation environment that has full-system capabilities, i.e. that simulates an operating system and the hardware architecture in great detail. Furthermore, simulating processors with large caches becomes increasingly challenging because the slow simulation speed does not allow for simulating huge dynamic instruction counts in a reasonable amount of time.

In cases where simulation times become infeasible, a common approach is to simulate a fixed number of instructions after a warm up period. This approach has proven to be effective in HPC benchmarks if the period of instructions simulated is large enough to capture the workload's behaviour in different phases. For long running benchmarks like TPC-H, simulating an entire query using a full-system simulator would take an infeasible amount of time (several days).

Industry simulators typically run at a speed of 1 to 10 kHz. Academic simulators, such as GEM5 and PTLSim are not truly cycle-accurate compared to real hardware, and therefore they are typically faster, with simulation speeds in the tens to hundreds of KIPS (kilo simulated instructions per second) range. One popular solution that newer simulators make use of in order to increase simulation speed is to employ sampling, or to simulate only a few simulation points. These simulation points are chosen either randomly, periodically or through phase analysis.

For the reasons mentioned above, and after initial testing using full-system simulators like GEM5 [Gem5], we decided to advocate for next-generation simulators that have been proposed recently, e.g. Sniper and Zsim [SniperSC11, ZsimISCA13].

2.1.1 The Sniper Simulator

Sniper is built on Graphite, which runs x86 binaries, enabling to run existing workloads without having to deal with porting issues across instruction-set architectures (ISAs). Graphite does so by building upon Pin [Pin], which is a dynamic binary instrumentation tool, used as the functional simulation front-end. Pin dynamically adds instrumentation code to a running x86 binary to extract instruction pointers, memory addresses, executed instructions, register content, etc. This information from Pin is then forwarded to a Pin-tool (Sniper in our case) which estimates timings for the simulated target architecture. Sniper also adds a shared multi-level cache hierarchy supporting write-back first-level caches and an MSI snooping cache coherency protocol to Graphite.

Sniper offers the combination of a highly accurate core model, with a fast and parallel simulation infrastructure. This combination provides accurate simulation with high performance, up to 2 MIPS. A key benefit of Graphite, inherited in Sniper, is that it is a parallel simulator by construction. A multi-threaded program running in Sniper leads to a parallel simulator. Sniper thus has the potential to be scalable as more cores are being integrated in

future multi-core processor chips.

Pin is instructed to add analysis routines, which send detailed instruction information to Sniper's timing models. By changing which analysis routines are enabled, one can efficiently switch into a functional simulation-only mode which runs at near-native execution speed (by adding no analysis routines), simulate just caches and branch predictors for functional warming (by instrumenting only memory operations and branch instructions), or detailed simulation using the provided core, interconnection and memory models (by placing additional instrumentation calls).

Even under a detailed simulation mode, simulations are at least two orders of magnitude faster than full-system simulators allowing execution of entire queries using small input sets in a reasonable amount of time. In addition, these simulators have other compelling features that we might exploit in the future, for example, accurate sampled simulation techniques for multi-threaded workloads [SniperISPASS13].

Sampling is done by periodically simulating detailed performance models during intervals of a predetermined length, separated by periods of non-detailed simulation. In contrast to single-threaded simulation, it keeps track of simulated time, and maintain inter-thread dependencies through shared memory and synchronisation events, even while fast-forwarding. Application periodicity needs to be taken into account to prevent detailed sampling intervals from aliasing with the application's periodic behavior, affecting both the fast-forwarding IPC and overall runtime prediction. In Sniper, a methodology to determine application phases and periodicity is used to select the appropriate options for the required speed and accuracy trade-offs. Through the use of micro-architecture independent methods of detecting periodicity, sampling parameters are derived up-front to allow for accurate run-time prediction.

Due to source code availability, our initial simulation environment is based on Sniper, however, as Zsim becomes available we might incorporate some features from it. This should be a straightforward process since both simulators follow the same philosophy and rely on Pin as its application programming interface (API). A caveat of these simulators is that they only simulate an application's user-space code, but solutions can be put in place to simulate system-calls with the details deemed necessary for a given workload. Simple solutions may use fixed costs for specific system calls, and more elaborate solutions could be implemented by adding an additional layer that simulates interactions between system calls. However, having simulations that are detached from the kernel (non full-system simulations) can also be an advantage, as it will allow us to simulate architectures that would not be supported by current kernels, such as heterogeneous architectures (combining different core models and/or specialized accelerators), or unconventional memory hierarchies.

2.2 First internal release of the simulator platform

This subsection describes the first internal release of the simulation platform that will be used from this point forward to investigate how emerging memory technologies and 3D stacking techniques can improve the performance of workloads that use very large data sets.

As stated before, the simulator is based on Sniper and its infrastructure. In the following subsections we describe the process we followed to integrate the workloads we are initially interested in, and how we configure the simulator to do an initial study to characterize how the hardware is being utilized when executing such workloads. We then describe the measurements and metrics we use to perform our initial evaluation in order to understand and to pinpoint potential bottlenecks present in current architectures.

Our initial simulation infrastructure does not include a kernel module. This is due to the fact that the chosen infrastructure is fast enough to relegate the anticipated need for a quick way to assess performance of DBMS executions.

2.2.1 Preparations to simulate PostgreSQL running TPC-H

Our main initial focus was to evaluate the PostgreSQL DBMS running the TPC-H suite of independent queries. In order to capture the region of interest (query execution) and not to simulate sections of the DBMS that are spurious to our study, such as the DBMS start-up and shutdown code regions, we modified the PostgreSQL sources to use Sniper “hooks” that let the simulator know when a certain region of interest (query execution) starts and finishes. These hooks were added and the source code was compiled with appropriate libraries to run successfully inside the simulator. These hooks also allow the simulator to avoid dynamic instrumentation of the code outside the region of interest allowing for near-native execution speed of these sections of code, known as fast-forwarding. The simulator later can switch to a detailed simulation mode when the region of interest is reached and switch back to fast-forwarding when it finishes. This approach saves a substantial amount of simulation time, allowing us to simulate in detailed mode the entire query execution in an acceptable period of time.

The release of the simulator thus includes the modified PostgreSQL sources along with a set of scripts that automate its compilation, manage simulator executions, gather statistics of interest in practical formats, and generate appropriate charts. This infrastructure will be maintained and extended as more workloads are included or new features added to the simulator.

2.2.2 Simulation methodology for initial evaluation

For our initial set of results shown in this study, we chose to configure the simulator to mimic an out-of-order processor modeling the Intel Nehalem processor. In particular, we model the Xeon X5550 Gainestown, which will allow us to see how our DBMS and workloads of interest

behave in current architectures of today. Table 1 lists the main characteristics of the simulated architecture. Only the region of interest of each query is included in our measurements, fast-forwarding was used during the startup and cleanup phases, and detailed simulation used during the entire region of interest. PostgreSQL was compiled using GCC 4.6.3 for x86_64. For our simulations we populated a 1GB database using the TPC-H data generator. We generated all the necessary indexes and set an appropriate PostgreSQL configuration to attain good performance. Larger databases would require simulation sampling methodologies [SniperISPASS13], which we have started to explore and will use in subsequent studies when using larger data sets. All queries were run on different PostgreSQL server instances to ensure isolation and correct measurements.

Component	**Parameter**
Processor	4 out-of-order cores
Core	2.66GHz, 4-way issue, 128-entry ROB
Branch predictor	Pentium M, 15 cycles penalty
L1-Instruction cache	32KB, 4 way, 4 cycle access time
L1-Data cache	32KB, 8 way, 4 cycle access time
L2 cache	256KB, private, 8 way, 8 cycle
L3 cache	8MB, shared, 16 way, 30 cycle
Main memory	45ns access time, 7.6GB/s per socket

Table 1: Simulated architecture parameters

In order to have a good insight of how the simulated queries behave, we chose to use a technique that employs cycle per instruction (CPI) stacks [SniperIISWC11]. The total cycle count for a running computer program can be divided into a base cycle count plus a number of cycle components that reflect lost cycle opportunities due to miss events such as cache misses. Cycle stacks are a good way to understand an application's behaviour providing a deeper insight into possible bottlenecks than what can be inferred from plain statistical data. Each cycle component represents the amount of execution time that can be attributed to that type of event. Table 2 lists the stack components we use later and a brief description of each. As an example, contributions related to the memory hierarchy (starting with mem-), indicate that a cache miss was resolved by that particular component.

In the figures that follow, we demonstrate a set of CPI stacks that represent an entire query execution and other stacks with respect to time for two selected queries, which help us understand the different execution phases that take place throughout the execution of a query.

A table with raw statistics is also provided for completeness. Due to simulation time constraints in our infrastructure, we could not complete the simulated execution of queries 1 and 18.

Component	Description
base	Base cycle count, core busy with no miss events.
depend-int	Amount of time the processor is stalled as a result of an integer instruction dependency.
depend-branch	Amount of time the processor is stalled as a result of a branch instruction dependency.
issue-port2	Amount of time stalled to issue a load request to memory.
branch	Amount of time lost due to branch mispredictions.
ifetch	Amount of time lost due to instruction cache misses.
mem-l1d	Amount of time spent waiting for data that resulted in an L1D hit.
mem-l2	Amount of time spent waiting for data that resulted in an L2 hit.
mem-l3	Amount of time spent waiting for data that resulted in an L3 hit.
mem-dram	Amount of time spent waiting for a DRAM access.

Table 2: Architectural components shown in CPI stack charts.

2.3 Evaluation results

2.3.1 CPI stacks of evaluated queries

Using cycle stacks allows us to easily identify performance bottlenecks for the executed queries, and help us draw meaningful conclusions that will drive the research to be done in the upcoming months.

Figure 1 shows the CPI stacks for the evaluated TPC-H queries. Overall it can be observed that the memory components (green) account for a large fraction of time. This fact is expected to be exacerbated when using larger databases. This suggests that most queries are memory bound, since the fraction of time where the CPU is busy with no miss events remains below 30%. As noted in previous research [ClearClouds12], instruction cache misses can be a significant performance bottleneck, this is because DBMS can have large instruction footprints that do not fit in the first level instruction caches. Moreover, due to the fact that nowadays large caches with large access times are used in the upper levels of the memory hierarchy, instruction misses of potentially performance-critical instructions are

further penalized [Scale-outISCA12].

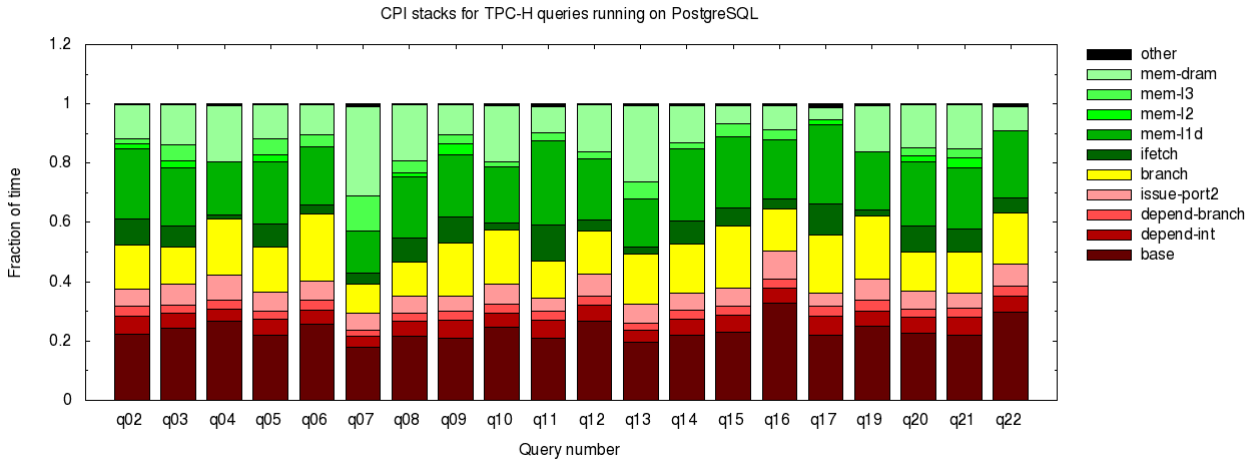

Figure 1: CPI stacks for evaluated queries.

Note that while the *base* component definition states that there are no miss events, it could be further improved by using more efficient ISA instructions or specialized accelerators that can perform the same operations in a reduced amount of time. Dependencies amongst instructions are currently not a critical bottleneck due to the memory-bound nature of the workloads, even though the simulated architecture can issue up to 4 instructions per cycle. However, these might need to be considered in the future if our improvements in the architecture put more pressure into a core's resources. Contention to issue load instructions is already a noticeable component (*issue-port2*) of the execution time, and is likely to become worse as the workload shifts to a more CPU intensive behaviour when making improvements in the memory hierarchy. Possible solutions might include additional ports in the caches to alleviate contention for load requests. Additionally, we observe that this kind of workloads have poor branch prediction hit rates throughout all query executions. The branch predictor is modeled after the Pentium M [BranchPredictor] branch predictor.

2.3.2 CPI stack histograms: Understanding query behaviour over time

In the previous subsection, we have shown breakdowns for each of the evaluated queries, however, these breakdowns do not help us to understand the behaviour of a query throughout its execution. For this purpose we make use of histograms of CPI stacks over time. This feature is also part of the Sniper infrastructure and is quite helpful in order to determine which kind of different phases these queries present, if any. We show the CPI stack with respect to time for two queries, q16 which is the most CPU intensive, and q07 which is the most memory intensive.

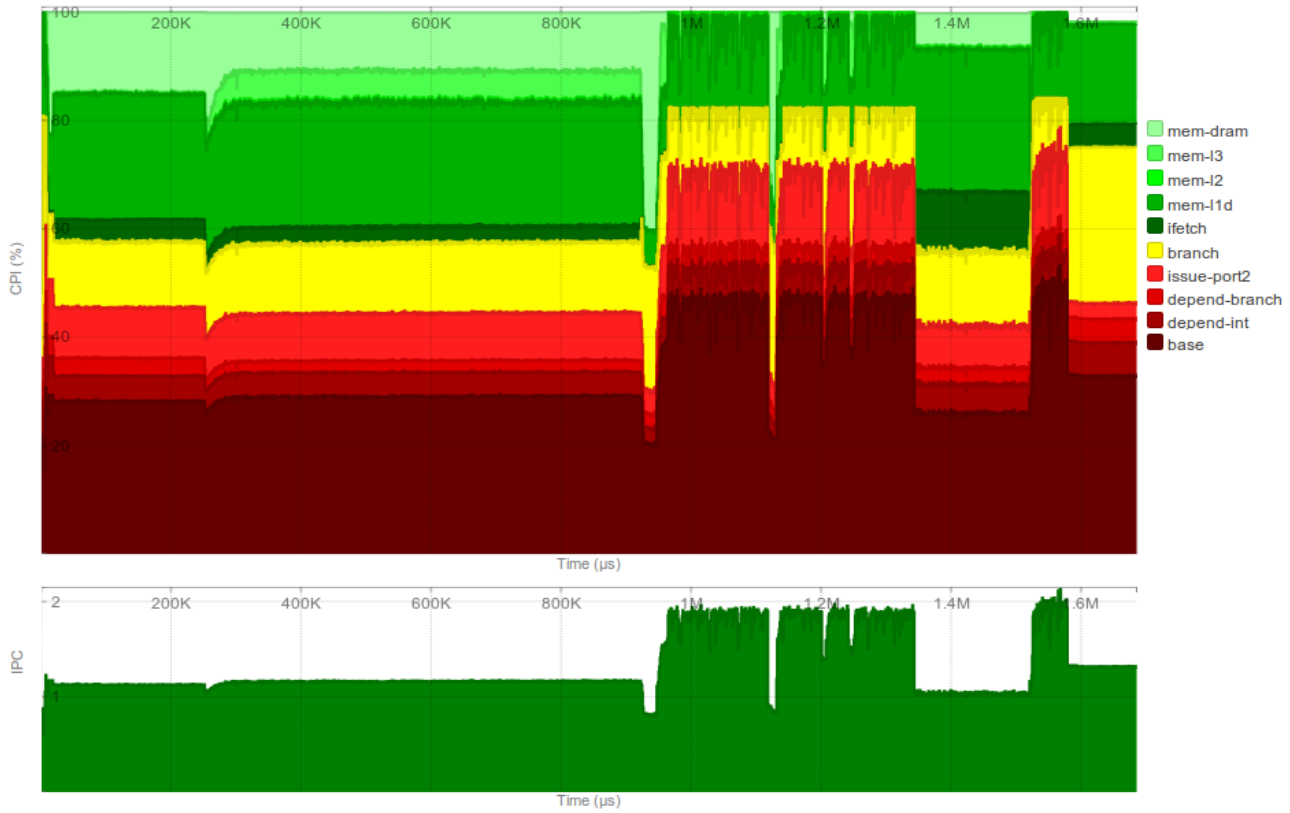

Figure 2: Normalized CPI (top) and IPC (bottom) with respect to time for query 16.

CPU-intensive query (q16)

Figure 2 shows the results for the query with the highest average CPU utilization, however, note that it is still memory bound during most of its execution time. After an initial phase of a near-flat instruction per cycle (IPC) rate of around 1.1, with high L1-Data cache usage, but still significant time spent in DRAM accesses, the execution changes to a more CPU intensive phase with an IPC of almost 2 instructions. The core has a theoretical maximum IPC of 4 (4-way issue core). We notice that when the core achieves an IPC of 2, the time spent in the memory hierarchy is almost entirely contained at the L1 cache level. L1-D utilization remains high throughout the execution, indicating that the amount of memory (load and store) instructions is high. The drop in performance at time 1.35M us (microseconds) is due to instruction cache misses (*ifetch*), suggesting that a different cache hierarchy might be able to improve such a scenario.

Memory-intensive query (q07)

Figure 3 shows a memory intensive query, spending a large percentage of the execution time servicing data from the DRAM subsystem. An initial phase can be observed where the IPC is low due to (1) instruction cache misses and (2) a significant amount of off-chip accesses to DRAM. Later, a second phase starts where an even lower IPC is observed (around 0.6) due

to an even larger number of accesses to main memory. In this second phase, instruction misses stop being a limiting factor to higher performance and over 60% of the execution cycles are spent in the upper levels of the memory hierarchy.

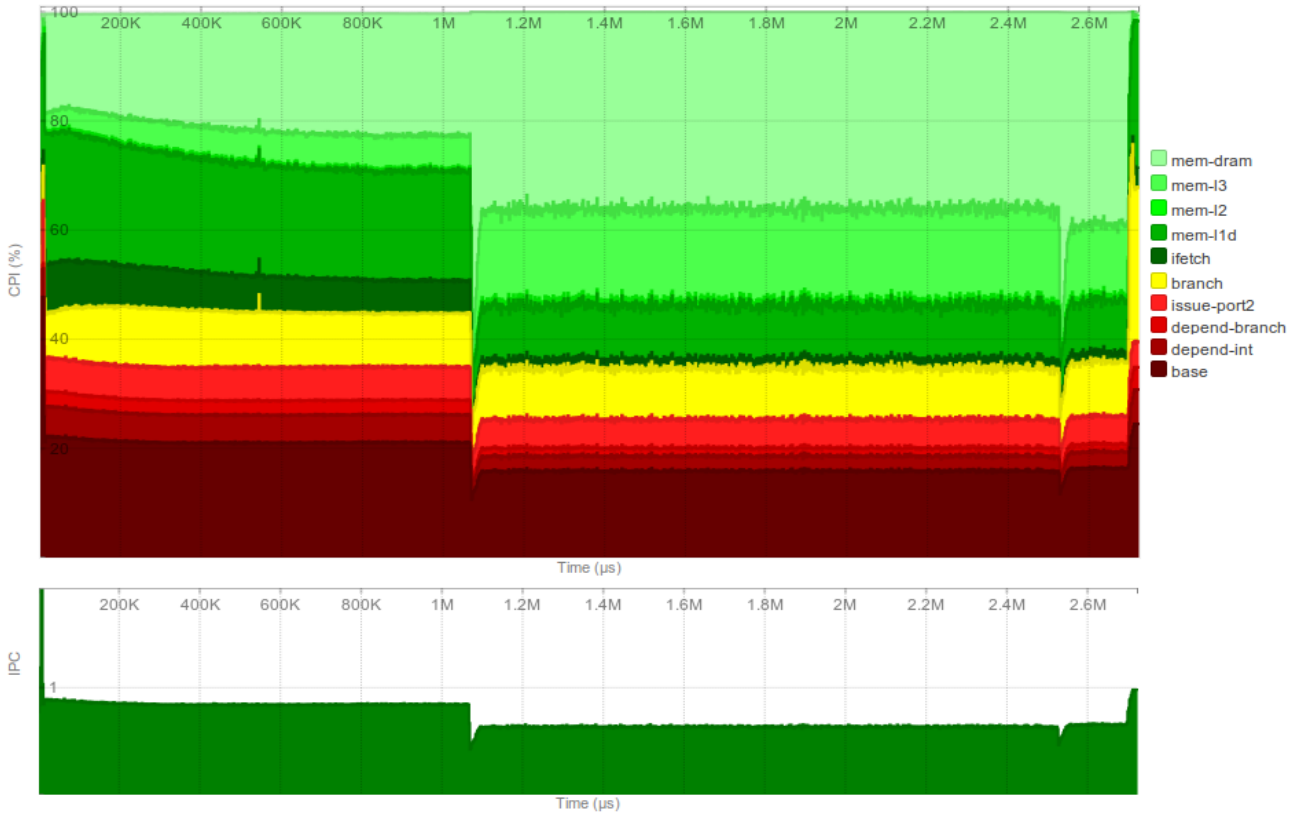

Figure 3: Normalized CPI (top) and IPC (bottom) with respect to time for query 7.

2.3.3 Performance statistics of evaluated queries

Table 3 shows a set of statistics for the executed queries of TPC-H: number of simulated instructions (in billions), instructions-per-cycle (IPC), real execution time (without simulation), branch prediction misses per kilo instruction (MPKI), L1 instruction and data caches MPKI, L2 and L3 MPKI, and DRAM accesses per kilo instruction (APKI). Note the large number of instructions that has been simulated, whereas usually a few hundreds of thousands of instructions are simulated in full-system simulators. However, all simulations even the ones with more than 20 billion instruction finished in less than 24 hours (queries 1 and 18 took more than 24 hours to complete). The average IPC is lower than 1, being rather low for an out-of-order core that can issue up to 4 instructions per cycle. Branch mispredictions remain quite high across all executions, and a recurring problem regarding L1 instruction misses can also be seen from the data, with up to 22MPKI in query 11. The number of DRAM accesses presents variation across queries ranging from 0.36 to 2.67 APKI, these numbers are likely to increase when using databases with larger data sets.

Query num.	Num. Instr.	IPC	Real time (s)	Branch MPKI	L1-I MPKI	L1-D MPKI	L2 MPKI	L3 MPKI	DRAM APKI
2	1.77B	0.89	0.75	4.88	14.82	2.95	1.81	1.19	1.73
3	6.90B	0.97	2.67	3.38	10.65	3.89	1.94	0.86	0.98
4	20.9B	1.07	7.39	3.82	1.46	1.42	1.40	1.18	1.50
5	2.66B	0.88	1.14	4.48	13.87	4.47	2.24	1.00	1.38
6	4.69B	1.03	1.72	5.24	3.75	1.68	1.52	0.67	0.81
7	5.25B	0.72	2.73	3.57	7.42	5.75	5.18	2.30	2.67
8	1.44B	0.86	0.63	3.46	14.47	4.72	2.53	1.38	1.70
9	20.3B	0.84	9.05	4.76	14.65	6.45	1.57	0.82	0.96
10	14.6B	0.98	5.57	5.03	3.37	2.48	1.51	1.11	1.23
11	0.63B	0.84	0.28	4.18	22.32	1.89	1.42	0.79	1.16
12	8.25B	1.07	2.90	3.33	4.49	2.11	1.77	1.09	1.48
13	9.74B	0.78	4.69	6.41	3.37	3.27	3.22	1.67	2.02
14	1.63B	0.88	0.70	4.91	12.53	2.19	1.75	1.02	1.40
15	4.66B	0.92	1.90	6.09	8.93	1.93	1.63	0.49	0.63
16	5.90B	1.31	1.69	2.83	3.21	1.16	0.98	0.43	0.53
17	1.93B	0.87	0.83	5.05	16.05	2.17	0.74	0.28	0.36
19	18.2B	1.00	6.85	5.13	2.43	1.27	1.06	0.91	0.96
20	0.87B	0.91	0.36	4.27	14.34	4.17	1.63	1.07	1.44
21	8.59B	0.88	3.65	4.31	12.43	5.76	1.83	1.08	1.23
22	2.04B	1.19	0.65	3.67	5.11	1.44	0.50	0.43	0.51

Table 3: Statistics of simulated queries

3. Future work: incorporating NVRAM and 3D stacking

One of the objectives of the AXLE project is to evaluate how the database engines and the query executions would behave using emerging memory technologies, such as Non-Volatile

RAM Memory and 3D stacking. The performance of a DBMS with each type of novel memory technology can help adapt DBMS for better performance in the future.

Non-Volatile RAM (NVRAM), such as ferroelectric RAM, magnetoresistive RAM, or phase-change memory (PCM), is an emerging and radical trend seen as a promising solution for future extremely large volumes of data. In contrast with the regular RAMs, NVRAM does not lose its contents when its power is turned off. As such, NVRAM provides long term data retention just as current hard disks do. In addition, the NVRAM technology scales very well with the shrinking CMOS technology nodes. However, the latency and the bandwidth of NVRAM solutions vary significantly between various technologies and the levels of maturity. The slowest NVRAM solutions are on the level of flash memories, and the fastest NVRAM solutions can be compared to fast SRAM caches present in processors.

3D stacking allows the integration of two or more layers of active electronic components. It is not yet widely used, mostly due to technological issues such as yield, heat, and the design and testing complexity of chips with 3D stacking. However, many hardware vendors are pushing the advancements in 3D stacking, in the hopes of solving many of the problems of today's chips, such as reducing the footprint, reducing the cost and increasing the yield, decreasing the power usage, allowing new design possibilities and, most importantly, offering a higher bandwidth of data transfers.

Emerging NVRAM memory technologies could significantly simplify the implementation of DBMS, since the data does not have to be committed to permanent storage. NVRAM and 3D stacking can reduce the latencies and improve the bandwidth. This can directly accelerate the execution of unmodified DBMS.

The emerging NVRAM technologies offer speeds that fall between DRAM and flash drive ranges. 3D stacking allows mixed memory technology systems, where regular DRAM and emerging NVRAM are used on the same chip but on different layers. Bearing in mind that RAM chips are one of the most significant sources of power dissipation, this can allow powering off some layers of the chip, and reduce the power consumption.

Using the insights gathered in this report and after further profiling, we will focus on exploring different memory hierarchies to improve execution of DBMS by using the mentioned technologies. Of special interest will be hybrid memory systems that combine DRAM (fast but non-scalable) and different emerging NVRAM technologies (slower, scalable, and non-volatile) architected to exploit the best characteristics each technology provides. Reduction of instruction cache misses and data placement closer to processing cores, by using 3D stacking technologies, can provide significant performance improvements.

4. References

[SniperSC11] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations,” in International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Nov. 2011.

[ZsimISCA13] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: fast and accurate microarchitectural simulation of thousand-core systems. In Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13). ACM, New York, NY, USA, 475-486.

[SniperISPASS13] Trevor E. Carlson and Wim Heirman and Lieven Eeckhout, Sampled Simulation of Multi-Threaded Applications, International Symposium on Performance Analysis of Systems and Software (ISPASS 2013), pp 2-12, apr 2013.

[SniperIISWC11] Wim Heirman; Trevor E. Carlson; Shuai Che; Kevin Skadron; Lieven Eeckhout. Using Cycle Stacks to Understand Scaling Bottlenecks in Multi-Threaded Workloads. International Symposium on Workload Characterization (IISWC), Nov. 2011.

[ClearClouds12] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware", In Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII). ACM, New York, NY, USA, 37-48, 2012.

[Scale-outISCA12] Lotfi-Kamran, Pejman; Grot, Boris; Ferdman, Michael; Volos, Stavros; Kocberber, Onur; Picorel, Javier; Adileh, Almutaz; Jevdjic, Djordje; Idgunji, Sachin; Ozer, Emre and Falsafi, Babak, "Scale-Out Processors", in 39th Annual International Symposium on Computer Architecture (ISCA), Portland, Oregon, USA, June 9-13, 2012.

[Graphite] Jason E. Miller, Harshad Kasture, George Kurian, Charles Gruenwald III, Nathan Beckmann, Christopher Celio, Jonathan Eastep and Anant Agarwal, "Graphite: A Distributed Parallel Simulator for Multicores", the 16th IEEE International Symposium on High-Performance Computer Architecture (HPCA), Jan 2010.

[Pin] Pin - A Dynamic Binary Instrumentation Tool, Intel Corp. http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[TPCH] http://www.tpc.org/tpch/

[Gem5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A.

Wood. 2011. The gem5 simulator. SIGARCH Comput. Arch News 39, 2 (August 2011), 1-7.

[BranchPredictor] V. Uzelac and A. Milenkovic. Experiment flows and microbenchmarks for reverse engineering of branch predictor structures. In Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 207–217, 2009.