Accelerates In-Memory Databases with Near Data Computing

Kevin Hsieh
kevinhsieh@cmu.edu

Amirali Boroumand
aborouma@andrew.cmu.edu

1. Problem and Motivation

Modern databases, such as H-Store[14], VoltDB[23], and Masstree[20], have been moving from disk to main memory because cheaper commodity DRAMs are orders of magnitude faster. Furthermore, DRAM-based key-value storage servers, such as memcached[9] and Redis[22], have been known as critical parts of major Internet services.

All the paradigm shifts from disks to DRAMs have made the bandwidth and latency of DRAMs, or memory wall[26], become the emerging bottleneck of data centers. Conventional databases are known to spend at least 50% of their execution time on memory stalls[3]. This problem is getting worse for in-memory databases. For example, the profiling of Masstree showed that the get operations spend 66%-75% of time on DRAM stalls[20].

While DRAMs have kept the scaling of cost-per-bit, their latency has remained almost constant[18]. In addition, many studies showed that DRAM systems accounts for 25-40% of total power consumption in data centers[4][19]. A promising technology to tackle these challenges is 3D-stacked memory, such as Hybrid Memory Cube (HMC) [13]. By stacking DRAM and logic dies with through-silicon vias (TSVs), this technology revives the idea of processing in memory (PIM)[8][10][21][17]. One can build a high performance and low power in-memory system by moving data-intensive computations to the memory side.

However, the key challenge of PIM is the constraint on cost and thermal[28]. A highly specialized PIM is not desirable because it may not be useful for most applications and can not reach commodity. A powerful general processor on the memory side is intractable due to the power consumption generates unmanageable heat. It is crucial to find general and cost efficient functions to enable PIM.

2. Idea

In this project, we aim to characterize the in-memory databases and investigate the feasibility to accelerate their bottlenecks with PIM. Databases are one of the most memory intensive, widely used applications. As they spend most of their time on DRAM stalls, they could be accelerated by PIM significantly. The key idea is no matter how databases are designed, they all need general functions such as index traversals, partitioning, sorting, and filtering. These functions are highly possible to be bounded by DRAM because their low spatial locality and massive working sets. As the computation of these operations is general simple, it is possible to design an array of these PIM functions to parallelize them and hide the memory latency. Doing these operations on the memory side can minimize the data movements over wires, which costs 23X energy of double-precision fused-multiply add operation in 10nm[15]. Processing bulk data on the memory side can also significantly reduce cache thrashing on the processor side.

3. Related Works

Several prior studies analyzed the bottlenecks of databases and proposed specialized accelerators to speed them up. The most common benchmarks are TPC-C for online transaction processing (OLTP) and TPC-H for online analytical processing (OLAP)[2]. Most OLAP related works used MonetDB[12], an in-memory, column-store database. On the other hand, the OLTP related works are based on Shore[7] or DBx1000[27]. According to their focus, they can be categorized as join operations, relational operators, and concurrency controls.

The analysis of MonetDB found that join operation accounts for 47% of the TPC-H execution time[24]. A hardware data partitioner was proposed to accelerate the data partition of join operations by 7.8X[24]. A set of specialized walker units were proposed to improve the hash index of join operations, which can be improved by 3.1x[16].

Another family of accelerators speed up the relational operators that manipulate database columns or tables. Q100 is a specialized database processing unit that can improve the performance of 24-thread software database by 1.5X-2.9X[25]. Commercial data warehouse have been using FPGA to accelerate the database query, such as IBM DB2 Analytics Accelerator (IDAA)[6].

The concurrency control of databases would emerge as a new bottleneck when running on many-core chips. Recent study showed that classic concurrency controls can not scale well to a 1024-core system, where databases spend most of the time on lock manager or abort[27]. A hardware-based lock manager may be needed at that scale.

The analysis of Shore showed that this OLTP database spend around 80% of time on logging, locking, latching, and buffer management. A single-threaded, lock-free, main memory database system without recovery can be up to 20x faster[11].

Our project is different from these prior works as we are trying to accelerate general operations as PIM instead of specialized accelerators in processors or FPGAs. The key difference is we need to make it general and simple enough so that is can be stacked with memory dies. In addition, our work distinguishes from them because we can fully utilize the extremely abundant memory bandwidth as there are no external wire constraints for PIM. These different design considerations will lead to unique optimal designs.

4. Logistics

4.1. Project plan

In this project, our goal is to characterize in-memory database and find the potential bottlenecks that could be sped up by shipping computation to the memory side. To begin with, we need to choose a DBMS that suits our constraints such as time-budget. Our plan is to characterize DBx1000[27] to identify potential data structure operations that could be accelerated by PIM. DBx1000 is a single node OLTP database management system (DBMS). The original purpose of DBx1000 was to make DBMS scalable on future 1000-core processors. We choose DBx1000 because it is relatively simple compared to other DBMSs. It was implemented from scratch and removed several artificial bottlenecks in conventional DBMSs. This makes it suitable for our purpose, especially considering our time-budget and simulation infrastructure. We believe that DBx1000 could be used as a representative example of other DBMSs, so our analysis result can be applicable to others as well.

As the first step, we need to run DBx1000 on a real machine and profile it using Perf[1]. Perf is a profiler tool for Linux 2.6+ based systems that abstracts away CPU hardware differences in Linux performance measurements and it can instrument CPU performance counters and tracepoints. The purpose of profiling is to find bandwidth and latency sensitive computational patterns that account for large fraction of the execution time. The profiling outcome will enable us to design one or multiple PIM functions that could accelerate a wide range of bottlenecks in DBx1000.

After designing the PIM functions, we need to evaluate our proposed design to demonstrate the performance improvement. Conducting evaluation in a real machine is not feasible because it requires PIM to be implemented in a 3D-stacked memory, which is an extremely time-consuming process. Thus, we plan to examine our proposed PIM function design using a full system simulator. We will use GEM5[5] full-system simulator and run DBx1000 on the simulated system with PIM. GEM5 is a modular event driven full-system simulator which supports various instruction set architecture such as x86, ARM, SPARC, MIPS, Alpha and PowerPC.

4.2. Milestones

Based on our plan, the project milestones are as follows: By milestone 1 (Nov 1), we will prepare the simulation infrustructure and we will finish the profiling of DBx1000. By milestone 2 (Nov 21), we plan to design the PIM function to accelerate the bottlenecks identified by profiling result, and we will the evaluate it using the simulator.

4.3. Goals

The project goal is to analyze in-memory databases and investigate the feasibility of designing one or multiple PIM functions which could reduce the execution time of those in-memory databases. While the in-memory DBMSs spend most of their time on DRAM stalls, it is still possible that these stalls are evenly distributed among different modules of the DBMSs. In that case, hardware acceleration can improve limited amount of computations and DBMSs might not benefit too much from shipping computation to the data. Thus, our 75% goal is to characterize DBx1000, design the PIM function, implement the proposed PIM function in the GEM5 simulator and achieve 10% performance improvement. Our 100% goal is that our proposed PIM function can reduce DBx1000 execution time by 30%. Finally, Our 125% goal is to characterize other databases such as MonteDB and attempt to find out how our proposed PIM functions could be applied to them.

4.4. Final Report Outline

In this section, we provide an outline of our final report. The outline will contain the following sections:

- The problem
- Background
  - Processing in memory
  - DBMS profiling and hardware acceleration
- The profiling result of DBx1000 (or MonetDB)
- The architecture of the proposed PIM design
- Evaluation
  - Performance evaluation
  - Sensitivity analysis
    - Different system configurations (core number, cache size)
    - Different size of database working set
- Conclusion

4.5. Required Recources

Our required recources are as follows:

- GEM5 full system simulator, which is a free, open source software.
- DBx1000, which is an open source software.
- Perf profiling tool. It is available in Linux kernel.
- The computation server to run DBx1000 on GEM5. It is available within our group.

References

[1] "Perf." [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[2] "Transaction processing performance council," *Web Site, http://www.tpc.org*, 2005.

[3] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a modern processor: Where does time go?" in *VLDB" 99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, no. DIAS-CONF-1999-001, 1999, pp. 266–277.

[4] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[5] N. Binkert, B. Beckman, A. Saidi, G. Black, and A. Basu, “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, 2011.

[6] P. Bruni, P. Becker, W. Favero, R. Kalyanasundaram, A. Keenan, S. Knoll, N. Lei, C. Molaro, P. Prem *et al.*, *Optimizing Db2 Queries with Ibm Db2 Analytics Accelerator for Z/os*. IBM Redbooks, 2012.

[7] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. Tan, O. G. Tsatalos *et al.*, *Shoring up persistent applications*. ACM, 1994, vol. 23, no. 2.

[8] D. G. Elliott, W. M. Snelgrove, and M. Stumm, “Computational RAM: A memory-SIMD hybrid and its application to DSP,” in *Custom Integrated Circuits Conference*, vol. 30. Citeseer, 1992, pp. 1–30.

[9] B. Fitzpatrick, “Distributed caching with memcached,” *Linux journal*, vol. 2004, no. 124, p. 5, 2004.

[10] M. Gokhale, B. Holmes, and K. Iobst, “Processing in memory: The Terasys massively parallel PIM array,” *Computer*, vol. 28, no. 4, pp. 23–31, 1995.

[11] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker, “Oltp through the looking glass, and what we found there,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 981–992.

[12] S. Idreos, F. Groffen, N. Nes, S. Manegold, S. Mullender, and M. Kersten, “MonetDB: Two decades of research in column-oriented database architectures,” *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 35, no. 1, pp. 40–45, 2012.

[13] J. Jeddeloh and B. Keeth, “Hybrid memory cube new DRAM architecture increases density and performance,” in *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 2012, pp. 87–88.

[14] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. Jones, S. Madden, M. Stonebraker, Y. Zhang *et al.*, “H-store: a high-performance, distributed main memory transaction processing system,” *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1496–1499, 2008.

[15] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, “GPUs and the future of parallel computing,” *Micro, IEEE*, vol. 32, no. 5, pp. 7–17, 2011.

[16] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, “Meet the walkers: Accelerating index traversals for in-memory databases,” in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 468–479.

[17] P. M. Kogge, “EXECUBE-A new architecture for scaleable MPPs,” in *Parallel Processing, 1994. Vol. 1. ICPP 1994. International Conference on*, vol. 1. IEEE, 1994, pp. 77–84.

[18] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, “Tiered-latency DRAM: A low latency and low cost DRAM architecture,” in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 615–626.

[19] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, “Understanding and designing new server architectures for emerging warehouse-computing environments,” in *Computer Architecture, 2008. ISCA’08. 35th International Symposium on*. IEEE, 2008, pp. 315–326.

[20] Y. Mao, E. Kohler, and R. T. Morris, “Cache craftiness for fast multicore key-value storage,” in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 183–196.

[21] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, “A case for intelligent RAM,” *Micro, IEEE*, vol. 17, no. 2, pp. 34–44, 1997.

[22] S. Sanfilippo and P. Noordhuis, “Redis,” 2009. Available: http://redis.io

[23] M. Stonebraker and A. Weisberg, “The VoltDB Main Memory DBMS.” *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 21–27, 2013.

[24] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross, “Navigating big data with high-throughput, energy-efficient data partitioning,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ACM, 2013, pp. 249–260.

[25] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross, “Q100: the architecture and design of a database processing unit,” in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014, pp. 255–268.

[26] W. A. Wulf and S. A. McKee, “Hitting the memory wall: implications of the obvious,” *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[27] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker, “Staring into the abyss: An evaluation of concurrency control with one thousand cores.”

[28] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. Greathouse, M. Meswani, M. Nutter, and M. Ignatowski, “A new perspective on processing-in-memory architecture design,” in *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*. ACM, 2013, p. 7.