The bionic DBMS is coming, but what will it look like?

Ryan Johnson
University of Toronto
ryan.johnson@cs.utoronto.ca

Ippokratis Pandis
IBM Almaden Research Center
ipandis@us.ibm.com

ABSTRACT

Software has always ruled database engines, and commodity processors riding Moore's Law doomed database machines of the 1980s from the start. However, today's hardware landscape is very different, and moving in directions that make database machines increasingly attractive. Stagnant clock speeds, looming dark silicon, availability of reconfigurable hardware, and the economic clout of cloud providers all align to make custom database hardware economically viable or even necessary. Dataflow workloads (business intelligence and streaming) already benefit from emerging hardware support. In this paper, we argue that control flow workloads—with their corresponding latencies—are another feasible target for hardware support. To make our point, we outline a transaction processing architecture that offloads much of its functionality to reconfigurable hardware. We predict a convergence to fully "bionic" database engines that implement nearly all key functionality directly in hardware and relegate software to a largely managerial role.

1. INTRODUCTION

Although database systems have flirted with custom hardware for decades, economic realities have historically favored purely software solutions riding on Moore's Law. However, a confluence of several trends is changing this.

First, processor clock speeds—already stagnant for a decade—are unlikely to increase any time soon due to power concerns. The industry reaction, to stamp out myriad identical cores, leads to difficulties with Amdahl's Law [6].

Second, transistor power scaling has reached a cross-over point: power savings due to smaller transistors no longer compensates fully for the increased number of transistors per chip. Fixed power envelopes will drive an ever-shrinking fraction of the chip at any given moment, a phenomenon known as *dark silicon* [3]. Performance gains in the future will come by utilizing transistors more effectively, rather than by utilizing more of them on as we do today, and homogeneous multicore designs will fall out of favor.

Finally, the recent explosion of cloud-based "big data" applications means that, for the first time, providers such as Google, Facebook, Amazon, and Microsoft have both the incentive and the economic clout to commission *custom hardware* if it lowers overall hardware costs or otherwise gives some edge over other providers [5].

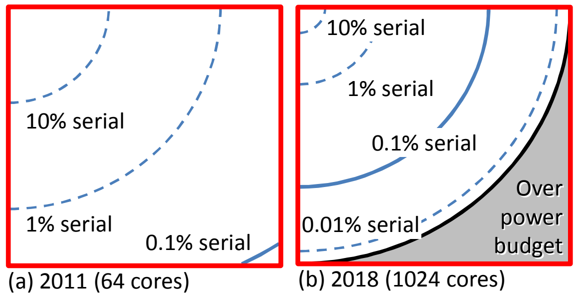


Figure 1: Fraction of chip (from top-left) utilized at various degrees of parallelism.

The community is already responding to these trends, and recent years have seen a variety of efforts, both commercial (including IBM Netezza and Oracle Exadata) and academic [9], to enhance data management operations with custom or semi-custom hardware. These efforts generally focus on *dataflow-style* computations which are broadly used in the business intelligence and streaming domains. Several important database applications, however, are left out because they are not based on dataflow computations; instead they are dominated by *control flow* and *high-latency operations* such as pointer chasing. The most prominent example of such application is online transaction processing (OLTP).

In this paper, we argue that custom hardware, and reconfigurable hardware in particular, also holds promise for computations dominated by control flow and high-latency operations, computations that are broadly used in transaction processing and graph traversals. In particular, using the recently proposed data-oriented architecture [10, 11] as a starting point, we briefly highlight the main sources of latency and software overhead in the system and explore the potential for hardware support to mitigate these problems. Along the way, we make two claims:

- Effective hardware support need not always increase raw performance; the true goal is to reduce net energy use.
- A surprising fraction of database operations are amenable to hardware implementation; we predict that future systems will mostly use software to coordinate the use and interaction of the available hardware units.

2. DARK SILICON

Simply put, dark silicon is the trend for an increasing fraction of the chip's transistors to remain unused at any given

6th Biennial Conference on Innovative Data Systems Research (CIDR'13)
January 6-9, 2013, Asilomar, California, USA.
.

time. The effect arises from two sources. First—ignoring power scaling for a moment—homogeneous multicore designs demand exponentially growing parallelism from the software. Although many important data management tasks are embarrassingly parallel, not all are. Further, even embarrassingly parallel tasks suffer from skew and imbalance effects as the data are spread across more and more cores. Either way, a seemingly innocuous serial component of work can lead to surprisingly large opportunity loss, as illustrated in Figure 1. The figure highlights how the fraction of hardware utilized (represented as the area from top-left to each labeled line) varies with available parallelism. Where achieving 0.1% serial work arguably suffices for today's hardware (a), next-generation hardware with perhaps a thousand cores demands that the serial fraction of work decreases by roughly two orders of magnitude. This is precisely the problem that motivated Amdahl to formulate his famous law, as an argument *against* an over-reliance on parallelism for performance. The second problem, the lack of power scaling in next-generation transistors, means that power constraints will force a growing fraction of hardware offline even if the software could have used it. A conservative calculation puts perhaps 20% of transistors outside of the 2018 power envelope, with the usable fraction shrinking by 30-50% each hardware generation after.

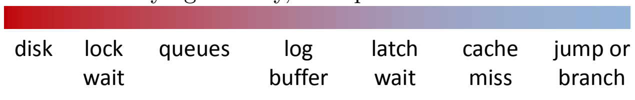
Performance is measured in joules/operation in the dark silicon regime, with performance (latency) merely a constraint. Making a computation use one tenth the power is just as valuable as making it ten times faster: both free up 90% of previously-expended joules for other uses or to be reclaimed as lower operating costs.

Because modifying already-optimized software has little impact on its power footprint [14], an operation's power footprint can only be reduced significantly by changing the hardware that runs it. Moving to leaner general-purpose hardware helps significantly, and several efforts are underway to run server workloads on embedded processors [2]. Unfortunately, even this extreme step gives only a one-time boost: a general purpose core can only be made so small, and Amdahl's law makes it difficult to utilize hundreds or thousands of extremely lean cores [6]. Once the move to embedded cores is complete, joules/op can only be reduced by moving to specialized or custom hardware.

3. LATENCY AND CONTROL FLOW

Query processing—particularly in columnar form—produces significant dataflow and relatively straightforward control flow that maps nicely to hardware. Transaction processing, on the other hand, features heavy control flow and small, irregular dataflow components (usually of the pointer chasing variety); they make notoriously inefficient targets for mainstream processors [1] and do not lend themselves to any obvious form of hardware acceleration. OLTP benefited from clock frequency scaling for decades, and currently benefits even more strongly from the high degree of parallelism offered by multicore hardware, but will suffer a severe performance disadvantage under dark silicon: fixed clock frequencies and fixed core counts across successive generations threaten to permanently cap OLTP throughput.

OLTP tends to be latency-bound, with multiple latency sources of varying severity, as depicted below:



The "big" delays like disk I/O are well known and fairly easy to schedule around in software, but the small delays at the other end of the scale inflict a kind of "death by a thousand paper cuts" that software and general-purpose hardware are ill-equipped to deal with.

Rather than attempting to increase raw performance, we submit that OLTP will benefit most from hardware that reduces its power footprint and helps hide or avoid as many latencies as possible. Techniques that avoid the myriad fine-grained latencies will be especially useful. An asynchronous and predictable delay of several μs is vastly easier to schedule around in software than an unexpected cache miss or pipeline stall; thoughput will improve, even if individual requests take just as long to complete. Reducing power footprint will free up power budget for other, more useful work, and a sufficiently efficient OLTP engine could even run on the same machine as the analytics, allowing up-to-the-second intelligence on live data.

4. CONTROL FLOW IN HARDWARE

Custom hardware has a reputation for poor handling of control flow, mostly from failed attempts to extract dynamic control flow from general purpose programs. Hardware actually excels at control flow, as evidenced by the ubiquitous finite state automaton, particularly non-deterministic finite state automata (NFA), which employ hardware parallelism to great effect. Viewing von Neumann-style control flow as a low-dimensional projection of some underlying state machine (with parallel threads of control being the dimensions), it becomes clear why mapping software back to hardware usually confers few benefits. Effective acceleration of control flow requires identifying the underlying abstract operation and mapping a high-dimensional projection to hardware. To give one concrete example, good regular expression matching and XPath projection algorithms employ NFA, whose fine-grained parallelism is easily captured in hardware [13] but leads to extremely inefficient software implementations; the majority of regexp libraries use inferior algorithms that map more cleanly to software.

Our initial exploration of common OLTP operations such as B+Tree probes and logging suggest that much of the inefficiency inherent to OLTP arises due to their poor mapping to software; the following sections explore this observation in more detail. We propose to target these software inefficiency hotspots by examining the underlying operations and mapping them directly to hardware, thus avoiding fidelity loss due to starting with the software implementation.

5. "BIONIC" TRANSACTION PROCESSING

As a concrete target system, we consider the Convey HC-2 machine,[1] which combines a field-programmable gate array (FPGA) with a modern Intel processor. The system architecture is depicted in Figure 2. It features a high-performing FPGA with direct access to disk and to a local memory pool; the FPGA-side memory is uncached, but its "scatter-gather" memory controllers deliver $80GBps$ bandwidth for random 64-bit requests, especially helpful for workloads with poor locality. FPGA and host-side memory are coherent and accessible by either CPU or FPGA, though the PCI bus imposes severe NUMA effects ($2\mu s$ round-trip). These characteristics dictate that the FPGA handle most data manipulation, and that CPU/FPGA communication must be asynchronous. The PCI bus provides ample bandwidth to support OLTP workloads, and Netezza-style filtering at the FPGA should ease bandwidth concerns for queries.

[1] http://www.conveycomputer.com

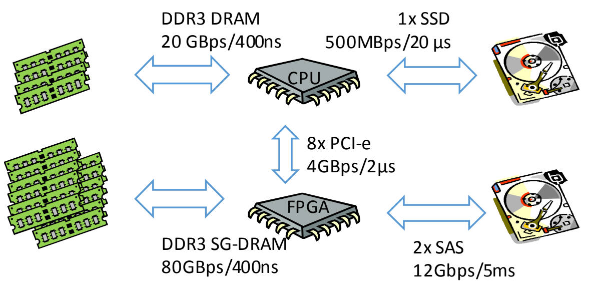


Figure 2: A high performance CPU/FPGA platform

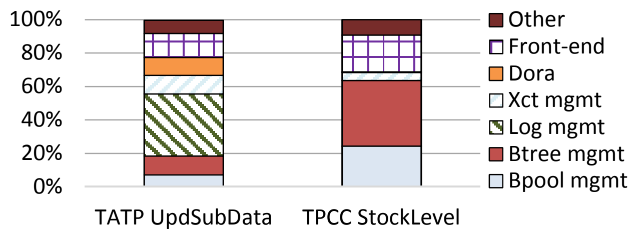


Figure 3: Time breakdown of a highly-optimized transaction processing system running two types of transactions on a conventional multicore system.

Again, the goal is not to reduce latency directly, but rather to bundle up myriad small sources of latency and offload them to a power-efficient, asynchronous medium.

Creating an purely hardware OLTP engine is infeasible (or at least uneconomical) for several reasons, not least of which is the variety and dynamic nature of transactional workloads. We therefore propose a hybrid software-hardware architecture that implements particularly important operations to hardware, with software coordinating their use and providing fallbacks for unimplemented corner cases. Due to their complexity, traditional DBMS architectures will make challenging targets for this transformation; in this paper we assume a data-oriented architecture [10, 11] as a starting point because it eliminates several sources of complexity and incorporates a system of queues that will prove useful in overlapping the latency imposed by hardware-software communication. We give a brief overview of DORA later.

5.1 Bottleneck analysis

The data-oriented architecture (DORA) allows a fully shared-everything transaction processing system to gain most of the benefits available from partitioning datasets, but without the data movement usually required with partitioning. DORA divides the database into logical partitions backed by a common buffer pool and logging infrastructure, and then structures the access patterns of threads so that at most one thread touches any particular datum. A full implementation eliminates locking and latching from the majority of code paths, replacing them with a significantly simpler arrangement of queues and rendezvous points and exploiting careful placement of data to pages [11].

In Figure 3 we examine the time breakdowns for an update (TATP UpdateSubData) and a read-only (TPC-C StockLevel) workload executing in a recent DORA prototype. The remaining overheads fall into four main categories: (a) B+tree index probes; (b) Logging; (c) Queue management and (d) Buffer pool management.

5.2 Architectural overview

Based on the bottlenecks identified in the previous section, we propose an architecture that offloads four major operations to hardware: tree probes, overlay management (details follow), log buffering, and queue management. We also assume a Netezza-style engine implements selections and projections for queries to reduce bandwidth pressure on the PCI bus. Higher level concerns, such as query execution, scheduling and routing, recovery, and index re-org, stay in software. Figure 4 illustrates these components and their interactions, with software at the top, hardware in the middle, and storage below. We exploit the non-uniform paths to storage by maintaining database files (and cached database content) on the FPGA side, while the CPU side stores log files on a fast SSD and temporary/intermediate results in memory.

We now discuss briefly each piece of hardware support and the types of latency it helps address.

5.3 B+Tree probes

OLTP workloads are index-bound, spending in some cases 40% or more of total transaction time traversing various index structures (e.g. Figure 3 (right)). Overhead could be reduced significantly by incorporating cache-friendly data structures and letting higher-level code handle concurrency control. However, the random nature of tree traversals will leave even the simplest software implementation latency bound, barring use of complex measures such as PALM [12].

A hardware solution is feasible and attractive for several reasons. First, virtually all concurrency control issues are resolved before a request ever reaches the tree, eliminating by far the largest source of complexity in the data structure. B+Tree operations are typically logically logged, so software can deal with the logging subsystem as long as the hardware unit guarantees the atomicity of any request it receives. High node branching factors mean that the entire index fits in memory for most datasets, so the hardware can rely on software for disk accesses and abort any operations that fall out of memory. Even if an index is too large to fit in memory, the inodes tend to still fit comfortably, particularly if leaves are sized for disk access (leading to branching factors of several hundred to a few thousand). Finally, giving a pipelined tree probe unit direct access to memory (bypassing the cache) should allow the unit to saturate using only perhaps a dozen outstanding requests, with no need for those requests to arrive simultaneously; the Convey SG-DRAM delivers high throughput even for pointer chasing. Initial experiments suggest that the proposed hardware unit would be extremely compact: software for probing the above B+Tree requires only a few dozen machine instructions, mostly triplets of the form "load-compare-branch." This kind of control flow maps extremely well to hardware, and we are currently in the process of building a generic hardware tree probe engine that can handle both integer and variable-length string keys.

Other complex operations, such as space allocation, inode splits, and index reorganization, are handled in software.

5.4 Logging

The DORA system eliminates most locking (with the remainder being thread-local), leaving the database log as the main centralized service. Although log contention can be alleviated for single-socket systems with some considerable effort, multi-socket systems remain an open challenge due to socket-to-socket communication latencies [7].

A hardware logging mechanism would have two significant advantages over the software version. Requests from the same socket can be aggregated before passing them on, and

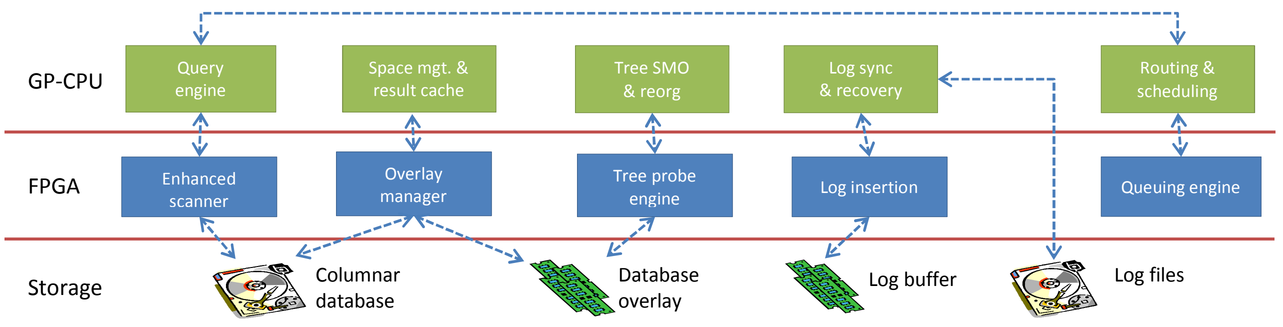


Figure 4: A "bionic" hardware-software hybrid DBMS architecture

hardware-level arbitration is significantly simpler to reason about than a typical lock-free data structure, while avoiding the complexities and overheads normally required to build a serial log using latches. For maximum effectiveness, the logging interface would need to be asynchronous, so that the latency of various log operations can overlap easily with other requests. Log synchronization can be done in software, with some assistance from the hardware queue management engine to keep the latency off the critical path.

We also note that efficient logging infrastructure could prove useful outside the database engine; high performance logging file systems are another obvious candidate.

5.5 Queue management

DORA uses queues extensively, to impose regularity on access patterns, eliminate contention hotspots, and hide latencies due to partition crossing and log synchronization. The queues in DORA usually see only light contention at worst, but they still have significant management overhead (which is part of the Dora and front-end components in Figure 3). The main challenges are scheduling-related, such as selecting the number of queues to use, assigning them appropriate owners, and knowing when to deschedule an idle agent thread with an empty input queue (a wrong choice can hold up an entire chain of queues, leading to convoys).

A wide space of potential hardware support surrounds queuing. Extensions to cache coherency protocols; resurrecting message-passing systems of yesteryear; implementing proposals such as QOLB [8], etc, with the goal to eliminate as much overhead as possible without overly restraining the software. We note that many of the challenges associated with queues are fundamentally hard; while hardware will undoubtedly reduce overheads, it will not magically solve the scheduling problem. We expect that software will continue to play a key role in this area.

5.6 Overlay database

Rather than a buffer pool, the bionic system would employ two data pools. The CPU side maintains a cache of intermediate results and other "cooked" data, while the FPGA side maintains an in-memory overlay of the database. The overlay serves to cache reads and to buffer writes until they can be bulk-merged back to the on-disk data (replacing the buffer pool), and will also patch updates into historical data requested by queries; SAP HANA [4] is an excellent example of this approach. Recognizing that OLTP workloads are heavy index-users, the overlay will consist entirely of various indexes that can be probed by the hardware engine. If disk access is needed, the hardware operation aborts so that software can trigger a data fetch and then retry.

6. CONCLUSIONS

Dark silicon is here. Hardware software co-design is unavoidable. In this paper we made the case of a mostly hardware-based transaction processing implementation. The preceding sections sketch a system architecture where every latency source has hardware support for avoiding or offloading it, so that software can continue with something else rather than blocking. This applies from I/O requests all the way down to cache misses. We have already seen good results with pipelined log writes [7], and web servers have honed this strategy to near-perfection [16].

Significant challenges remain: not only to design hardware (hard!), but also to re-architect the software to exploit it. Initial attempts by others to automate the process have not gone particularly well [15], and database engines have a long legacy of being impervious to hardware-only changes. Overcoming this challenge presents a huge opportunity for data management, however, because it already insulates its users from disruptive hardware changes. More ad-hoc approaches will face steep disadvantages in this regard.

References

[1] Ailamaki, A., DeWitt, D., Hill, M., and Wood, D. DBMSs on a modern processor: Where does time go? In *VLDB*, 1999.
[2] Andersen, D. G., et al. FAWN: a fast array of wimpy nodes. In *SOSP*, 2009.
[3] Esmaeilzadeh, H., et al. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.
[4] Färber, F., et al. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1), 2012.
[5] Hamilton, J. R. Internet scale storage. In *SIGMOD*, 2011.
[6] Hill, M. D. and Marty, M. R. Amdahl's law in the multicore era. *Computer*, 41, 2008.
[7] Johnson, R., et al. Scalability of write-ahead logging on multicore and multisocket hardware. *The VLDB Journal*, 20, 2011.
[8] Kägi, A., Burger, D., and Goodman, J. R. Efficient synchronization: let them eat QOLB. In *ISCA*, 1997.
[9] Mueller, R., Teubner, J., and Alonso, G. Streams on wires: a query compiler for FPGAs. *PVLDB*, 2(1), 2009.
[10] Pandis, I., Johnson, R., Hardavellas, N., and Ailamaki, A. Data-oriented transaction execution. *PVLDB*, 3(1), 2010.
[11] Pandis, I., Tözün, P., Johnson, R., and Ailamaki, A. PLP: page latch-free shared-everything OLTP. *PVLDB*, 4(10), 2011.
[12] Sewall, J., et al. PALM: Parallel architecture-friendly latch-free modifications to B+trees on many-core processors. *PVLDB*, 4(11), 2011.
[13] Teubner, J., Woods, L., and Nie, C. Skeleton automata for fpgas: reconfiguring without reconstructing. In *SIGMOD*, 2012.
[14] Tsirogiannis, D., Harizopoulos, S., and Shah, M. A. Analyzing the energy efficiency of a database server. In *SIGMOD*, 2010.
[15] Venkatesh, G., et al. Conservation cores: reducing the energy of mature computations. In *ASPLOS*, 2010.
[16] Welsh, M., Culler, D., and Brewer, E. SEDA: an architecture for well-conditioned, scalable internet services. In *SOSP*, 2001.