The Q100 Database Processing Unit

This article demonstrates a proof-of-concept design, called the Q100, which provides one to two orders of magnitude improvement in efficiency over software database management systems. The Q100 exploits the innate structure of the workload, efficiently moving and manipulating the data in tables and columns rather than as an unstructured array of bytes. This approach complements other specialized computation engines.

Lisa Wu
Intel Labs

Andrea Lottarini
Timothy K. Paine
Martha A. Kim
Kenneth A. Ross
Columbia University

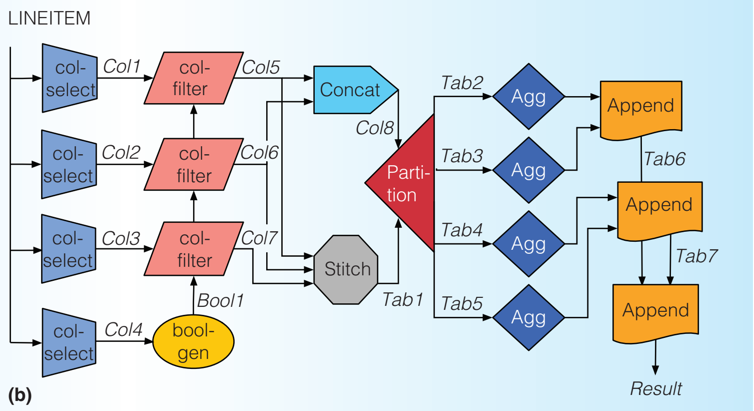
Today, big data analytics are not just important, they are essential. Analyses must process large volumes of various data at or near real-time velocity. With the big data technology and services market forecast to grow at a 26.24 percent annual growth rate through 2018 to reach $41.52 billion,[1] and with 2.6 exabytes of data created each day,[2] the research community must develop machines that can keep up with this data deluge.

For its part, the database management system (DBMS) software community has been exploring optimizations such as using column stores,[3–5] pipelining operations,[6] and vectorizing operations[7] to take advantage of commodity server hardware. This work applies those same techniques, but in hardware, to construct a domain-specific processor for databases. Just as conventional DBMSs operate on data in logical entities of tables and columns, our processor manipulates the same data primitives. Just as DBMSs use software pipelining between relational operators to reduce intermediate results, we too can exploit pipelining between relational operators implemented in hardware to increase throughput and reduce query-completion time. In light of the SIMD instruction set advances in general-purpose CPUs in the last decade, DBMSs also vectorize their implementations of many operators to exploit data parallelism. Our hardware does not use vectorized instructions but exploits data parallelism by processing multiple streams of data, corresponding to tables and columns, at once.

Streams of data. Pipelines. Parallel functional units. All of these techniques have long been known to be excellent fits for hardware, creating an opportunity to address some practical real-world concerns regarding big data. Our vision is of a class of domain-specific processors called database processing units (DPUs), which are analogous to GPUs. Whereas GPUs target graphics applications, DPUs target analytic database workloads. As GPUs operate on vertices, DPUs operate on tables and columns.

We designed and evaluated a first DPU called the Q100, a performance- and energy-efficient data analysis accelerator. Q100 contains a heterogeneous collection of fixed-function application-specific integrated circuit (ASIC) tiles, each of which implements a

```
SELECT S_SEASON,
       SUM(S_QUANTITY ) as SUM_QTY
FROM   SALES
WHERE  S_SHIPDATE <= '1998-12-01'
       - INTERVAL '90' DAY
GROUP BY S_SEASON
ORDER BY S_SEASON
```
(a)



(b)

Figure 1. An example query for producing a summary sales quantity report. (a) A SQL query and (b) the corresponding Q100 query plan, where nodes represent relational operators, and the edges between them are data dependencies.

well-known relational operator, such as a join or sort. The Q100 tiles operate on streams of data corresponding to tables and columns, over which the microarchitecture aggressively exploits pipeline and data parallelism.

The Q100 instruction set architecture

The Q100 instructions implement standard relational operators that manipulate database primitives such as columns, tables, and constants. The producer and consumer relationships between operators are captured with dependencies specified by the instruction set architecture (ISA). Queries are represented as graphs of these instructions, with the edges representing data dependencies between instructions. For execution, a query is mapped onto a spatial array of specialized processing tiles, each of which carries out one of the primitive functions. When producer-consumer node pairs are mapped to the same temporal stage of the query, they operate as a pipeline with data streaming from producer to consumer.

The basic instruction is a spatial one, implementing standard SQL-esque operators (namely, select, join, aggregate, boolgen, colfilter, partition, and sort). Other helper spatial instructions (stitch, append, and concatenate) perform various auxiliary functions, such as tuple reconstruction and query execution optimization. Figure 1a shows a simple query written in SQL to produce a summary sales quantity report per season for all items shipped as of a given date. Figure 1b shows the entire query as one graph with each shape representing a different spatial instruction and edges representing data dependencies.

In situations where a query does not fit on the array of available Q100 of tiles, it must be split into multiple temporal stages. These temporal stages are called temporal instructions and are executed in order. Each temporal instruction contains a set of spatial instructions, pulling input data from the memory subsystem and pushing completed partial query results back to the memory subsystem. Figure 2a shows an example array of specialized hardware tiles, or a resource profile, for a particular Q100 configuration. Figure 2b depicts how the query must to be broken into three temporal instructions, because the resource profile does not have enough column selectors, column filters, aggregators, or appenders at each stage.

This ISA is energy efficient because it closely matches building blocks of our target domain, while simultaneously encapsulating operations that can be implemented efficiently in hardware. Spatial instructions are executed in a dataflow style seen in dataflow machines of the 80s,[8,9] the 90s,[10] and more recently,[11–13] eliminating complex issue and control logic, exposing parallelism, and passing data dependencies directly from producer to consumer. All of these features provide performance benefits and energy savings.

The Q100 microarchitecture

The Q100 design uses hardware tiles that implement the Q100 ISA. Using 19 TPC

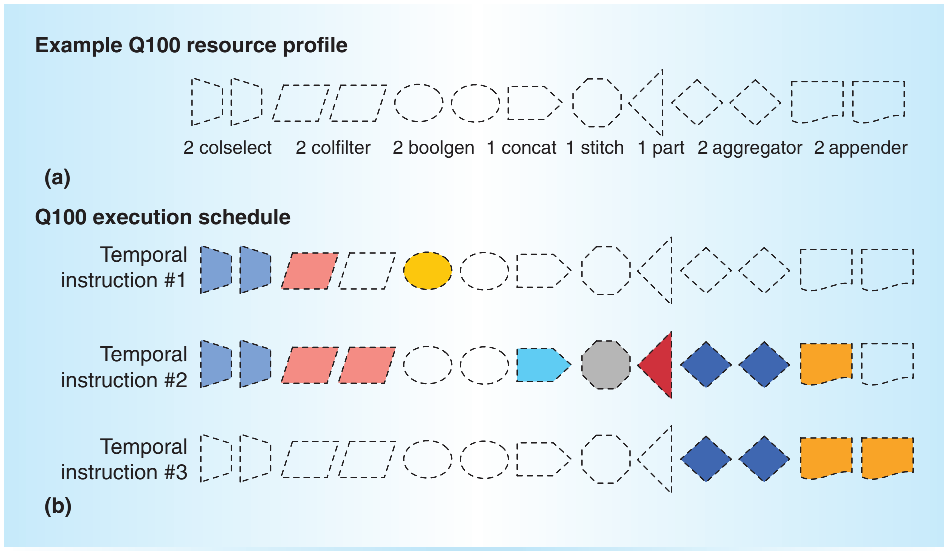


Figure 2. Given a Q100 instance with finite resources, as shown in (a), the query plan is broken into a sequence of temporal instructions by a scheduling algorithm that attempts to maximize resource usage and minimize data spillage between instructions.[14] In (b), the query plan from Figure 1 requires three steps to complete.

Benchmark H (TPC-H) queries as benchmarks, we performed a detailed Q100 design space exploration with which we explored the tradeoffs and selected three interesting Q100 designs: minimal power, peak performance, and a balanced design that offers maximal performance per watt. We then explored the impact of communication—both intra-tile and memory—on these three designs.

The Q100 tile implementation and characterization

The Q100 contains 11 types of hardware tile corresponding to the 11 operators in the ISA. As in the ISA, we break the discussion into core functional tiles and auxiliary helper tiles. Table 1 gives the facts and figures of the Q100 tile implementation and characerization. The slowest tile determines the Q100's clock cycle. As Table 1 shows, the partitioner limits the Q100 frequency to 315 MHz.

Methodology. Each tile was implemented in Verilog and synthesized, placed, and routed using Synopsys 32-nm generic libraries with the Synopsys[15] design and Integrated Circuit (IC) compilers to produce timing, area, and power numbers. We report each design's post-place-and-route critical path as logic delay plus clock network delay, adhering to the industry standard of reporting critical paths with a margin.

Q100 functional tiles. The sorter sorts its input table using a designated key column and a bitonic sort.[16] In general, hardware sorters operate in batches and require all items in the batch to be buffered prior to the start of the sort. Because buffers and sorting networks are costly, this limits the number of items that can be sorted at once. For the Q100 tile, this is 1,024 records, so to sort larger tables, they must first be partitioned with the partitioner.

The partitioner splits a large table into multiple smaller tables called partitions. Each row in the input table is assigned to exactly one partition based on the value of the key field. The Q100 implements a range partitioner, which splits the space of keys into contiguous ranges. We chose this because it tolerates irregular data distributions[17] and produces ordered partitions, making it a suitable precursor to the sorter.

The joiner performs an inner-equijoin of two tables, one with a primary key and the other with a foreign key. To keep the design simple, the Q100 currently supports only inner-equijoins. They are by far the most common type of join, although extending the joiner to support other types (such as

Table 1. Physical design characteristics of the Q100 tiles post place and route, and compared to a Xeon core.

		Area		Power		Critical path	Design width (bits)*		
Q100 tiles	Tile	mm^2	Xeon (%)†	mW	Xeon (%)	ns	Record	Column	Comparator
Functional	Aggregator	0.029	0.07	7.1	0.14	1.95	—	256	256
	Arithmetic logic unit (ALU)	0.091	0.21	12.0	0.24	0.29	—	64	64
	BoolGen	0.003	0.01	0.2	<0.01	0.41	—	256	256
	ColFilter	0.001	<0.01	0.1	<0.01	0.23	—	256	—
	Joiner	0.016	0.04	2.6	0.05	0.51	1,024	256	64
	Partitioner	0.942	2.20	28.8	0.58	3.17†	1,024	256	64
	Sorter	0.188	0.44	39.4	0.79	2.48	1,024	256	64
Auxiliary	Append	0.011	0.03	5.4	0.11	0.37	1,024	256	—
	ColSelect	0.049	0.11	8.0	0.16	0.35	1,024	256	—
	Concat	0.003	0.01	1.2	0.02	0.28	—	256	—
	Stitch	0.011	0.03	5.4	0.11	0.37	—	256	—

*Intel E5620 Xeon server with two chips. Each chip contains four cores and eight threads running at 2.4 GHz with a 12-Mbyte last-level cache and three channels of DDR3, providing 24 Gbytes of RAM. Comparisons are done using estimated single-core area and power consumption derived from published specifications.
†The slowest tile, the partitioner, determines the frequency of Q100 at 315 MHz.

outer-joins) would not increase its area or power substantially.

The arithmetic logic unit (ALU) tile performs arithmetic and logical operations on two input columns, producing one output column. It supports all arithmetic and logical operations found in SQL (that is, ADD, SUB, MUL, DIV, AND, OR, and NOT) as well as constant multiplication and division. We use these latter operations to work around the current lack of a floating-point unit in the Q100. We use fixed-point arithmetic to support single-precision floating-point arithmetic, as most domain-specific accelerators have done. SQL does not specify precision requirements for floating-point calculations, and most commercial DBMSs support either single-precision floating-point or double-precision floating-point calculations.

The Boolean generator compares an input column with either a constant or a second input column, producing a column of Boolean values. Using just two hardware comparators, the tile provides all six comparisons used in SQL (that is, EQ, NEQ, LTE, LT, GT, and GTE). Although this tile could have been combined with the ALU, offering two tiles à la carte leaves more flexibility when allocating tile resources. The Boolean generator is often paired with the column filter with no need for an ALU. It also is often used in a chain or tree to form complex predicates—again, not always in a 1-to-1 correspondence with ALUs.

The column filter takes in a column of Booleans (from a Boolean generator) and a second data column. It outputs the same data column but drops all rows where the corresponding bool is false.

Finally, the aggregator takes in the column to be aggregated and a “group by” column whose values determine which entries in the first column to aggregate. For example, if the query sums purchases by ZIP code, the data columns are the purchase totals, and the group-by is the ZIP code. The tile requires that both input columns arrive sorted on the group-by column so that the tile can simply compare consecutive group-by values to determine where to close each aggregation. This decision has tradeoffs. A hash-based implementation might not require presorting, but it would require a buffer of unknown size to maintain the partial aggregation results for each group. The Q100 aggregator supports all aggregation operations in the

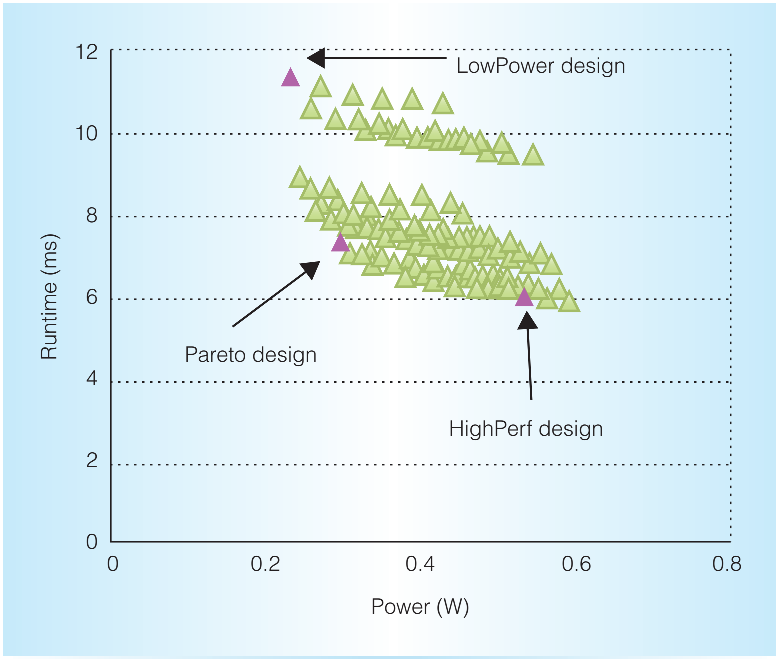


Figure 3. Out of 150 configurations, we selected three designs for further evaluation: LowPower for an energy-conscious configuration, HighPerf for a performance-conscious configuration, and Pareto for a design that maximizes performance per watt.

SQL spec, namely, MAX, MIN, COUNT, SUM, and AVG.

Q100 auxiliary tiles. The column selector extracts a column from a table, and the column stitcher does the inverse, taking multiple input columns (up to a maximum total width) and producing a table. This operation often precedes partitions and sorts where queries frequently require column A sorted according to the values in column B. The column concatenator concatenates corresponding entries in two input columns to produce one output column. This can cut down on sorts and partitions when a query requires sorting or grouping on more than one attribute (that is, column). Finally, the table appender appends two tables with the same schema. This is often used to combine the results of per-partition computations.

Q100 tile mix design space exploration

To understand the relative utility of each type of tile, and the tradeoffs among them, we explored a wide design space of different sets of Q100 tiles. We began with a sensitivity analysis of TPC-H performance, evaluating each type of tile in isolation to bound the maximum number of useful tiles of each type. We then carried out a complete design space exploration, considering multiple tiles at once, from which we understood the power performance shape of the Q100 space and selected three configurations (that is, tile mixtures) for further analysis.

Methodology. We developed a functional and timing Q100 simulator in C++. We validated each tile's function and throughput against simulations of the corresponding Verilog. Because we did not yet have a compiler for the Q100, we manually implemented each TPC-H query in the Q100 ISA. Using the simulator, we confirmed that the Q100 query implementations produce the same results as the SQL versions running on MonetDB (www.monetdb.org). Given a query and a Q100 configuration, a scheduling algorithm described and evaluated in our previous work[14] schedules each query into a sequence of temporal instructions. The simulator produces cycle counts, which we convert to wall clock time using a Q100 frequency of 315 MHz.

Tile count sensitivity. To understand how sensitive Q100 is to the number of each type of tile—say, aggregators—we simulate a range of Q100 configurations, sweeping the number aggregators, while holding all other types of tiles at sufficiently high counts so as not to limit performance. Having run this experiment for each of the 11 types of tile, we identified the maximum useful count for each type of tile.

Design space parameters. We explored the full tile design space up to the maximum useful count for each type of tile, except the small negligible tiles that consumed less than 10 mW. For those tiles, we always allocated the maximal useful instances.

Power-performance design space. Figure 3 plots the power-performance tradeoffs for 150 Q100 designs. Among these configurations, we selected three designs for further evaluation. *LowPower* is an energy-conscious design point that has just one partitioner, one sorter, and one ALU, and which consumed

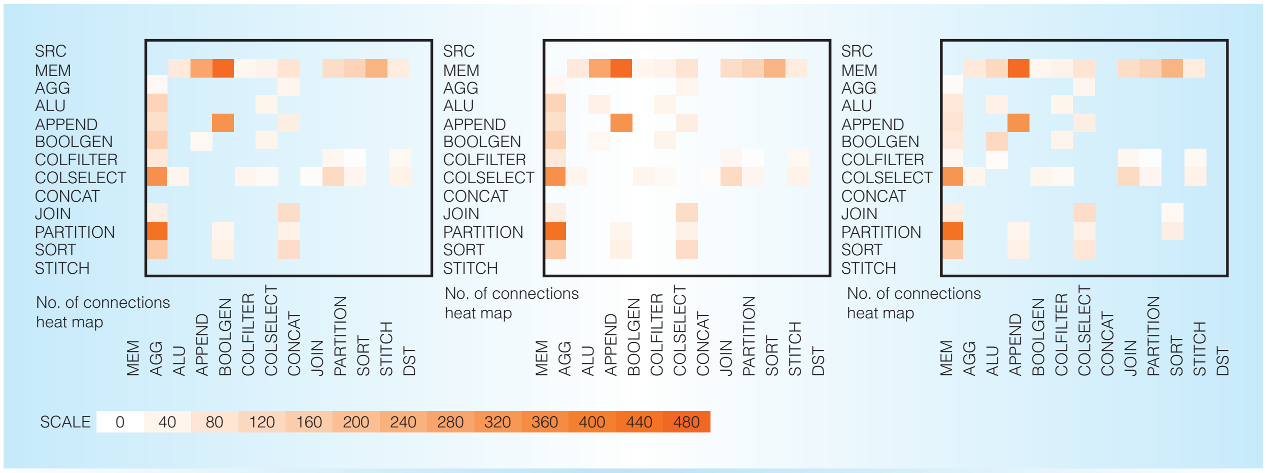


Figure 4. A heat map of tile-to-tile connection counts: (a) LowPower, (b) Pareto, and (c) HighPerf designs. Most intra-tile connections exist mostly when communicating to and from memory.

the lowest power among all the configurations. *Pareto* is a balanced design on the Pareto-optimal frontier, which, with two partitioners, one sorter, and four ALUs, provides the most performance per watt among the designs. *HighPerf* is a performance-optimized design, with three partitioners, six sorters, and five ALUs, which maximizes performance at the cost of a relatively higher power consumption.

The Q100 communication needs

Having explored the Q100's computational needs, we now consider its communication needs, both on-chip intra-tile communication and off-chip with memory. Because the target workload is large-scale data, each of these channels will need to support substantial throughput.

Communication topology. In the experiments and simulations thus far, we have assumed all-to-all communication for all of the Q100 tiles and memory. However, analytic queries are not random, and we expect them to have certain tendencies. Figure 4 indicates how many times a particular source (*y*-axis) feeds into a particular destination (*x*-axis) across all of TPC-H. First, we observe that most tiles communicate to and from memory so often that we must properly understand and provision for the Q100 to/from memory bandwidth. Second, tiles tend to communicate with a subset of each other, validating our hypothesis that the communication was not truly all-to-all. Third, we note that these communication patterns do not vary across the three Q100 designs.

On-chip bandwidth constraints. We envisioned a network on chip (NoC) like the 2D, 80-node mesh on Intel's TeraFlops chip.[18] For a conservative estimate, we scaled down TeraFlops's node-to-node 80 Gbytes per second (GBps) at 4 GHz to the frequency of the Q100, resulting in a conservative Q100 NoC bandwidth of 6.3 GBps.

To understand and quantify the performance impact of the Q100 NoC bandwidth, we performed a sensitivity study, sweeping the bandwidth from 5 to 20 GBps (see Figure 5). The runtime of all queries in all three configurations was normalized to that of the HighPerf design with unlimited NoC bandwidth (ideal). We observed that only a handful of queries were sensitive to an imposed NoC bandwidth limit; however, the slowdown for those queries could be as much as 50×, making interconnect throughput a performance bottleneck when limited to 6.3 GBps.

Off-chip bandwidth constraints. Memory, we have also seen, is a very frequent communicator, acting as a source or destination for all types of Q100 tiles. Half of those connections also required high-throughput connections. In Figures 6 and 7, we examine the high, low, and average read and write memory bandwidth for each query, sorted by

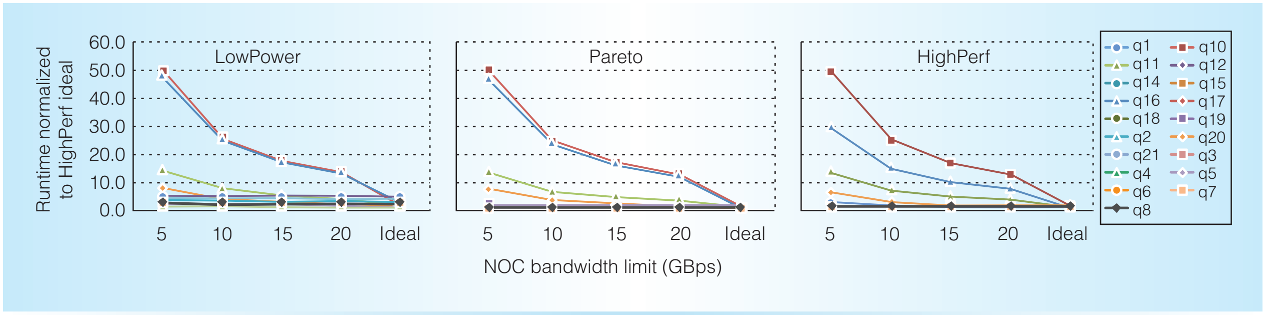


Figure 5. Most TPC-H queries are not sensitive to the Q100 intraconnection throughput, except for Q10, Q16, and Q11. These queries process large volumes of records throughout the query with little local selection conditions to funnel down the intermediate results. When network-on-chip (NoC) bandwidth is constrained, these queries could execute 50 times slower.

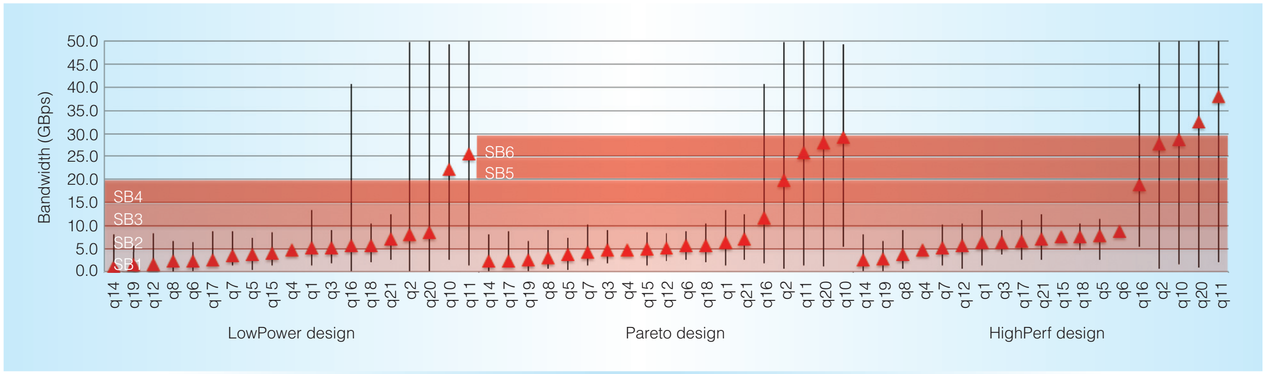


Figure 6. A plot of all TPC-H query read memory bandwidth demands (high, low, and average), sorted by average. Read bandwidth varies quite a bit from query to query, with Q10 and Q11 being the most bandwidth starved. For Q100, the LowPower design is provisioned with four stream buffers, and Pareto and HighPerf designs are provisioned with six stream buffers, as shown in shaded gradations.

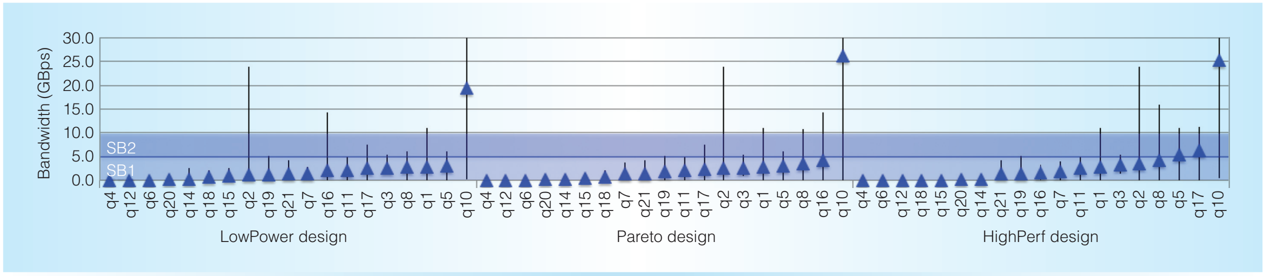


Figure 7. Write bandwidth demands are quite a bit lower than read bandwidth demands for most queries. We sized all three designs with two stream buffers, providing 10 Gbytes per second (GBps) write bandwidth to memory.

average bandwidth. We first notice that queries vary substantially in their memory read bandwidths but relatively little in their write bandwidths. This is largely because analytic queries take in large volumes of data and produce comparatively small results, matching the volcano style of software relational database pipelined execution.[19] Second, queries generally consume more bandwidth as the design becomes higher performance, because faster designs tend to process more data in a smaller period of time. Finally, in the same fashion that we expect the NoC will limit performance, realistic available bandwidth to and from memory is also likely to slow query processing.

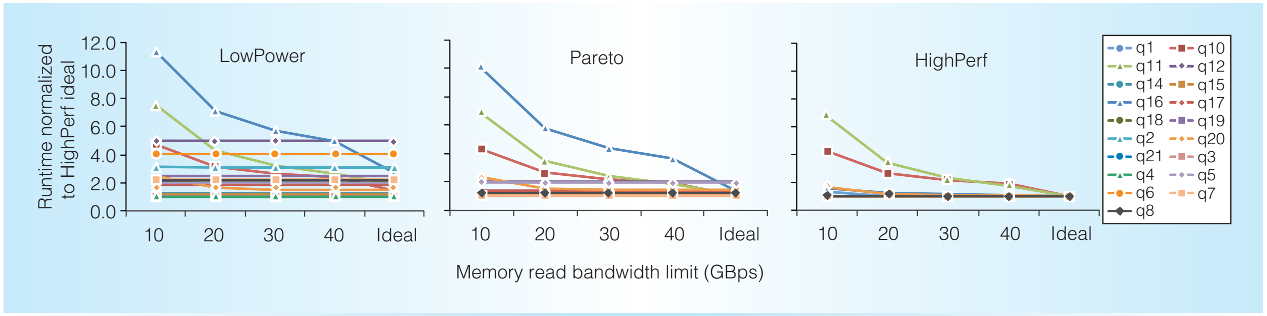


Figure 8. Similar to NoC bandwidth, most queries are not sensitive to memory read bandwidth. However, in the HighPerf design, more resources allow for a more efficient scheduling of temporal instructions, reducing high-volume communications to and from memory.

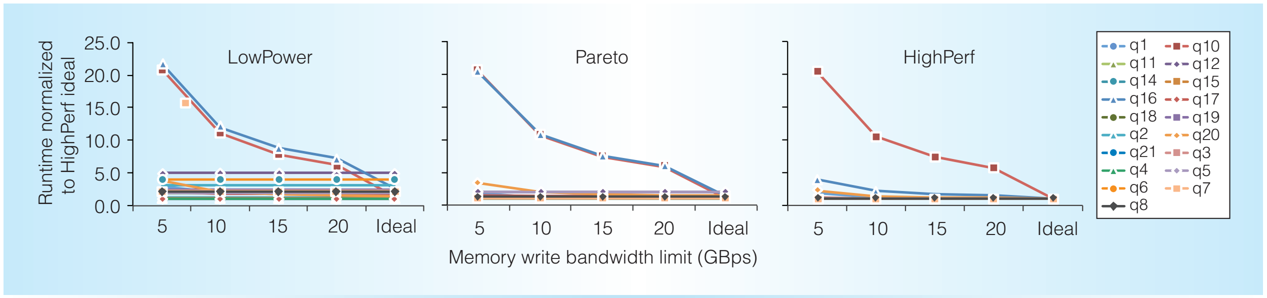


Figure 9. With 10 GBps of memory write bandwidth, only one or two queries are performance-limited by memory write bandwidth.

To quantify the performance impact of memory bandwidth, we swept memory read bandwidth from 10 to 40 GBps and memory write bandwidth from 5 to 20 GBps (see Figures 8 and 9). As with the NoC study, only two or three queries are sensitive to memory read and write bandwidth limits, but with much more modest slowdowns.

Performance impact of communication resources. Applying the NoC and memory bandwidth limits discussed earlier, we simulated an NoC bandwidth cap of 6.3 GBps, a memory read limit of 20 GBps for LowPower and 30 GBps for Pareto and HighPerf, and a memory write limit of 10 GBps. Figure 10 shows the impact as each of these limits is applied to an unlimited-bandwidth simulation. On account of on-chip communication, queries slow down 33 to 61 percent, with only a slight additional loss on account of memory to 34 to 62 percent slowdown overall. These effects are largely due to Q10 and Q11, the two most memory-hungry queries, which suffer 1.4 to 1.5 times slowdown and 6 to 11 times slowdown, respectively, compared to software.

Our simulator models a uniform memory access latency of 160 ns, based on a 300-cycle memory access time from a 2-GHz CPU. When the imposed interconnect and memory throughput slow the execution of a spatial and a temporal instruction, respectively, the simulator reflects that—although we found that throughput was primarily interconnect-limited, and thus the visible slowdown beyond that due to memory was negligible. The Q100 reduces total memory accesses relative to software implementations by eliminating many reads and writes of intermediate results. For the remaining memory accesses, the Q100 can mask most stalls thanks to heavily parallelized computation that exploits both data and pipeline parallelism.

Area and power impact of communication resources. Starting with the area and power for the tiles in each Q100 design, we added the additional area and power due to the NoC and stream buffers. Table 2 lists the area of the three design points broken down by tile, NoC, and stream buffers. We added an extra 30 percent area and power to the

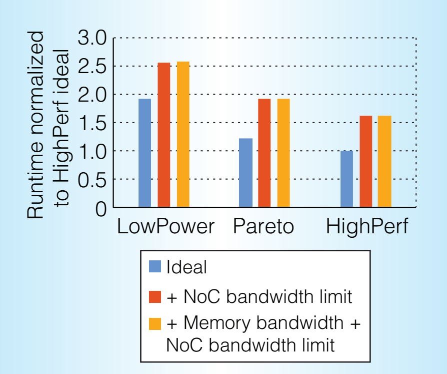


Figure 10. From the bandwidth heat maps plotted earlier, we see that Q100 was demanding a lot more NoC bandwidth than provisioned. Here, we plotted runtime with respect to no bandwidth-limit penalties, and we see a large slowdown at 30 to 60 percent, a caution for future implementations to design sufficient bandwidth for intra-tile connections.

Q100 designs for the NoC, based on the characteristics of the TeraFlops implementation.[18] For the stream buffers, we added 0.13 mm^2 and 0.1 W for each stream buffer.[17] In sum, the Q100 remains quite small, with the large, HighPerf configuration, including NoC and stream buffers, taking 17.3 percent of the area and 26.1 percent of the power of a single Xeon core.

The Q100 evaluation

Taking what we've learned about the Q100 system, its ISA, its implementation, and its internal and external communication, we now compare our three configurations, LowPower, Pareto, and HighPerf, with a conventional software DBMS. This evaluation has three parts: initial power and performance benchmarking for the TPC-H queries as executed on a conventional DBMS+CPU system, comparison of Q100's execution of TPC-H to that system's, and, finally, an evaluation of how a Q100 designed for one scale of database handles the same queries over a database 100 times larger.

Methodology

We measured the performance and energy consumption of MonetDB 11.11.5 running on a Sandy Bridge Xeon server and executing the set of TPC-H queries. Each reported result is the average of five runs during which we measured the elapsed time and energy consumption. For the latter, we used Intel's Running Average Power Limit (RAPL) energy meters,[20,21] which expose energy usage estimates to software via model-specific registers deducting the idle background power.

Although MonetDB supports multiple threads, our measurements of power and speedups indicate that individual TPC-H queries do not parallelize well, even for large databases (that is, 40 Gbytes). Here, we compare the Q100's performance and energy to the measured single-threaded values, as well as to an optimistic estimate of a 24-way parallelized software query, one that runs 24 times faster than the single threaded at the same average power as a single software thread.

The Q100 performance comparison

Figure 11 plots the query execution time on the Q100 designs relative to the execution time on single-threaded MonetDB. We see that Q100 performance exceeds a single software thread by 37 to 70 times, and exceeds a perfectly scaled 24-thread software by 1.5 to 2.9 times. This is primarily because of Q100's reduced instruction control costs, which are a byproduct of the large instruction granularity, where each Q100 instruction does the work of billions (or more, depending on the data size) of software instructions. In addition, the Q100 processes many instructions at once, in pipelines and in parallel, generating further speedups. Finally, the Q100, having brought some data onto the chip, exploits on-chip communication tile parallelism to perform multiple operations on the data before returning the results to memory, thereby maximizing the work per memory access and hiding the memory latency with computation.

The Q100 energy comparison

Figure 12 plots the query energy consumption relative to the energy consumption on single-threaded MonetDB. Fixed-function ASICs, which comprise the Q100, are inherently more energy efficient than general-purpose processors. Both industry and

Table 2. Area and power of the three Q100 configurations, broken down by tile, on-chip interconnect, and stream buffers.

	Area				Power			
Q100 configuration	Tiles mm^2	NoC mm^2	Stream buffers mm^2	Total Xeon (%)	Tiles W	NoC W	Stream buffers W	Total Xeon (%)
LowPower	1.890	0.567	0.520	7.0	0.238	0.071	0.400	14.2
Pareto	3.107	0.932	0.780	11.3	0.303	0.091	0.600	19.9
HighPerf	5.080	1.524	0.780	17.3	0.541	0.162	0.600	26.1

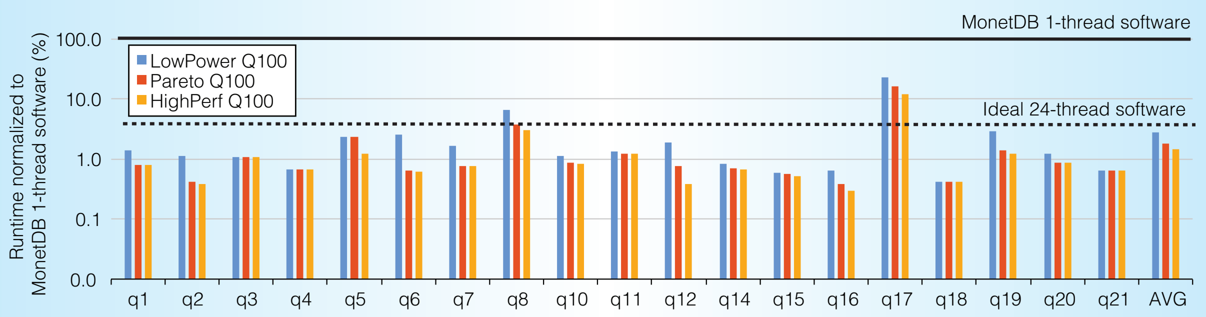


Figure 11. TPC-H query runtime normalized to MonetDB single-threaded software shows a performance improvement of 37 to 70 times on average across all queries.

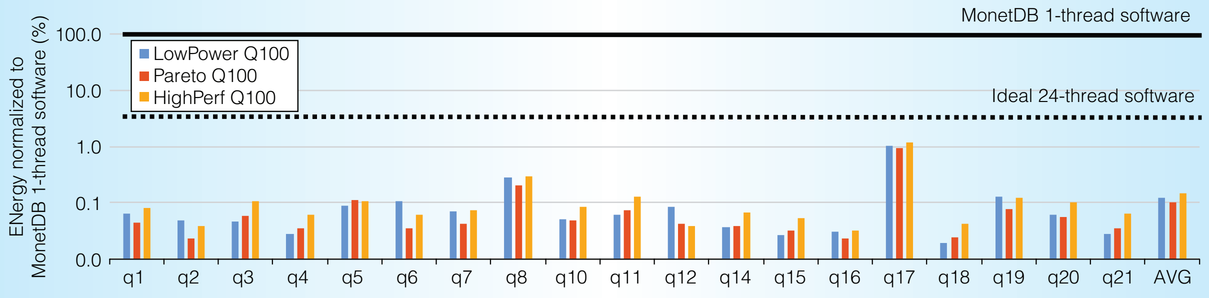


Figure 12. TPC-H query energy consumption normalized to MonetDB single-thread running on cores consuming non-idle power shows 691 to 983 times energy efficiency on average across all queries.

academia, for example, state that GPUs are 10 to 1,000 times more efficient than multicore CPUs for well-suited graphics kernels. Similarly, the Q100 is 1,400 to 2,300 times more energy efficient than MonetDB when executing the analytic queries for which it was designed. The energy efficiency of our Pareto design is 1.1 times better than our LowPower design and 1.6 times better than our HighPerf design.

Scaling up data

Finally, as big data continues to grow, we wanted to evaluate how the Q100 handles databases that are orders of magnitude larger than the ones for which it was initially

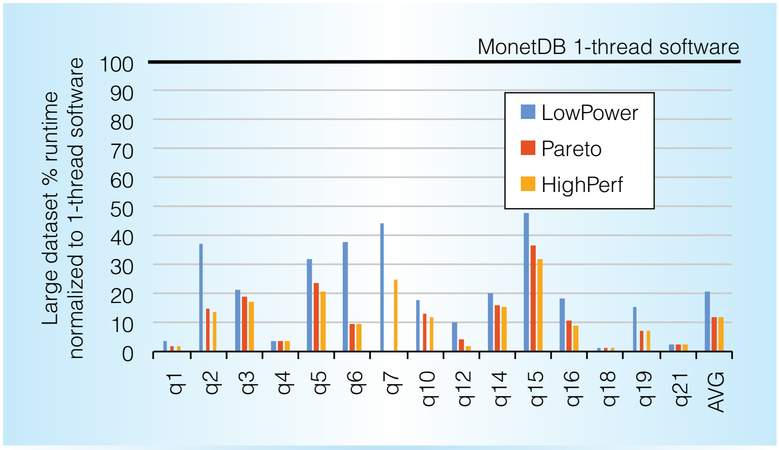


Figure 13. With a dataset that is 100 times the size of our previous input tables, TPC-H still shows a 10 times performance improvement relative to software on average.

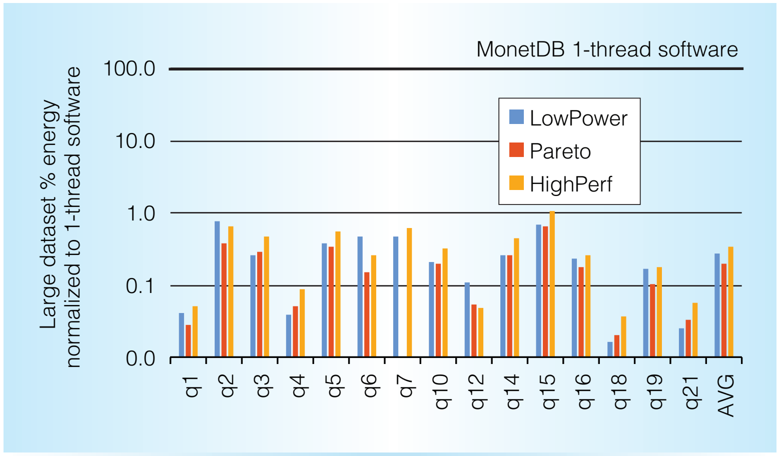


Figure 14. With a 100 times larger dataset, the Q100 still consumes 1/100th of the energy that software consumes.

developed. Thus, we performed the same Q100-MonetDB comparison using 100 times larger data, a TPC-H scaling factor of SF1. Figures 13 and 14 show the results. With the input data having grown by 100 times, the Q100 speedup over software dropped from 100 to 10 times. This is because the initial design configuration targeted a much smaller dataset size, and therefore the optimal design point reflected the best performance efficiency for the smaller dataset. Another reason is that the Q100 sorter can only sort up to 1,024 records at once, and bigger input datasets require many more partitioners and sorters for processing than the smaller datasets. However, the total energy remains 100 times lower regardless of data size.

As data quantities continue to explode, technology must keep pace. To mitigate commensurate increases in time and energy required to process this data with conventional DBMSs running on general-purpose CPUs, this article has presented the Q100, a DPU for analytic query workloads.

This research proposes a novel class of domain-specific accelerators for big data. The Q100 demonstrates the feasibility of a new class of domain-specific accelerators. Unlike prior work that compiles queries to hardware (such as LINQits[22] or Teradata [www.teradata.com]), DPUs use domain-specific circuits to accelerate analytic queries in general. The Q100 is one instance of such a DPU that demonstrates the potential of this acceleration style and opens new approaches for both the architecture and database communities.

The Q100 architecture embodies a unique approach toward efficiency with coarse-grained operations and operands. Specializing in granularities that are too large makes the solution not broadly applicable (for example, see recent H.264 work from Stanford[23]), whereas specializing in granularities that are too small provides flexibility but sacrifices efficiency (for example, AVX or other vector instructions). This work uniquely deploys a collection of heterogeneous building blocks whose operations and operands match the computing and data primitives used in relational analytic applications. These spatial instructions eliminate complex issue and control logic, resulting in a solution that both broadly applies to big data analytics and provides orders of magnitude performance and energy efficiency. This architecture also demonstrates the potential of data-oriented specialization. Moving data through the memory subsystem and CPU cache hierarchy consumes more than double the energy of the computation itself.[24] With an ASIC designed specifically to manipulate relational tables, the Q100 delivers an order of magnitude improvement in energy efficiency.

The Q100 streaming architecture provides scalability, an important attribute for big data acceleration. Using spatial instructions, the architecture eliminates data dependency stalls by exposing parallelism and streaming data directly from producer to consumer. We demonstrated how this streaming architecture's energy efficiency is insensitive to input data size and scalable. This result is particularly significant in light of the large volumes of data used in modern analyses.

Looking forward, the system architecture provides modularity for further expansion. The modularity of the Q100 architecture makes it easy to introduce other streaming accelerators, such as a regular expression matching or a compression/decompression engine, to expand both the use and benefits of having a big data accelerator.

MICRO

References

1. IDC Research, "Worldwide Big Data Technology and Services 2014–2018 Forecast," Sept. 2014; www.idc.com/getdoc.jsp?containerId=250458.
2. A. McAfee and E. Brynjolfsson, "Big Data: The Management Revolution," *Harvard Business Rev.*, 2012; https://hbr.org/2012/10/big-data-the-management-revolution/ar.
3. S. Idreos et al., "MonetDB: Two Decades of Research in Column-Oriented Database Architectures," *IEEE Data Eng. Bull.*, vol. 35, 2012, pp. 40–45.
4. A. Lamb et al., "The Vertica Analytic Database: C-Store 7 Years Later," *Proc. VLDB Endowment*, vol. 5, no. 12, 2012, pp. 1790–1801.
5. D.J. Abadi, P.A. Boncz, and S. Harizopoulos, "Column-Oriented Database Systems," *Proc. VLDB Endowment*, vol. 2, no. 2, 2009, pp. 1664–1665.
6. D.J. Abadi et al., "Materialization Strategies in a Column-Oriented DBMS," *Proc. IEEE 23rd Int'l Conf. Data Eng.*, 2007, pp. 466–475.
7. M. Zukowski and P. Boncz, "Vectorwise: Beyond Column Stores," *IEEE Data Eng. Bull.*, vol. 35, 2012, pp. 21–27.
8. L. Wu et al., "Q100: The Architecture and Design of a Database Processing Unit," *Proc. 19th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2014, pp. 255–268.
9. J. Gurd, C.C. Kirkham, and I. Watson, "The Manchester Prototype Dataflow Computer," *Comm. ACM*, vol. 28, no. 1, 1985, pp. 34–52.
10. J.B. Dennis, *Advanced Topics in Data-flow Computing*, Prentice-Hall, 1991.
11. J. Hicks et al., "Performance Studies of the Monsoon Dataflow Processor," *J. Parallel and Distributed Computing*, vol. 18, no. 3, 1993, pp. 273–300.
12. S. Swanson et al., "The Wavescalar Architecture," *ACM Trans. Computer Systems*, vol. 25, no. 2, 2007, article 4.
13. M. Gebhart et al., "An Evaluation of the TRIPS Computer System," *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2009, doi:10.1145/1508244.1508246.
14. A. Parashar et al., "Triggered Instructions: A Control Paradigm for Spatially-Programmed Architectures," *Proc. 40th Ann. Int'l Symp. Computer Architecture*, 2013, pp. 142–153.
15. "32/28nm Generic Library for IC Design, Design Compiler, IC Compiler," *Synopsys*, 2015; www.synopsys.com/Community/UniversityProgram/Pages/32-28nm-generic-library.aspx.
16. M.F. Ionescu and K.E. Schauser, "Optimizing Parallel Bitonic Sort," *Proc. 11th Int'l Parallel Processing Symp.*, 1997, pp. 303–309.
17. L. Wu et al., "Navigating Big Data with High-Throughput, Energy-Efficient Data Partitioning," *Proc. 40th Ann. Int'l Symp. Computer Architecture*, 2013, pp. 249–260.
18. S. Vangal et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2007, pp. 98–589.
19. G. Graefe and W.J. McKenna, "The Volcano Optimizer Generator: Extensibility and Efficient Search," *Proc. 9th Int'l Conf. Data Eng.*, 1993, pp. 209–218.
20. "Intel 64 R and IA-32 Architectures Software Developer's Manual," Intel; http://download.intel.com/products/processor/manual/253669.pdf.

21. D. Howard et al., "RAPL: Memory Power Estimation and Capping," *Proc. ACM/IEEE Int'l Symp. Low-Power Electronics and Design*, 2010, pp. 189–194.
22. E.S. Chung, J.D. Davis, and J. Lee, "LINQits: Big Data on Little Clients," *Proc. 40th Int'l Symp. Computer Architecture*, 2013, pp. 261–272.
23. R. Hameed et al., "Understanding Sources of Inefficiency in General-Purpose Chips," *Proc. 37th Int'l Symp. Computer Architecture*, ISCA, 2010, pp. 37–47.
24. W.J. Dally et al., "Efficient Embedded Computing," *Computer*, vol. 41, no. 7, 2008, pp. 27–32.

Lisa Wu is a research scientist at Intel Labs. Her research interests include computer architecture, accelerators, energy-efficient computing on high-performance computing, and emerging applications related to big data, machine learning, and computational biology. Wu has a PhD in computer science from Columbia University. She is a member of the ACM SIGARCH. Contact her at lisa.wu@intel.com.

Andrea Lottarini is a PhD student in the Department of Computer Science at Columbia University. His research interests include computer architecture and heterogeneous computing. Lottarini has an MS in computer science from Università di Pisa and Scuola Superiore Sant'Anna. Contact him at flottarini@cs.columbia.edu.

Timothy K. Paine is a master's student in the Department of Computer Science at Columbia University. His research interests include hardware accelerators for machine learning and data processing. Paine has a BS in computer science from Columbia University. Contact him at tkp2108@columbia.edu.

Martha A. Kim is an assistant professor in the Computer Science Department at Columbia University. Her research interests include computer architecture, parallel hardware and software systems, and energy-efficient computation on big data. Kim has a PhD in computer science from the University of Washington. She is a member of IEEE and the ACM. Contact her at martha@cs.columbia.edu.

Kenneth A. Ross is a professor in the Computer Science Department at Columbia University. His research focuses on database management systems, particularly their performance on modern multicore machines, GPUs, and other accelerator platforms. Ross has a PhD in computer science from Stanford University. He is a member of the ACM. Contact him at karg@cs.columbia.edu.