Master of None Acceleration: A Comparison of Accelerator Architectures for Analytical Query Processing

Andrea Lottarini*
Google

João P. Cerqueira
Columbia University

Thomas J. Repetti
Columbia University

Stephen A. Edwards
Columbia University

Kenneth A. Ross
Columbia University

Mingoo Seok
Columbia University

Martha A. Kim
Columbia University

ABSTRACT

Hardware accelerators are one promising solution to contend with the end of Dennard scaling and the slowdown of Moore's law. For mature workloads that are regular and have high compute per byte, hardening an application into one or more hardware modules is a standard approach. However, for some applications, we find that a programmable homogeneous architecture is preferable.

This paper compares a previously proposed heterogeneous hardware accelerator for analytical query processing to a homogeneous systolic array alternative. We find that the heterogeneous and homogeneous accelerators are equivalent for large designs, while for small designs the homogeneous is better. Our analysis explains this counter-intuitive result, finding that the homogeneous architecture has higher average resource utilization and lower relative costs for the communication infrastructure.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; **Systolic arrays**; • **Information systems** → *Database query processing*.

KEYWORDS

spatial architectures, database implementation, design space exploration, accelerators

ACM Reference Format:
Andrea Lottarini, João P. Cerqueira, Thomas J. Repetti, Stephen A. Edwards, Kenneth A. Ross, Mingoo Seok, and Martha A. Kim. 2019. Master of None Acceleration: A Comparison of Accelerator Architectures for Analytical Query Processing. In *The 46th Annual International Symposium on Computer Architecture (ISCA '19), June 22–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307650.3322220

1 INTRODUCTION

With the end of Dennard scaling it is possible to shrink a transistor's size but not its power consumption. Accelerators have been widely adopted to circumvent this constraint. Die area that would be otherwise underused is instead invested in tailored, resource-efficient implementations of critical applications [8, 16, 19, 20, 22, 44, 45]. Such accelerators have been successfully deployed for a variety of workloads including machine learning training and inference [9, 12, 16], web search [34], and network processing [7, 10].

Accelerators implement key computational blocks for an application directly in hardware, thus offering high energy efficiency. This approach avoids CPU overheads such as instruction fetch and decode and exploits parallelism in the application [15]. We examine analytical query processing [26, 37], a workload that has drawn significant acceleration study, both for key computational kernels such as partitioning [44] and hash joins [20], as well as entire queries [1, 8, 31, 45].

This paper compares two accelerator architectures for this domain: one that hardens multiple application kernels into heterogeneous fixed function functional units and one that implements the same set of functions via homogeneous programmable processing elements. For the heterogeneous design, we analyze an accelerator previously designed by our group: the Q100 [45, 46]. The Q100 targets full analytical queries, and is composed of a set of highly efficient functional units. Each functional unit, or tile, can execute a specific relational algebra operation on incoming streams of data. The tiles communicate via an on-chip network originally proposed to be a mesh [45] and subsequently refined to a semi-custom topology [23]. To execute a particular query, one reconfigures the modules and adjusts the routing function in the NoC. We compare and contrast this approach with a homogeneous systolic array design. The functional units in the homogeneous design are equipotent, and can be individually programmed to collectively implement the same operators as the Q100.

Our results show that the configurability in the homogeneous design incurs per-operator power and area losses relative to the Q100, with multiple homogeneous functional units often needed to implement the function of a single Q100 tile. However, these per-operator losses are recouped by selecting and sizing relational operators on a per-query basis. On the communication side, because most of the inter-functional unit communication in the homogeneous accelerator is between neighbors, a spartan interconnect is sufficient. Ultimately, we find that the two designs offer comparable

[*] Andrea Lottarini performed this work while a graduate student at Columbia University. He (lottarini@google.com) and Martha Kim (martha@cs.columbia.edu) are the corresponding authors.

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6669-4/19/06…$15.00
https://doi.org/10.1145/3307650.3322220

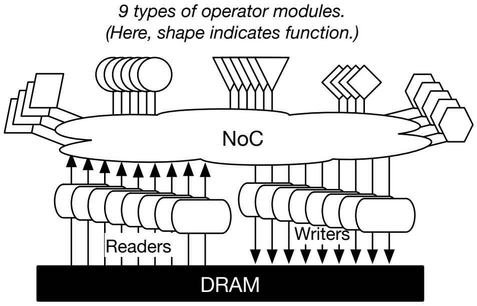

9 types of operator modules.
(Here, shape indicates function.)
NoC
Readers
Writers
DRAM

Figure 1: The Q100 [45] is a spatial architecture for the acceleration of relational analytic queries. It contains a mix of compute modules, specialized units that perform relational algebra operations. An instance of a Q100 accelerator can have replicated compute modules to exploit parallelism in the query. The modules stream data via an on-chip network.

performance per area and power, with only subtle differences. Our experiments show that, when area and power budgets are small, the homogeneous approach is preferable.

We use a previously developed SQL compiler for the Q100 and extend it to target the homogeneous design. The benefits of a full software infrastructure are twofold: It demonstrates that software support for these custom architectures is feasible, and it contributes to a controlled comparison between the two architectures, minimally influenced by software differences. The compiler differs for the two targets only as strictly necessary in the backend. The frontend – including, importantly, the query planner – is identical. This limits our experimental analysis to the operations supported by the Q100. Although either accelerator could be extended to cover other primitives – via new Q100 tiles or instruction set extensions for the homogeneous units – the experiments in this paper control for functional coverage. The homogeneous approach increases software complexity due to increased degrees of freedom when scheduling operations. We encountered this complexity when implementing the homogeneous backend, and implemented a greedy scheduler to demonstrate a viable solution. We also evaluate an optimistic scheduler that relaxes some of the routing constraints. Our measurements of the optimistic schedules reveal performance headroom that more sophisticated heuristics might try to harvest.

While previous work has contended with the relationship between programmability and efficiency [15, 28, 36], it has always presented the programmable designs as having a cost relative to the specialized ones. To the best of our knowledge, this is the first study to demonstrate comparable performance per area.

2 Q100 BACKGROUND

The Q100 [45, 46] is a spatial architecture that accelerates relational analytic queries. It achieves speedups over a CPU by turning software operations into hardware ones, processing multiple columns at a time, and pipelining operations across fields in a column. For background, we describe the key properties of the architecture and its toolchain. This architecture is more fully detailed in other papers [23, 45, 46].

Architecture. The accelerator, depicted in Figure 1, contains a collection of heterogeneous hardware modules, called *tiles*. In all, there are 9 types of Q100 tiles, each implementing a particular relational operator (e.g., a *Join*, *Sort*). The minimal Q100 design has one instance of each tile, but the desirable designs will have multiple. The original Q100 paper carefully explored tile mix as it is a key determinant of performance [45]. Tiles communicate directly between producer and consumer via an on-chip network which streams columns of data record by record. This network allows any tile to communicate with any other. The original Q100 paper [45] used a mesh topology, but subsequent work presented a methodology to automatically derive custom topologies that exploit the communication patterns of the workload (e.g., *Aggregator* is frequently a consumer of *Joiner*'s results) [23]. In the Q100, only dedicated *Reader* and *Writer* tiles, called stream buffers in the original paper, can speak to memory. *Readers* and *Writers* appear to the other tiles as standard data producers and consumers respectively.

Query Planning. As in any database system, SQL queries must be compiled to a query plan prior to execution. For this, we cannot simply use a standard DBMS plan as it is liable to include unsupported operations.

A Q100 query plan is a directed acyclic graph in which each node indicates an operation supported by the Q100 hardware, and edges indicate producer-consumer dependencies between them.

We have developed an automated SQL compiler that generates Q100 query plans as depicted in Figure 2. This compiler makes every attempt to use the tiles available on the target system. For this reason, synchronized subqueries, i.e. nested queries that use values from the outer query, are implemented joining the parent and nested query results. Standard database optimizations such as re-ordering of joins are applied in the flow.

Because a plan may require more operators than an accelerator has available, the subsequent scheduling step will divide the execution of the plan into a series of sequential steps.

Scheduling. Query plans are next divided into a series of temporal steps for execution. This schedule must respect the order of operations indicated in the plan, i.e. a column's producer must execute before or in parallel to its consumer. Moreover, none of the steps can exceed the available resources of the target Q100 instance. If, for example, the target contains five sorter tiles, the scheduler can not schedule six sorts in a step. If an instruction in the plan operates on more input columns than the tiles support, the scheduler will automatically split it into multiple instructions. Lastly, the scheduler maps each operation to a specific physical tile for execution.

Like most DBMS query planners, this scheduler must execute in real time; scheduling for the next time step while the Q100 device executes the current one. This produces accurate estimates of column size that in turn affect scheduling decisions. Exhaustive approaches [30] are thus not viable. Instead the scheduler uses a greedy longest-job-first heuristic which on average achieves within 5% of the best schedule obtained via a semi-exhaustive search [23] for the TPC-H benchmark. When multiple tiles are available to execute a given instruction, the scheduler selects the tile that requires the fewest network hops to route all of the input data streams.

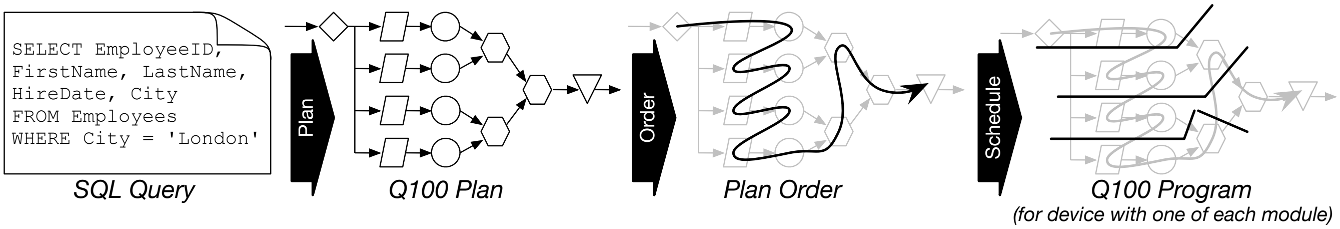


Figure 2: Our toolchain is able to automatically compile SQL queries into query plans that can execute on the Q100. As a first step for execution, operations are re-ordered with the objective of producing better schedules that group operations with similar latency. This reordering must still satisfy producer-consumer relationships between instructions. Finally, operations are mapped to a physical module in the device. Since a generic query plan might call for more resources than available in the hardware, the scheduler divides the execution into sequential steps. The mapping process is aware of the NoC topology and tries to minimize the number of hops needed to route all data flows.

3 MOTIVATION

In the Q100 architecture we find a tension between the variability of the computational requirements of the workload and the immutability of the fixed-function tiles. SQL queries can vary in their relational algebra operators, the order of the operators, and the dependencies between them. While the tiles of the Q100 are individually highly efficient, they require external orchestration to support variable demand, leading to two phenomena that decrease overall efficiency.

First, runtime utilization of the Q100 tiles is low. Figure 3 shows the average idle tile area for each TPC-H query. Since a Q100 design can be configured with an arbitrary tile mix, Figure 3 shows measurements from a set of Pareto-optimal designs, hence the error bars. The reason for the idleness is the variability in computational requirements within and across TPC-H queries. While clock and power gating can recoup most of the power associated with unused area, it still represents a resource that is not contributing to performance.

Table 1 enumerates four forms of variability, illustrating the reduced utilization that arises when an individual query meets an architecture designed for the general case query. Tiles may not be needed in a given time step, they could process less data than others and thus have to wait, they may be slowed by congestion in NoC links, or finally, an instruction might utilize fewer inputs than the hardware tile provides, leaving a fraction of the tile idle.

In Figure 4 we see all these factors at play in the TPC-H queries. Each query exercises the Q100 system in a different way, resulting in a different usage pattern. As in Figure 3, we profile a Pareto-optimal set of Q100 designs. The bigger error bars here reflect the diversity of fits of query to hardware.

The second hidden overhead in the heterogeneous, fixed function Q100 is the interconnect. In the Q100 a large fraction of the area goes to the interconnect that supports all-to-all communication (30-50% for Pareto optimal tile mixes). Moreover, links can become oversubscribed and therefore slowdown the entire computation (Table 1). Our previous work on interconnect specialization techniques [23] shows that is possible to reduce interconnect size and/or conflicts. However, for this study we rely only on standard topologies since the specialization techniques we developed create an heterogeneous NoC which might affect placement and routing significantly. Since we are evaluating a large number of designs, placing and routing each one would be prohibitive.

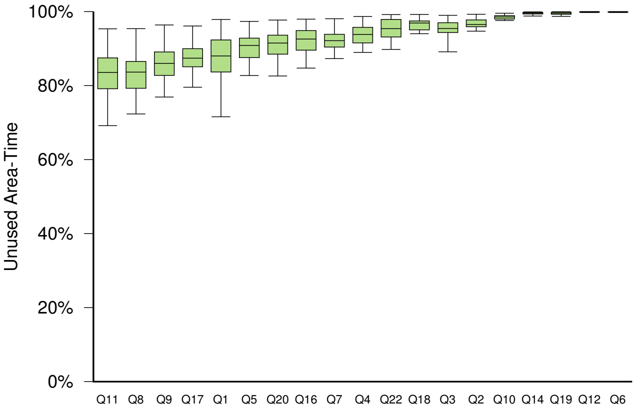


Figure 3: A large fraction of the Q100 area is idle when executing TPC-H queries.

However, the Q100 architecture also suggests an opportunity. When implemented in hardware, relational algebra operators share similar datapaths. For example, aggregators contain an adder similar to the one found in the ALU. The compare and swap logic in a sort tile is similar to the one in a merge. This suggests that a compound tile, configurable to perform all operations, might be a viable alternative. This is the approach we take with the homogeneous system described in the next section.

4 HOMOGENEOUS ARCHITECTURE

We have developed a homogeneous, programmable dataflow architecture, that supports the same relational operators as the Q100. This homogeneous system, depicted in Figure 5 is composed by a grid of small programmable processing elements (or PEs). Each PE is connected to four neighbors via latency insensitive channels. Data can flow horizontally in either direction, but vertically it travels only top to bottom, creating a systolic-array-like system. A row of *Reader* elements sits at the top of the array, and a row of *Writer* elements sits at the bottom. These units behave like their Q100

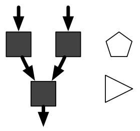
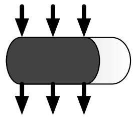
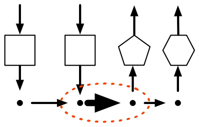
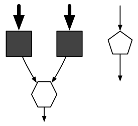
	**Idle tiles** occur when the query does not use all available tiles. This results from variation between queries or phases of a query. For example, analytical queries tend to filter inputs at the start and aggregate at the end, leaving aggregators idle at the start and filters idle at the end.
	**Input mismatch** happens when an operator processes fewer inputs than are provisioned in a tile. For example, a sorter that can process up to six inputs is over-provisioned for sorting two. Roughly one third of that tile will actually be in use. This happens, for example, in Q22 which operates on a small number of columns from each table.
	When a link in the interconnection network is in high demand, there is **interconnect congestion**, and all tiles involved in the congested flows are slowed down. Since each time step finishes when the last tile completes, this network congestion can ripple out and degrade performance. Congestion tends to occur when there are many active tiles, as seen in Figure 4 where congestion is highest when idle tiles are lowest.
	**Load imbalance** arises when instructions with different running times or duty cycles are scheduled in the same time step. Consider, for example, a highly selective filter that feeds some consumer. The consumer will sit mostly idle awaiting the occasional token that passes through the filter.

Table 1: Four ways that query variability contributes to reduced utilization of a fixed set of resources. In the cartoons, darker shading indicates higher utilization, and arrow thickness indicates data volume.

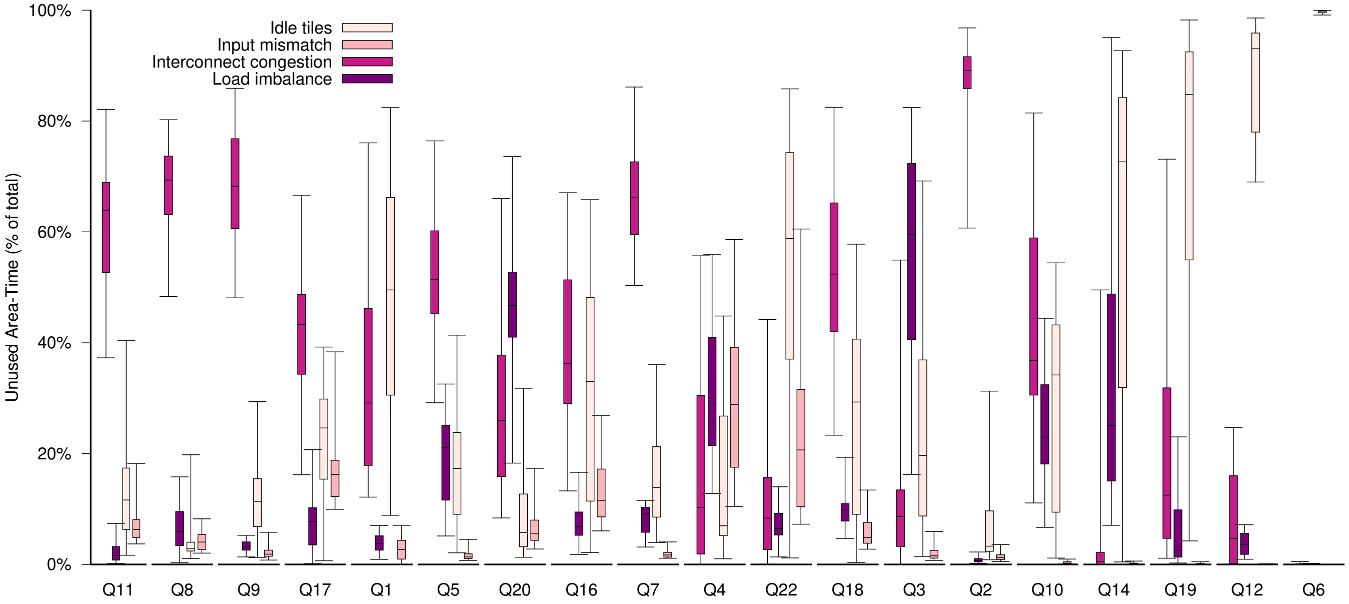


Figure 4: Attribution of idle area-time in the Q100 to different causes.

counterparts and are responsible for reading and writing memory. Streams of data are read from the top, percolate through, and are received at the bottom by *Writers* which store the results – sometimes final, sometimes intermediate – back to memory.

The detail on the right side of Figure 5 shows the three principal components of each PE. Data tokens arrive from either the North, East or West and are routed to the compute core. Data streams are 32 bits wide, the same as the Q100. The 8-bit control streams run parallel to the data streams and operate independently. These carry metadata and control information, e.g. boolean conditions evaluated by one module on a given column that have to be transmitted to another processing element to filter a second column.

The compute core of a PE can consume up to two data and control tokens per cycle and produce up to two data tokens and two

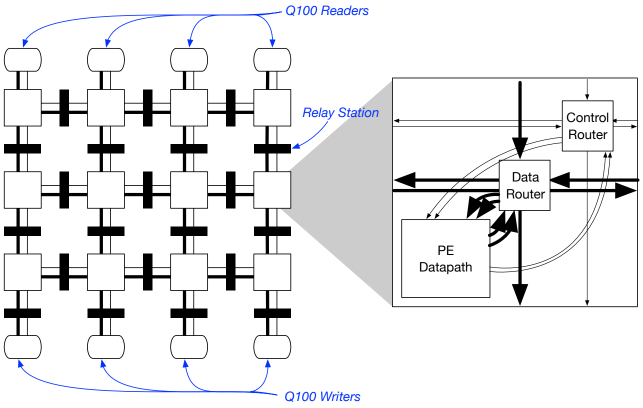


Figure 5: Homogeneous system architecture. Each processing element operates independently. Within each PE compute and communications modules are also independent. The memory subsystem is equivalent to the one used in the Q100.

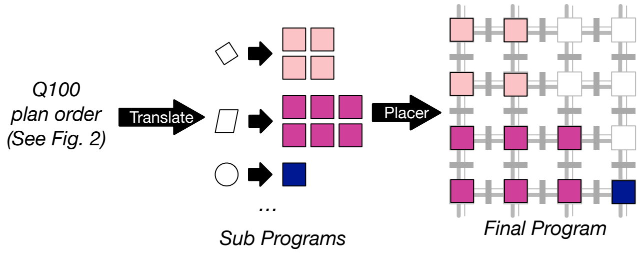


Figure 6: Compilation for our homogeneous design re-uses the Q100 fronted. After an ordering is found for the Q100 instructions each one is compiled into a graph of micro-ops. These are then mapped – over multiple time steps – to the processing elements in the homogeneous system.

output control tokens. The compute core is configurable, executing essentially one instruction continuously until the device is reprogrammed. Similarly to the Triggered Instructions [32] dataflow architecture, instruction in the homogeneous design fire when all necessary inputs are available, and there is space in output buffers of the PE.

In total, there are 43 instructions in the homogeneous ISA. These comprise standard arithmetic instructions, e.g. addition, subtraction, multiplication, as well as more complex ones that target database workloads, e.g., a merge of two input streams conditional on an input control stream. Like the Q100, the homogeneous system supports only integer operations and assumes that either the query can be executed with fixed point precision or the CPU will have to step in. While the PE of the homogeneous system could easily be extended to support more operations, so too could the Q100 with new tiles. Any differences in coverage was liable to be an artifact of our choices, so for these comparisons we match coverage, weighing the merits of each architecture on precisely the same workload.

Each output port of a PE can be connected via a configurable interconnect to the input port of another PE. The destination PE need not be a neighbor, provided the interconnect is able to route the flow without conflicts with other communications. Multicast to multiple PEs is possible and is supported in the routers which duplicate packets and transmit them to multiple outputs at once. The routing logic in each PE is separate from the compute core so that a PE can route tokens while computing. The network uses circuit switched flow control, and any tokens received on a particular network input are forwarded to the configured output until the design is reconfigured.

Since PEs have equal computational capabilities they can all be employed for any operation. Relational algebra operators in a query plan can be mapped to any combination of PEs, with only the necessary number of PEs used for each operation.

Since all PEs are equipotent, we try to schedule instructions to neighboring PEs. This allows us to reduce power consumption of the interconnection network. Furthermore, it allows us to use more area/power efficient circuit switched routers compared to the packet switched routers used by the Q100, 5.9X smaller and 3.8X lower power.

5 HOMOGENEOUS COMPILATION BACKEND

To ensure a controlled comparison, we re-use the front end of the Q100 compiler for both systems. The homogeneous backend, depicted in Figure 6 picks up from the point where the Q100 query plan has been sorted according to the longest-job-first heuristic. Each operator that appears in a Q100 query plan is translated into a graph of instructions for the homogeneous PEs. We will call these small instructions micro-ops, and "instructions" will continue to mean Q100 operations. These micro-ops have inter- and intra-instruction producer consumer relations. The number of micro-ops generated per Q100 instruction depends both on the instruction type and the number of its input/outputs. For example, a Join instruction will translate to as many micro-ops as it has input columns.

Mapping micro-ops to a homogeneous device is more challenging than mapping Q100 instructions to a Q100 device, as any PE in the homogeneous device can execute any micro-op. There are a large number of possible mappings for each instruction and each one might affect the mapping of subsequent instructions that operate on the same data.

We generate a set of mappings from micro-ops to PEs using simulated annealing, i.e. repeatedly applying random permutations to the best known solution. A mapping is invalid if data dependencies cannot be routed. For simplicity, the backend requires that all micro-ops in an instruction be scheduled at the same time.

This process is repeated for each Q100 instruction – in the ordering provided by the Q100 scheduler – until there are no more PEs or instructions available. The lack of available instructions does not automatically mean that the query is done. Due to the producer-consumer relations in a query plan, only a subset of instructions are eligible for scheduling at any given time.

As a cost function for each evaluated mapping, we count the number of network routing rules that must be introduced. Minimizing this metric reduces both router utilization and path length. It will prefer mappings where producer and consumer micro-ops are close together as long paths will require adding more rules along the route.

Given the large space of possible mappings, of which many are nonsensical (e.g. mapping two connected micro-ops to opposite corners of the homogeneous mesh), we kickstart the process by using predetermined "shapes" for each Q100 instruction. These shapes are mapped rigidly (if possible) to the device in the *first_map* method in the pseudocode below. Later the best mapping has one or more of its instruction swapped with a random adjacent location in the *tweak_map* method. This pass, similar to a peep-hole optimization, could cause the introduction of empty spaces in the mesh or exchange the location of two micro-ops, eliminating possible routing conflicts and/or reducing cost. After a predetermined number of mappings are evaluated, the least expensive valid mapping is selected and the PEs marked as used.

Algorithm 1 Homogeneous Micro-op Mapping

```
procedure MAP($inst, device$)
    $inst\_graph \leftarrow inst\_to\_graph(inst)$
    $best\_map \leftarrow null$
    $best\_cost \leftarrow Inf$
    for $i \leftarrow 1, max\_tries$ do
        $map \leftarrow null$
        if $i = 1$ then
            $map \leftarrow first\_map(inst\_graph, device)$
        else
            $map \leftarrow tweak\_map(best\_map, device)$
        end if
        if $map = null$ then
            break
        else
            $tries \leftarrow tries + 1$
            $cost \leftarrow compute\_cost(map, device)$
            if $cost < best\_cost$ then
                $best\_map \leftarrow map$
                $best\_cost \leftarrow cost$
            end if
        end if
    end for
    return $best\_map$
end procedure
```

We recognize that by considering only one instruction at a time, this algorithm may find local minima. It could be beneficial to change the order in which instructions are scheduled or use more complex heuristics rather than random choice to create possible mappings. While we do not evaluate more complex solutions, we will quantify in Section 7 how much performance may be left on the table due to sub-optimal mapping.

6 METHODOLOGY

In order to evaluate and compare the Q100 and homogeneous architectures we have developed an RTL implementation for both. Both architectures share a common cycle level simulator infrastructure as well as identical synthesis flow.

Q100 Hardware. The RTL implementation of the Q100 has been refreshed with respect to the original paper [45] with the main difference being the presence of a *Merge* tile instead of a *Partitioner* tile. This slightly modified design has already been utilized in follow up work on NoC specialization [23].

For the Q100 NoC, we use an open-source router implementation [2]. NoCs for the Q100 are packet switched and use reverse credit flow. Reverse credit flow and additional buffer slots are used in the Q100 to avoid application deadlock when different modules share a link. We decided not to use previously published techniques to optimize the NoC topology of Q100 designs [23] as it would be infeasible to place and route a custom NoC for each of the thousands of Q100 tile mixes that we wish to evaluate. Therefore, we restrict our analysis to standard NoC topologies.

Homogeneous Hardware. The processing element for the homogeneous architecture has been designed for the purpose of this study. The NoC for the homogeneous design uses a single circuit switched plane each for its data and control networks. The router for this circuit switched NoC has been developed in house.

While the different NoCs may at first appear to complicate the comparison, we have tried to pick the most suitable network for each architecture. The Q100 expects that any tile be able to stream data to any other tile and our measurements suggest the network already limits some queries on the Q100. Therefore, the circuit switched network in the homogeneous design would likely hurt performance even more as only a few circuits could be established before all of the physical links were occupied. This would prevent the Q100 from maximizing its utilization of the available physical tiles. Similarly a packet switched NoC is unnecessary for the homogeneous design as communication mostly happens between endpoints that are spatially close.

Logic Synthesis and Power Estimation. We synthesized netlists for each module and PE using Synopsys Design Compiler (2013.12-SP1) and TSMC 65 nm general-purpose CMOS standard cells. For both designs, we ran at maximum frequency using nominal Vdd with the low threshold voltage version of the libraries. We use clock gating for both systems and therefore only consider static power consumption for idle processing elements, tiles, or routers during a time step. Since a significant fraction of the area could be idle depending on the input query, one could argue that both systems would benefit from power gating on a module by module basis. This would result in a non trivial area and power overhead due to the small size of most Q100 modules and the homogeneous PE. Due to the small impact of static power at our operating point (less than 10% of total power), and the aforementioned issues, we decided not to use power gating in either system.

For each hardware module, we extracted gate-level activity factors. The resulting fully annotated netlist-level VCD was used as a dynamic activity input into for the Synopsys PrimeTime (2013.12-SP1) fine-grain power and timing modeling software. We report

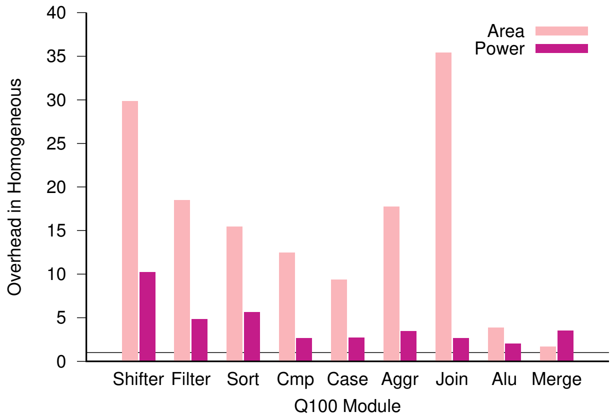


Figure 7: On a per operator per column basis the homogeneous design incurs area and power overheads. Here the Q100 tiles are sorted from smallest to largest, and we find that the homogeneous overheads is largest for the smaller tiles.

pre-extraction results, but use a uniform wire-load model of 2 fF for all local nets. We characterize each module in isolation and later combine them to compute the total static and dynamic power. In each time step we use the computed static power for the inactive modules and the dynamic power for the active ones.

Simulation Infrastructure. All cycle counts are obtained using an in house cycle-level simulator. The scheduler described in the previous section is embedded in the simulator and governs the execution of each query plan. For all homogeneous experiments, the mapping algorithm described in Section 4 attempts at most 100 diverse mappings.

Since our homogeneous instruction mapping algorithm is not exhaustive we consider an "ideal" homogeneous mapper. This mapper takes into account only the number of available processing elements and does not evaluate the routing necessary to satisfy each micro-op producer consumer relationship. This effectively sets an upper limit on the performance of a mapping algorithm for our homogeneous device. This upper bound is equivalent to assuming as many circuit switched interconnection planes as necessary to ensure maximum throughput. Notice that micro-ops are arranged in a partial order, similarly to instruction in a Q100 query plan. Therefore they are agnostic to the latency in communication between them.

All our tests are performed on TPC-H, the most widely used benchmark for query processing analytic workloads [4], both in academia and industry. We used scaling factor 0.01 (a 10MB database) to have manageable simulation times. Our infrastructure supports 19 out of 22 queries, and we do not account for time spent in unsupported operations. Since both systems use the same streaming memory interface and subsystem, changing the size of the workload should affect both architectures equally. Finally, we assume that a single query is executed by either accelerator at a time. Therefore, latency correlates directly to throughput of the system. We suspect this assumption is reasonable as having multiple queries run on the Q100 would exacerbate network congestion and complicate scheduling, but might mitigate the observed underutilization. However, we have not explicitly measured those scenarios.

Design Space Exploration. Since Q100 performance is highly tied to the selection of hardware modules, we evaluated more than 4000 Q100 configurations and focus on the Pareto optimal points for further evaluation. For the homogeneous design we sweep mesh size from 10x10 to 48x48 PEs. For both designs we set an upper limit of 120mm^2 and 10W of power. We believe these are reasonable ranges considering that this power budget excludes clock tree and parasitics (since we do not perform place and route), and the memory subsystem (which is identical in the two designs). Our upper limit of 120mm^2 for the die is fairly large, and close to that of a two core Intel Xeon 3050 processor at 65nm. While our analysis considers only a single accelerator die, other techniques could derive a Total Cost of Ownership (TCO) optimal ASIC [25].

7 COMPARISON OF Q100 AND HOMOGENEOUS DESIGN

In this section we want to understand which accelerator design has better performance given an area/power budget and why. Figure 7 shows how, taken singularly, each Q100 module is more efficient than a homogeneous PE. This is of course an intuitive result since each homogeneous processing element can be programmed to perform any operation. Furthermore, the Q100 maximum operating frequency is higher (1100MHz) than that of the homogeneous design (950 MHz).

However, when looking at the whole system, we find their performance is similar. The homogeneous design recoups lost per-operator performance with increased utilization and more efficient communication. From Figure 8 we see that the homogeneous is comparable to the Pareto optimal Q100 designs in terms of performance for most area budgets. But, when the area budget is small, it outperforms the Q100. This edge at low area makes sense, as the homogeneous resources are interchangeable and readily scale down. At the small end, the heterogeneous design is more restrictive as it requires at least one of each tile. We find similar results when power is the objective resource (Figure 9). The homogeneous design provides comparable performance at most power budgets and improvements under small ones. In Figure 10 we can see a more detailed comparison on a query by query basis that shows the relationship between the two accelerators is query dependent.

Figure 11 differentiates the Q100 designs by interconnect topology. Devices that use a ring tend to be smaller but incur penalties in terms of running time. While it is possible to construct Q100 designs with a wide range of relative interconnect and tile areas, the Pareto optimal ones all dedicate a relatively large share of the design area to the NoC. In all cases, this is a larger proportion of area than in the homogeneous designs, where the proportion is fixed via the one-to-one pairing of routers and PEs. In Figure 12 we can also see that the reduced NoC area in homogeneous corresponds to a lower ratio of power spent in this component. The homogeneous design invests fewer resources on communication than the heterogeneous Q100 does.

Lastly, the homogeneous design is able to use a larger fraction of its area at any given time. Figure 13 shows that utilization is

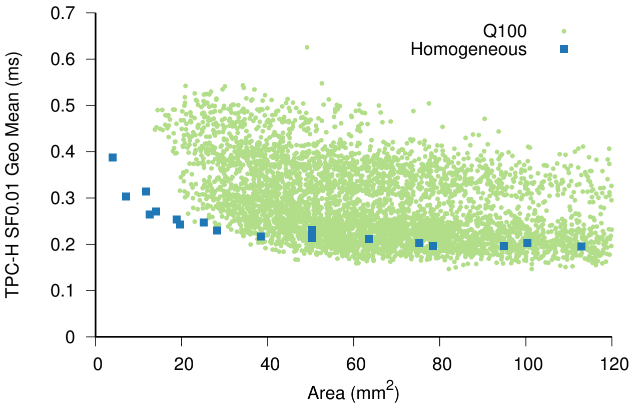


Figure 8: Performance/Area comparison of homogeneous and Q100 implementations

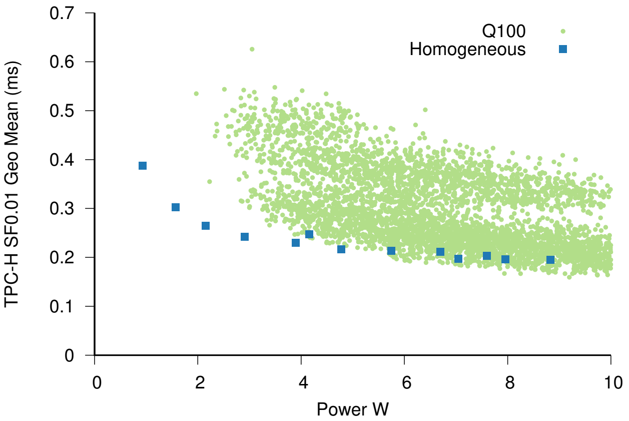


Figure 9: Performance/Power comparison of homogeneous and Q100 implementations.

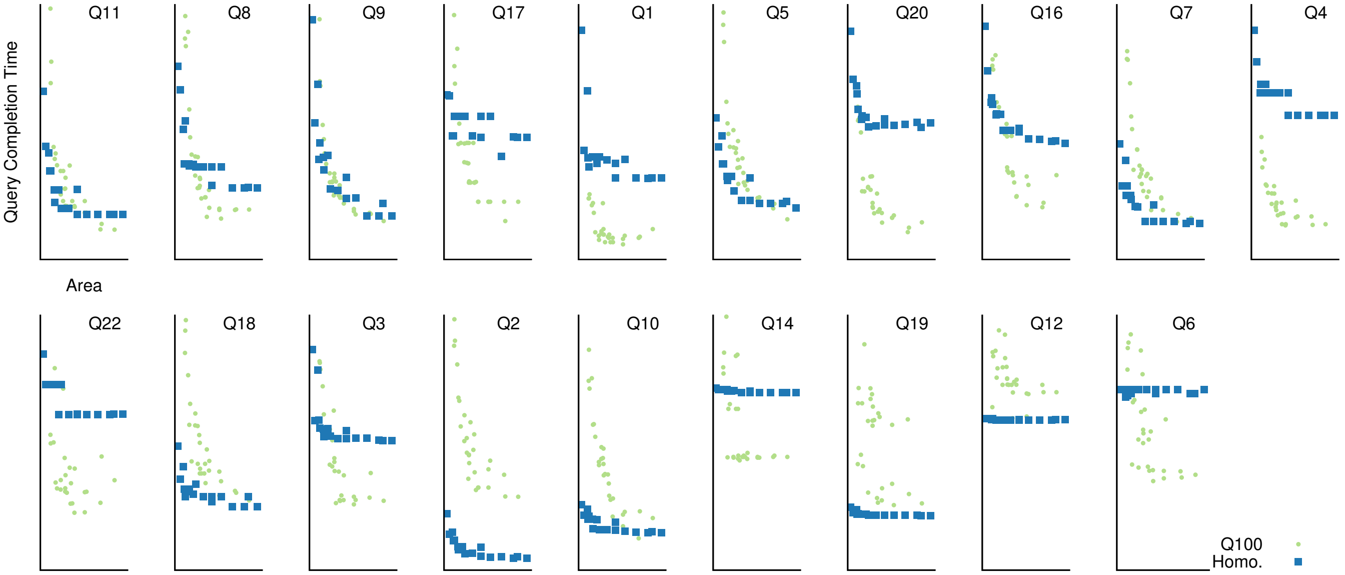


Figure 10: Relative performance of the Q100 and homogeneous designs depend on the query.

on average higher on a homogeneous device than on Q100. If an ideal mapping of micro-ops to PE were attainable at every time step, utilization would be even higher. Figure 14 shows the speedup attainable by the ideal mapping algorithm over our probabilistic algorithm. There is significant variability depending on the size of the homogeneous device considered with larger system benefiting more from the increased utilization. While our scheduler is sub-optimal, what we think is interesting is that there is a large performance gap that could be bridged at least partially with software techniques.

In addition to increased tile utilization, there are other dynamics, not directly measured, but that are captured and contribute to the overall performance. The homogeneous design exposes communication routing explicitly. If an instruction cannot be routed in the network, it is forced to a different time step. By contrast, the Q100 architecture does not expose limitations on communication. It instead exposes function, requiring instructions to execute on their corresponding functional tiles. The homogeneous hardware, for all its rigidity around communication, is flexible about computation. The hardware does not specify the type or size of any operator, instead exposing an ISA to program them.

As summarized in Table 2, our principal finding is that the programmability of the homogeneous design allows it to better adapt the architecture to variations in the workload, in this case variations across and within queries. While the heterogeneous approach

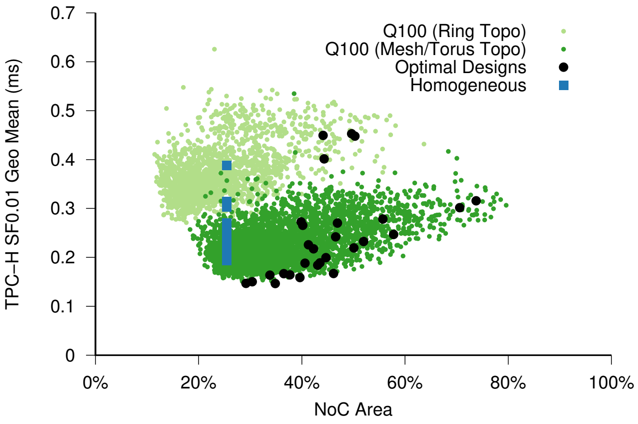


Figure 11: Fraction of area spent in the NoC. Pareto designs for the Q100 devote a larger fraction of their area to the NoC compared to homogeneous designs.

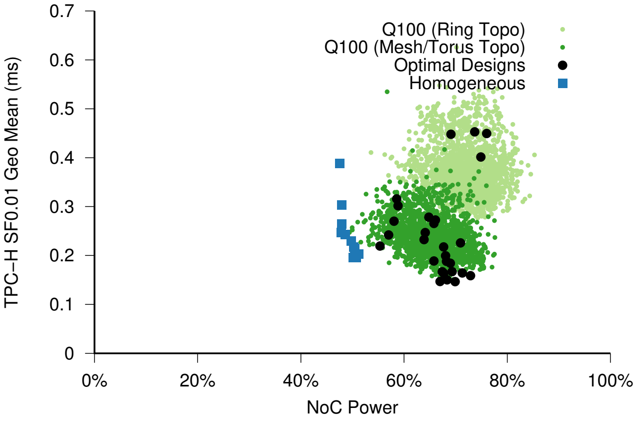


Figure 12: Homogeneous designs spend a smaller fraction of their power in the NoC.

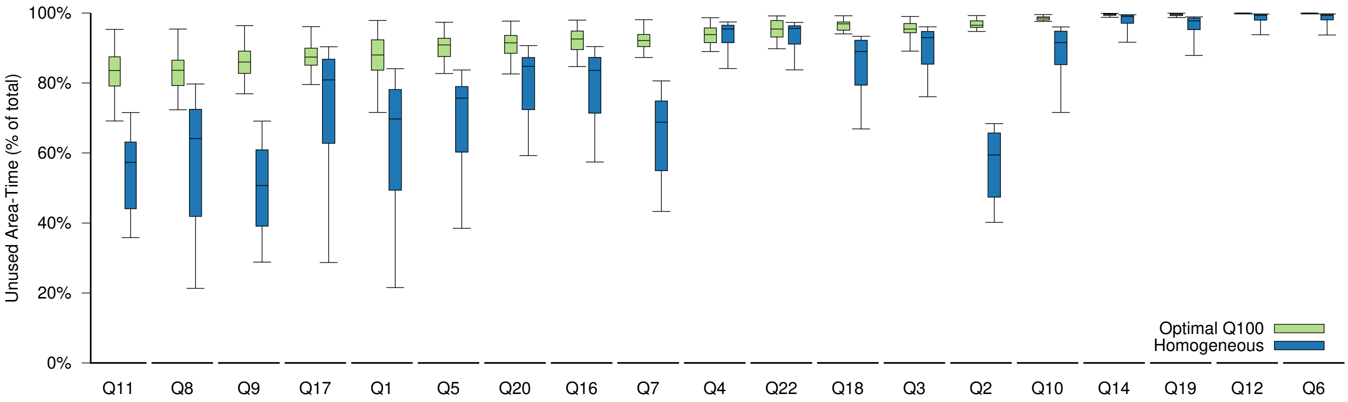


Figure 13: Unused Area-Time by both devices under exam. We can see that homogeneous makes a better use of the available resources.

is significantly more performant on a per operator basis, problems seem to crop up when composing a more general purpose mix of heterogeneous components to accelerate queries with varying demands. At that point, our results suggest that a homogeneous mix of general purpose components should be considered as an alternative. When area budgets are large, an accelerator with low utilization still ends up using a lot of resources, and we find the approaches are comparable. Lastly, we wish to highlight the scenarios that are not covered here. Most prominently, we offer no conclusion about accelerators running multiple queries at once or scenarios where the accelerator functionality varies.

8 RELATED WORK

The tension between programmability and efficiency is a longstanding subject of study in the architecture community. Hameed et al.'s seminal paper analyzed the benefits of increasing specialization [15]. As their title suggests they analyze the inefficiencies of software running on general purpose CPUs, whereas we analyze the inefficiencies that can arise in specialized hardware. Subsequent work by the same group highlighted how it is possible to create an architecture for convolution workloads that approximates the performance of a fully specialized hardware design [36]. Nowatzki et al. performed a study re-targeting proposed accelerators to a custom accelerator substrate they designed [28]. This work shows a significant cost (4X area and power) to pay for programmability. There has been previous work that highlighted diminishing returns for ISA customization [11]. While there are similarities in message, the setting is restricted to microarchitectural changes to a processor pipeline, instead of the much larger architectural design space we have evaluated. A common thread in all of these studies is the notion that programmability has a cost; creating a programmable architecture for a workload will incur a hopefully modest penalty compared to an ASIC implementation. To the best of our knowledge we are the first to show how programmability can be beneficial for performance when the workload at hand has sufficient computational variability.

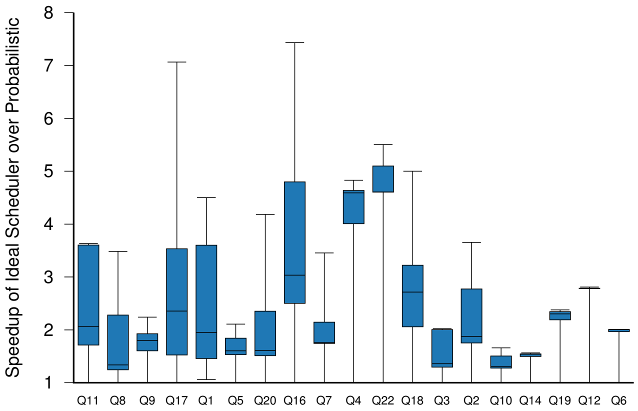


Figure 14: Comparison of our probabilistic homogeneous scheduling algorithm with an ideal scheduler. Significant performance improvements could be obtained by better software scheduling.

Homogeneous wins	→	Small designs, for the reasons outlined in Table 1 and the ability to scale down more gracefully
Heterogeneous wins	→	Scenarios where the target workload is dominated by a small number of important or frequent queries that are known in advance
Doesn't matter	→	Large designs, where both architectures are subject to diminishing returns
Don't know	→	Scenarios where multiple queries can run concurrently
	→	Scenarios with different implemented functionality in the two designs

Table 2: Summary of findings.

Databases and their operations are established acceleration targets. Kung et al. first suggested using systolic arrays to accelerate database workloads [21] in 1980, however, high performance improvement of CPUs year after year made the development of ASICs for this kind of computation unappealing [5]. The end of Dennard scaling spurred the more recent interest in this topic. Single critical targets including partitioning [44] or hash-joins [20] have been evaluated for acceleration as ASICs with high expected benefits. A recent proposal used a similar architecture to ours [6] but was limited to acceleration of nested loop join in analytical query processing. Industry too, has examined database acceleration. Oracle, for example, proposed a many core architecture [1]. It differs from our design as its cores have a standard Von-Neumann architecture and data is received from a DMA engine rather that passed in a dataflow manner from one PE to the other.

There has been also interest in FPGA acceleration of DBMSs with proposals targeting the storage engine [17, 43] to entire queries [8]. In industry, Baidu recently showed results from an FPGA database accelerator that closely resembles the Q100 [31] while FPGAs are now routinely deployed in datacenters [35]. FPGAs provide clear benefits in a datacenter scenario since they can be re-configured to adapt to changing workloads. On the other hand, if maximum performance or power is necessary, ASICs remain the best solution, and it seems likely that they will all co-exist in datacenter deployments.

There have already been deployments of ASICs in the datacenter, most notably targeting machine learning inference [16] or training [9]. As previously mentioned these are ideal workloads for acceleration due to the high compute per byte and their regular control flow structure. On the other hand, it has been shown that no single workload dominates in a datacenter, and there is a long tail of workloads [18]. These might include applications with high control flow divergence and that are memory latency bound such as information retrieval or memory throughput bound such as the OLAP workloads that we analyzed in this paper. Another interesting example is video transcoding at datacenter scale which has been analyzed recently [24]. While we might think of video transcoding as a perfect candidate for ASIC acceleration, that study highlighted how there is enough irregularity in the workload that current hardware accelerators risk imposing quality/performance tradeoffs.

Coarse-grained reconfigurable architectures – similar to the systolic array architecture we use here – have been an active area of research in computer architecture since the early 1990s. Interest in these architectures tends to wax and wane over time, but they remain a compelling design point to explore despite lack of widespread commercial adoption [3, 13, 14, 27, 29, 32, 33, 38–42, 47]. CGRAs differ from one another mainly in the way they control instruction execution on their PEs. Some require all operations and data movement to happen in global lockstep [13, 27, 40]. Others map circuit-switched dataflow networks onto an array of PEs with local control consisting only of if-then-else predication [14, 39, 41]. Finally, some designs give PEs complete local autonomy to execute sequential routines covering their designated portion of a spatial program [3, 32, 42, 47].

Stream-dataflow [29] and Triggered Instructions [32] are the closest proposals to the homogeneous architecture we evaluate here. Stream-dataflow operates explicitly on memory streams fed into and read from the reconfigurable fabric by external stream readers and stream writers [29]. Our homogeneous PEs use triggered-instruction-like operand-availability as a criterion for instruction scheduling [32] and can thus tolerate arbitrary data latencies. The tags that are added to each data channels are the closest in spirit to our independent control plane. Neither is optimized for database applications in the ways that our homogeneous design is.

9 CONCLUSIONS

Accelerator designers always face a choice as to how far and in what way to specialize their architecture. We have examined two approaches to architectural specialization for analytical query processing. The first embodied by the previously proposed Q100 accelerator [45], exposes a heterogeneous collection of highly-tailored functional units. The second exposes a homogeneous array of equipotent functional units. Although individual operators are far more efficient on the first accelerator, we find that the two approaches offer comparable performance at most area and power budgets. When resources are constrained, we find that the homogeneous approach is more efficient. The variability of computational requirements over time in the workload explains these results. The programmable architecture can adapt more easily to these changes leading to higher resource utilization and a lower cost for communication. We believe that the results presented here could extend to other applications that have similar computational properties. This would motivate further research in non Von-Neumann architectures together with their software support.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. This work was supported in part by National Science Foundation Grants (CCF-1065338 and CCF-1162124), a Qualcomm Innovation Fellowship, and a Google Faculty Award.

REFERENCES

[1] Sandeep R Agrawal, Sam Idicula, Arun Raghavan, Evangelos Vlachos, Venkatraman Govindaraju, Venkatanathan Varadarajan, Cagri Balkesen, Georgios Giannikis, Charlie Roth, Nipun Agarwal, and Eric Sedlar. 2017. A Many-core Architecture for In-memory Data Processing. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[2] Daniel Becker. 2018. EFFICIENT MICROARCHITECTURE FOR NETWORK-ON-CHIP ROUTERS. (07 2018).

[3] Brent Bohnenstiehl, Aaron Stillmaker, Jon J Pimentel, Timothy Andreas, Bin Liu, Anh T Tran, Emmanuel Adeagbo, and Bevan M Baas. 2017. KiloCore: A 32-nm 1000-processor computational array. *IEEE Journal of Solid-State Circuits* 52, 4 (2017).

[4] Peter Boncz, Thomas Neumann, and Orri Erling. 2014. *TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark.* Springer International Publishing.

[5] Haran Boral and David J. DeWitt. 1983. Database machines: an idea whose time has passed?. In *Database Machines: International Workshop*.

[6] Bingyi Cao, Kenneth A. Ross, Stephen A. Edwards, and Martha A. Kim. 2017. Deadlock-free Joins in DB-mesh, an Asynchronous Systolic Array Accelerator. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*.

[7] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A Cloud-scale Acceleration Architecture. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[8] Eric S. Chung, John D. Davis, and Jaewon Lee. 2013. LINQits: Big Data on Little Clients. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*.

[9] J. Dean, D. Patterson, and C. Young. 2018. A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution. *IEEE Micro* 38, 2 (Mar 2018). https://doi.org/10.1109/MM.2018.112130030

[10] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[11] Joseph A. Fisher, Paolo Faraboschi, and Giuseppe Desoli. 1996. Custom-fit Processors: Letting Applications Define Architectures. In *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO)*.

[12] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A Configurable Cloud-scale DNN Processor for Real-time AI. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*.

[13] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, and R Reed Taylor. 2000. PipeRench: A reconfigurable architecture and compiler. *Computer* 4 (2000).

[14] Venkatraman Govindaraju, Chen-Han Ho, Tony Nowatzki, Jatin Chhugani, Nadathur Satish, Karthikeyan Sankaralingam, and Changkyu Kim. 2012. Dyser: Unifying functionality and parallelism specialization for energy-efficient computing. *IEEE Micro* 32, 5 (2012).

[15] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. 2010. Understanding Sources of Inefficiency in General-purpose Chips. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[16] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1145/3079856.3080246

[17] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. 2015. BlueDBM: An Appliance for Big Data Analytics. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[18] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a Warehouse-scale Computer. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.

[19] Svilen Kanev, Sam Likun Xi, Gu-Yeon Wei, and David Brooks. 2017. Mallacc: Accelerating Memory Allocation. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[20] Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim, and Parthasarathy Ranganathan. 2013. Meet the Walkers: Accelerating Index Traversals for In-memory Databases. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[21] H. T. Kung and Philip L. Lehman. 1980. Systolic (VLSI) Arrays for Relational Database Operations. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

[22] Kevin Lim, David Meisner, Ali G. Saidi, Parthasarathy Ranganathan, and Thomas F. Wenisch. 2013. Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[23] Andrea Lottarini, Stephen A. Edwards, Kenneth A. Ross, and Martha A. Kim. 2017. Network Synthesis for Database Processing Units. In *Proceedings of the Annual Design Automation Conference (DAC)*. https://doi.org/10.1145/3061639.3062289

[24] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. 2018. Vbench: Benchmarking Video Transcoding in the Cloud. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[25] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. 2016. ASIC Clouds: Specializing the Datacenter. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[26] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2011. Dremel: Interactive Analysis of Web-scale Datasets. *Commun. ACM* 54, 6 (June 2011).

[27] Ethan Mirsky, Andre DeHon, et al. 1996. MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources..

In *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[28] T. Nowatzki, V. Gangadhan, K. Sankaralingam, and G. Wright. 2016. Pushing the limits of accelerator efficiency while retaining programmability. In *International Symposium on High Performance Computer Architecture (HPCA)*.

[29] Tony Nowatzki, Vinay Gangadhar, Newsha Ardalani, and Karthikeyan Sankaralingam. 2017. Stream-dataflow acceleration. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[30] Tony Nowatzki, Michael Sartin-Tarm, Lorenzo De Carli, Karthikeyan Sankaralingam, Cristian Estan, and Behnam Robatmili. 2013. A General Constraint-centric Scheduling Framework for Spatial Architectures. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

[31] Jian Ouyang, Wei Qi, Yong Wang, YichenTu, Jing Wang, and Bowen Jia. 2016. SDA: Software-Defined Accelerator for General-Purpose Distributed Big Data Analysis System. In *IEEE Hot Chips Symposium (HCS)*.

[32] Angshuman Parashar, Michael Pellauer, Michael Adler, Bushra Ahsan, Neal Crago, Daniel Lustig, Vladimir Pavlov, Antonia Zhai, Mohit Gambhir, Aamer Jaleel, Randy Allmon, Rachid Rayess, Stephen Maresh, and Joel Emer. 2013. Triggered Instructions: A Control Paradigm for Spatially-programmed Architectures. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[33] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2017. Plasticine: A reconfigurable architecture for parallel patterns. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[34] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[35] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2016. A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services. *Commun. ACM* 59, 11 (Oct. 2016).

[36] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A. Horowitz. 2013. Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing. In *Proceeding of the Annual International Symposium on Computer Architecuture (ISCA)*.

[37] Raghu Ramakrishnan and Johannes Gehrke. 2003. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA.

[38] Thomas J. Repetti, João P. Cerqueira, Martha A. Kim, and Mingoo Seok. 2017. Pipelining a Triggered Processing Element. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[39] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W Keckler, and Charles R Moore. 2003. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. IEEE.

[40] Roy A Sutton, Vason P Srini, and Jan M Rabaey. 1998. A multiprocessor DSP system using PADDI-2. In *Design Automation Conference, 1998. Proceedings*. IEEE.

[41] Steven Swanson, Andrew Schwerin, Martha Mercaldi, Andrew Petersen, Andrew Putnam, Ken Michelson, Mark Oskin, and Susan J Eggers. 2007. The wavescalar architecture. *ACM Transactions on Computer Systems (TOCS)* 25, 2 (2007).

[42] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, et al. 2002. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE micro* 22, 2 (2002).

[43] Louis Woods, Zsolt István, and Gustavo Alonso. 2014. Ibex: An Intelligent Storage Engine with Support for Advanced SQL Offloading. *VLDB* (2014).

[44] Lisa Wu, Raymond J. Barker, Martha A. Kim, and Kenneth A. Ross. 2013. Navigating Big Data with High-throughput, Energy-efficient Data Partitioning. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*.

[45] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. 2014. Q100: The Architecture and Design of a Database Processing Unit. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[46] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. 2015. The Q100 Database Processing Unit. *IEEE Micro* (May 2015).

[47] Zhiyi Yu, Michael J Meeuwsen, Ryan W Apperson, Omar Sattari, Michael Lai, Jeremy W Webb, Eric W Work, Dean Truong, Tinoosh Mohsenin, and Bevan M Baas. 2008. AsAP: An asynchronous array of simple processors. *IEEE Journal of Solid-State Circuits* 43, 3 (2008).