Design Space Exploration of Accelerators for Warehouse Scale Computing

Andrea Lottarini

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2019

ABSTRACT

Design Space Exploration of Accelerators for Warehouse Scale Computing

Andrea Lottarini

With Moore's law grinding to a halt, accelerators are one of the ways that new silicon can improve performance, and they are already a key component in modern datacenters. Accelerators are integrated circuits that implement parts of an application with the objective of higher energy efficiency compared to execution on a standard general purpose CPU. Many accelerators can target any particular workload, generally with a wide range of performance, and costs such as area or power. Exploring these design choices, called Design Space Exploration (DSE), is a crucial step in trying to find the most efficient accelerator design, the one that produces the largest reduction of the total cost of ownership.

This work aims to improve this design space exploration phase for accelerators and to avoid pitfalls in the process. This dissertation supports the thesis that *early design choices – including the level of specialization – are critical for accelerator development* and therefore *require benchmarks reflective of production workloads*. We present three studies that support this thesis. First, we show how to benchmark datacenter applications by creating a benchmark for large video sharing infrastructures. Then, we present two studies focused on accelerators for analytical query processing. The first is an analysis on the impact of Network on Chip specialization while the second analyses the impact of the level of specialization.

The first part of this dissertation introduces `vbench`: a video transcoding benchmark tailored to the growing video-as-a-service market. Video transcoding is not accurately represented in current computer architecture benchmarks such as SPEC or PARSEC. Despite posing a big computational burden for cloud video providers, such as YouTube and Facebook, it is not included in cloud benchmarks such as CloudSuite. Using vbench, we found that the microarchitectural profile of video transcoding is highly dependent on the input video, that SIMD extensions provide limited benefits, and that commercial hardware transcoders impose tradeoffs that are not ideal for cloud video providers. Our benchmark should spur architectural innovations for this critical workload. This work shows how to benchmark a real world warehouse scale application and the possible pitfalls in case of a mischaracterization.

When considering accelerators for the different, but no less important, application of analytical query processing, design space exploration plays a critical role. We analyzed the Q100, a class of accelerators for this application domain, using TPC-H as the reference benchmark. We found that the hardware computational blocks have to be tailored to the requirements of the application, but also the Network on Chip (NoC) can be specialized. We developed an algorithm capable of producing more effective Q100 designs by tailoring the NoC to the communication requirements of the system. Our algorithm is capable of producing designs that are Pareto optimal compared to standard NoC topologies. This shows how NoC specialization is highly effective for accelerators and it should be an integral part of design space exploration for large accelerators' designs.

The third part of this dissertation analyzes the impact of the level of specialization, e.g. using an ASIC or Coarse Grain Reconfigurable Architecture (CGRA) implementation, on an accelerator performance. We developed a CGRA architecture capable of executing SQL query plans. We compare this architecture against Q100, an ASIC that targets the same class of workloads. Despite being less specialized, this programmable architecture shows comparable performance to the Q100 given an area and power budget. Resource usage explains this counterintuitive result, since a well programmed, homogeneous array of resources is able to more effectively harness silicon for the workload at hand. This suggests that a balanced accelerator research portfolio must include alternative programmable architectures – and their software stacks.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I feel very lucky to be writing these acknowledgments and I wouldn't be in this position if it wasn't for the people that helped me along the way. First of all, I want to thank my advisor, Martha Kim. She taught me how to be a better thinker, writer, and presenter. Always without losing her kindness and positive outlook, she relentlessly went over my drafts, providing constructive and helpful feedback (as well as adding all the commas and s's that I inevitably missed). I am very grateful to have met her and to have had the privilege to work with her.

I had several great collaborators at Columbia during my graduate years: Stephen Edwards, Ken Ross, Mingoo Seok, João Cerqueira, Tom Repetti, Tim Paine, and Lisa Wu. Thank you for your feedback, help, and guidance. I would also like to thank Luca Carloni for agreeing to be the chair of my defense committee.

I was lucky to participate in many internships during my PhD and to find great collaborators in industry as well: Joel Coburn, Parthasarathy Ranganathan, Alex Ramirez, Tammo Spalink, Ian Kasprzak, March Wachsler, Danner Stodolsky, Asit Mishra, Debbie Marr, and many others. Partha was a very good mentor to me. Sitting in a meeting with him is inspiring; I highly recommend it. Joel has been a great manager, a gracious committee member, and now a friend. I couldn't have asked for more. Alex was an invaluable help in completing the vbench project. During these internships, I was also lucky to meet other amazing intern colleagues. Thank you all for hiking and drinking with me along the west coast. Please come find a job in NYC. Warm weather is grossly overrated anyway.

I want to thank all my Columbia colleagues, and fellow fishbowl students especially. For all the technical questions and conversations we shared, but more importantly for all the happy hours. Social gatherings that I believe were very instrumental for me to not lose my sanity during these long years. I don't think you grow older when sitting at 1020 in good company, so thank you for wasting time and brain cells with me.

Speaking of happy hours, I would also like to thank all my friends back home that I don't get to see as often as I would like. Despite the distance, you always opened your hearts, houses, and liquor cabinets for me. Grazie bimbi.

Of course, all this wouldn't have been possible without my family back home. Although small, they make it up by being incredibly kind and supportive. I couldn't have made it without you. Also thank you to my acquired family, the Desalvo's, who always make me feel welcome and gave me a much needed motivational boost by being the only kid without a PhD. Finally, thank you to my wonderful wife, Giulia. We have been together for a long time and you have always been kind to me. I always felt your support even when I doubted myself the most. This big accomplishment of mine, we built it together. Thank you.

With this big chapter of my life closing, I hope that I will continue to keep challenging myself, keep smiling, and hopefully give back what I have received from you all. Thanks.

Chapter 1

Introduction

Datacenters have been growing constantly in the last decade and are now estimated to consume more than 3% of the global electricity supply with a carbon footprint comparable to the airline industry [Bawden, 2016]. This growth has been fueled by the rise of Big Data and the increased adoption of mobile and IoT devices. These devices are connected to the web and constantly produce data. All this data is stored in datacenters that are expanding in order to keep up with this growing demand. This data deluge is also continuously processed by an ever growing set of applications, from simple ones such as webmail to computationally demanding machine learning pipelines [McAfee and Brynjolfsson, 2012].

At the same time, there has been a historical shift in the computer architecture field with the end of Dennard Scaling [Dennard *et al.*, 1974; Bohr, 2007] and the slowdown of Moore's law [Moore, 2006; Simonite, 2016]. Compared to the past, it has become increasingly difficult to pack more transistors in a given area budget, and – even more troubling – it is not possible to have all of them switching at once. As technology scaling slowly advances, a larger fraction of a chip will have to be idle, creating what has been called dark silicon [Venkatesh *et al.*, 2010; Taylor, 2012; Esmaeilzadeh *et al.*, 2011].

This continuous growth of computational demand in the datacenter, coupled with the end of predictable single core performance improvements between CPU generations, is forcing computer architects to find new solutions to enable more energy efficient computation. Accelerators, i.e. integrated circuits that are specialized for a given application or application domain, are one of these possible solutions. Accelerators are more energy efficient than general purpose CPUs and often provide significant performance improvements. Accelerators also have the benefit of harnessing dark silicon that would have to be left idle or underutilized. Therefore, it is not surprising that accelerators have recently found their way into the datacenter [Jouppi *et al.*, 2017; Caulfield *et al.*, 2016; Putnam, 2017; Fowers *et al.*, 2018; Firestone *et al.*, 2018].

Accelerators implement key computational blocks for an application directly in hardware, thus offering high energy efficiency. Accelerators' designs tend to share common approaches to achieve high energy efficiency: they exploit the application parallelism by spatially mapping computation into separate hardware modules, reduce the overhead of the memory hierarchy by relying instead on explicitly managed buffers (scratchpads) and point to point communication, reduce the overhead of programmability (fetch and decode) by employing coarse grain instructions that operate on large amounts of data.

An unbounded number of accelerators can be created for an application as any circuit that correctly implements it is a valid design. All these designs will differ in terms of performance, i.e. latency and throughput, and cost, i.e. area and power consumed. *Design Space Exploration* is the phase of an accelerator development where different possible designs are evaluated with the objective of finding the best performing one at a given area and power budget. While there are no established methodologies to follow, generally Design Space Exploration goes though a set of phases:

- *Benchmarking*: The application to be accelerated is analyzed with the objective of creating a benchmark to guide subsequent phases. Representative inputs and application parameters are identified in this stage. Computational bottlenecks and memory requirements of different functions are also determined.

- *Architectural*: The most computationally demanding functions indentified during the benchmarking phase are turned into hardware modules and the architecture of the accelerator is defined. Choices that are made at this step include the mapping from functions to hardware modules, the number of modules to include of each type and the connections between them.

- *Microarchitectural*: The implementation of each hardware module is defined. As an example of a microarchitectural design choice, a module that has the lowest throughput might be pipelined in order to increase performance of the whole design.

- *Implementation*: The final accelerator implementation is created. Operations performed at this step include place and route, clock and power gating, voltage/frequency scaling.

Of course the actual development of an accelerator might cycle through these phases multiple times as the implementation is refined. This dissertation will focus on the *Benchmarking* and *Architectural* phases of Design Space Exploration.

Towards the goal of more effective Design Space Exploration, this dissertation supports the thesis that *early design choices – including the level of specialization – are critical for accelerator development* and therefore *require benchmarks reflective of production workloads*.

In this dissertation, we are going to present a case study on profiling the computational requirements of YouTube. This profiling study highlights the diversity of computational requirements that arise when applications have to operate at the warehouse scale[1]. Failing to capture all this complexity will negatively affect the following design space exploration phase by mischaracterizing the predicted impact of different accelerator designs.

Next, we will focus on another application: analytical query processing. This is a good candidate for acceleration as it is a slow evolving application that is at the cornerstone of data mining pipelines. Similarly to video transcoding, we again observe a high variability of the workload depending on the inputs. The computational requirements change with respect to the input queries or input tables. Even within a single query, the computation generally transitions between multiple phases, i.e. filtering, joining, and aggregation, each with different computational requirements.

We will present two studies that highlight how these application properties can be used during design space exploration of an accelerator to derive more desirable designs. Our first study shows how it is possible to specialize the internal communication between functional modules of an accelerator by tailoring connections to the application data dependencies. In

[1]In this dissertation we will use the term warehouse scale to denote any application running in a datacenter on user generated inputs

particular, we notice that some relational operator pairs are more likely than others to have a data dependency. Therefore, we developed an algorithm that can exploit these imbalances to produce an application specific Network-on-Chip (NoC) for the Q100, an analytical query accelerator previously proposed by our group [Wu *et al.*, 2014]. This study shows how NoC specialization can be effective at exploiting patterns in the communication in the same way that accelerators exploit computational patterns of an application.

In our second study, we focus on another aspect of accelerator design: its ideal level of specialization. Our study shows how it is possible to overspecialize an accelerator. For this work, we used again the Q100 as a baseline and compared it against a Coarse Grain Reconfigurable Architecture that we designed. This latter architecture has significant qualitative benefits compared to the Q100: it has an homogeneous architecture that simplifies its implementation and it has the added benefit of programmability. Despite the increased programmability, we find that it shows comparable performance at a given area or power budget. Our analysis explains this counterintuitive result and shows how the more specialized architecture has, on average, a larger fraction of its compute resources idle and a higher cost of communication.

1.1 Summary of Contributions

1.1.1 Benchmarking of Warehouse Scale Applications

A critical aspect of the development of accelerators is benchmarking the application to be accelerated on relevant use cases, and with inputs that are reflective of production scenarios. To this end, we performed a study to characterize the video-as-a-service workload [Lottarini *et al.*, 2018] and released a publicly available benchmark called `vbench`. Unlike prior video processing benchmarks, `vbench`'s videos are algorithmically selected to represent a large commercial corpus of millions of videos. Reflecting the complex infrastructure that processes and hosts these videos, `vbench` includes carefully constructed metrics and reporting rules that reveal nuanced tradeoffs between speed, visual quality, and compression. Moreover, `vbench` can be upgraded to reflect changes in video content and emergence of new applications (e.g., cloud gaming or virtual reality). `vbench` is not just a collection of videos and programs, but a methodology to extract new benchmarks from future video sharing infrastructures.

We demonstrate the importance of video selection with a micro architectural study of cache, branch, and SIMD behavior. `vbench` reveals trends from the commercial corpus that are not visible and sometimes even reversed in other video corpuses. Our experiments with GPUs under `vbench`'s scoring scenarios reveal that context is critical: GPUs are well suited for live-streaming, while for video-on-demand, they shift costs from compute to storage and network. Counterintuitively, they are not viable for popular videos, for which highly compressed, high quality copies are required. We find that popular videos are well-served by the current trajectory of software encoding. This dependency between results and context, i.e. transcoding scenario and input video, that we uncover in our work shows the importance of rigorous benchmarks that reflect production scenarios.

We hope that `vbench` will stimulate a virtuous cycle by encouraging optimization and ensuring fair comparisons of different solutions for this growing market.

1.1.2 Design Space Exploration of Networks on Chip for Accelerators

Our second study shows how performing design space exploration of communication resources can be as effective as optimizing computational ones. We performed a design space exploration of the compute [Wu *et al.*, 2014], and communication [Lottarini *et al.*, 2017] part of an accelerator for analytic query processing. In our ASPLOS 2014 paper [Wu *et al.*, 2014], we presented the Q100, a class of domain-specific database processors that can efficiently handle analytical query processing workloads. This architecture uses coarse grain instructions that operate on streams of data. Each instruction corresponds to a relational algebra operator and has an associated ASIC module – called tile in Q100 terminology – that can execute the operation quickly and energy efficiently. During query execution, relational algebra operators in a query plan are mapped at runtime to Q100 tiles. An instance of a Q100 accelerator can have replicated tiles of any given type in order to exploit parallelism in the query plan. The tiles stream data to each other over an on-chip interconnection network and different query plans can be implemented by changing the routing functions between tiles. The Network on Chip (NoC) effectively constitutes the brain of a Q100 device.

The original Q100 paper [Wu *et al.*, 2014] focused mostly on a holistic analysis of the entire system. It showed how the choice of tile mix is critical to derive an efficient accelerator design given the workload specification (TPC-H [Boncz *et al.*, 2014]). In further work [Lottarini *et al.*, 2017] we optimized the interconnection network as it greatly influences both area and performance of the system. We have shown that various interconnect topologies can trade a factor of 2.5× in performance for 3.3× area. Moreover, standard topologies (e.g., ring or mesh) are not optimal. Significant prior work on network topology specialization augments generic topologies with additional dedicated links. Instead, we presented a network specialization algorithm that first builds a specialized network and then introduces a generic network as a fallback. We find our algorithm produces networks that are 1.24× slower than the highest-performance generic topology considered (a fat tree), and 18% smaller than the least expensive (a double ring). Moreover, our method produces topologies that outperform those produced by other methods [Ogras and Marculescu, 2006] by 1.21× while being 25% smaller.

1.1.3 Tradeoffs Between Levels of Specialization for Accelerators

Datacenters already deploy accelerators targeting the most resource hungry applications, such as Deep Neural Network (DNN) training and inference. These applications are in most cases dominated by a few computational kernels – matrix multiply and activation functions – which are computed repeatedly in this order. This regularity simplifies the design of accelerators for this class of workloads. On the other hand, accelerator design is more complex for irregular applications, i.e. applications that do not have a single dominant computational kernel, have significant control flow divergence and complex data dependencies between computational kernels. For this class of applications, we have found that specialization might not necessarily correlate with higher performance and/or energy efficiency. We are going to examine analytical query processing and compare the Q100 class of accelerators to a homogeneous Coarse Grain Reconfigurable Architecture. This architecture is composed of a mesh of processing elements. Each processing element has equal computational capabilities and can be programmed to execute any relational algebra operator supported by the different Q100 tiles. Despite the increased programmability of the new architecture, we find that it shows comparable performance at a given area or power budget. Our analysis shows how the more specialized accelerator can be less efficient than a programmable non Von-Neumann architecture. Resource usage explains this counter-intuitive result, as a well-programmed homogeneous array of resources is able to more effectively adapt to the time varying computational requirements of analytical queries. On the other hand, the Q100 is composed by a fixed mix of tiles, of which only a small fraction are active on average. Furthermore, since each processing element in the programmable architecture has equal computational abilities, data tends to flow between neighbors reducing the cost of communication. When considered together, the higher silicon utilization and lower cost of communication, explain how the programmable architecture can outperform the Q100.

These results suggests that a balanced accelerator research portfolio must include alternative programmable architectures, and their software stacks.

1.2 Dissertation Outline

Chapter 2 includes the motivation of our work and other possible alternative approaches. Section 2.1 contains a study on the performance trends of commercial CPUs over time, explaining how the slowdown in the performance increase of modern CPUs is due to the end of Dennard scaling. This slowdown motivated industrial and academic research on accelerators – the focus of this dissertation – of which we present a survey in Section 2.2.

Chapter 3 presents `vbench` the first benchmark for large video sharing infrastructures that we developed in collaboration with YouTube. Some background information about video transcoding, including the metrics that we used for our benchmark scoring functions, is presented in Section 3.1. Section 3.4 presents the actual benchmark and the methodology used to synthesize it from YouTube production transcoding pipelines. Finally Section 3.5 and Section 3.6 present insights gained by using `vbench`.

Chapter 4 presents our work on application specific Network-On-Chip for accelerators. First, the Q100 class of accelerators for analytical query processing is presented in Section 4.1. Then, our algorithmic approach to derive application specific NoCs is presented in Section 4.3 and results of its application to the Q100 is presented in Section 4.5.

Chapter 5 presents our work on the specialization tradeoffs for the acceleration of irregular applications. We use the Q100 as our baseline and the reader can refer back to Section 4.1 for background information on this architecture. We analyze the drawbacks of the Q100 architecture in Section 5.1 and then introduce the programmable homogeneous architecture that we use as a comparison point in Section 5.2. The results of this comparison – including an analysis of the two architectures salient differences – are shown in Section 5.5.

Finally, Chapter 6 summarizes the contributions of this dissertation and proposes selected future work.

Chapter 2

Background

Gordon Moore predicted in 1975 that the amount of transistors that could fit in a given area would double every two years [Moore, 2006]. His prediction was correct until recently [Simonite, 2016] and this exponential growth fueled the digital revolution of the 20th Century. Moore's law is now slowing, resulting in increasing time between the commercial release of newer chips at smaller technology nodes.

More importantly, the last decade saw the end of Dennard scaling [Dennard *et al.*, 1974] and general purpose processors hit the so called "Power wall." Current CPU designs have a significant fraction of their area underutilized or idle in order to keep dissipated power at bay. This limitation severely stunted performance growth of general purpose CPUs (Figure 2.1) with single core performance plateauing after more than 40 years of exponential growth.

Accelerators are a promising way to contend with the end of Dennard scaling and the slowdown of Moore's law. In this chapter, we are going to introduce the reasons behind the end of Dennard scaling that motivated the development of accelerators (Section 2.1). Then, a survey of deployed and proposed accelerators is presented in Section 2.2. Two classes of accelerators that are particularly relevant in this dissertation are presented here in greater detail: accelerators targeting query processing (Section 2.2.1) as well as Coarse Grain Reconfigurable Architectures (Section 2.2.2).

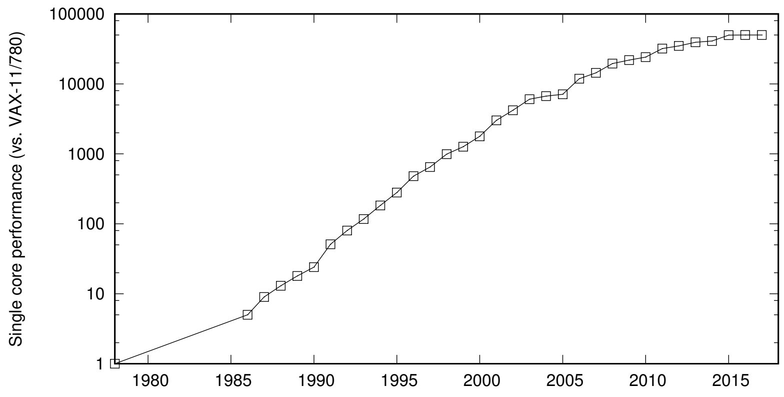


Figure 2.1: Single core performance has been growing exponentially in the last 40 years (notice the logarithmic y axis). This growth started plateauing in the last decade due to the slowdown of Moore's law and the end of Dennard scaling. Data from [Hennessy and Patterson, 2017]

2.1 The End of Dennard Scaling

Robert N. Dennard observed in the 1970s that it should be possible to linearly scale all figures of merit of a transistor, i.e. voltage and capacitance, with respect to its size [Dennard *et al.*, 1974]. His observation proved to be true for decades. Thanks to improved fabrication processes, circuits could pack not only more, but also faster transistors in the same area (hence at the same materials' cost) while keeping a constant power density. Over time, the same **computation became cheaper and cheaper** to perform.

However, the last decade saw the end of Dennard scaling [Borkar and Chien, 2011; Bohr, 2007]. While the size of transistors keeps shrinking (albeit at a slower pace than in previous years), the voltage at which they operate cannot. More precisely, Dennard's prediction was based on the assumption that it would always be possible to scale down the threshold voltage of a MOSFET proportionally to its reduction in size. This is no longer true as transistors have reached feature sizes of a few atoms and subthreshold leakage has a significant enough effect that it prevents lowering threshold voltage any further [Bohr, 2007].

This implies that the power consumed by a circuit operating at full frequency has become too large for practical cooling solutions. Effectively, this constitutes a limit (the so called

"Power Wall") on the increase of circuits performance. Large portions of a chip have to be powered down at any time in order to keep power dissipation low enough for practical cooling solutions.

Dynamic power consumed in a transistor is a function of the switching activity (α), capacitance (C), frequency (F), and voltage (V), that the chip is operating at:

$$P = \alpha \times C \times F \times V^2$$

The switching activity α corresponds to the probability of the given transistor to switch at each clock cycle. Notice that the power has a quadratic relationship with the voltage. Let's assume that it is possible to reduce the minimum feature size by a factor S. Table Table 2.1 provides a quick comparison of the implications of technology scaling in the past and in our current, post Dennard scaling regime.

Scenario	**Dennard Scaling Regime**	**Post Dennard Scaling Regime**
Voltage	1/S	1
Capacitance	1/S	1/S
Maximum Frequency	S	S
Transistors in fixed area budget	S^2	S^2
Power consumed at full speed	1	S^2
Fraction of chip switching at full speed	1	$1/S^2$

Table 2.1: Comparison of scaling for circuits in the Dennard scaling (past) and post Dennard scaling regime (now). Notice how the power dissipated can grow quadratically in the post Dennard scaling regime.

If we analyze the performance of commercial CPUs in the last two decades we can clearly observe the implications of the end of Dennard scaling. Figure 2.2 shows that around 2005 we reached the limit of power that could be reasonably dissipated on a socket. Notice also that CPU architects slowly increased the power dissipation of commercial CPUs even in the Dennard scaling era in search for higher single core performance.

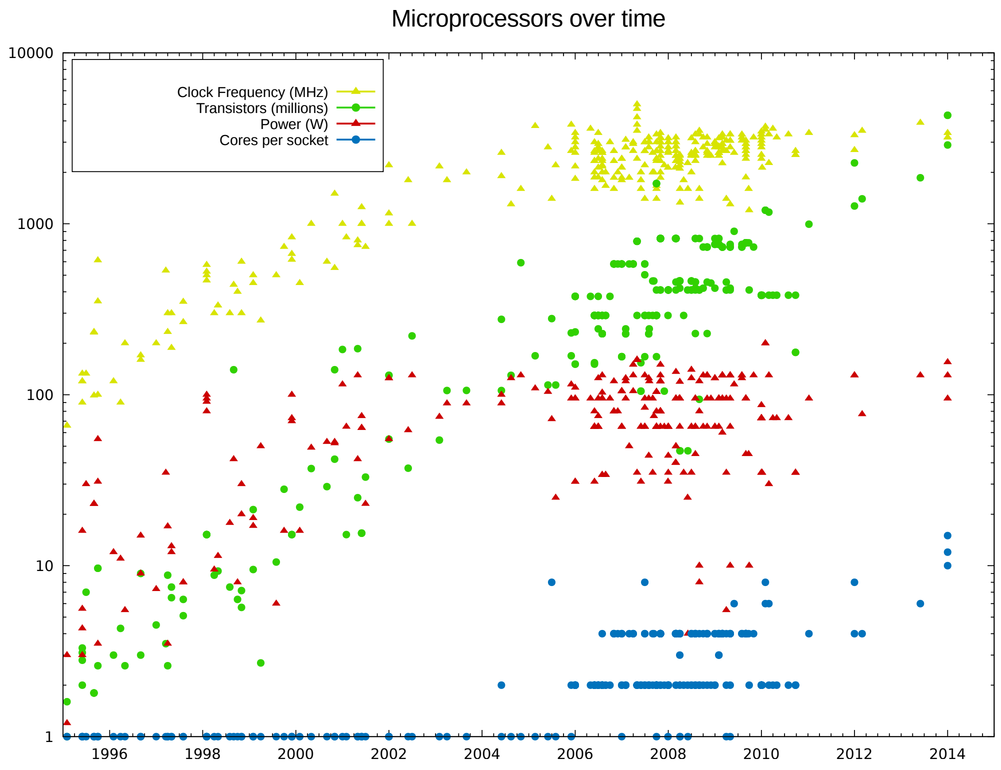


Figure 2.2: Around 2005 CPU design hit the "Power Wall" as the limit of power that could be dissipated on a single socket was reached. As a consequence, the frequency of CPUs also plateaued around that time. The increasing number of transistors available are now spent on simpler and more energy efficient CPU cores. Data from [Danowitz *et al.*, 2012]

With power dissipation imposing a limit on CPU design, operating frequency increases had to stop to reduce power consumption. As a stopgap measure, CPU architects reverted back to simpler CPU designs. While these provide lower single core performance, they are more energy efficient and multiple instances can be packed in a single socket creating the Chip Multiprocessors (CMPs) that are ubiquitous today.

CMPs have clear limitations that were soon documented by the computer architecture community [Taylor, 2012; Esmaeilzadeh *et al.*, 2011]. The most important one is Ahmdal's law. Not all workloads can be sped up by operating in parallel on multiple CPUs since the portion of an application that has to run sequentially will always limit the possible speedups attainable.

Accelerators are a longer term solution to this problem. Accelerators implement algorithms in hardware in order to attain high energy efficiency and will be the focus of this dissertation. We present a survey of proposed and deployed accelerators in the next section.

2.2 Accelerators

It should be clear from the previous section that in our post Dennard scaling regime, high energy efficiency has become key. In order to understand how accelerators achieve high energy efficiency, we have to first analyze software running on general purpose CPUs. Only a small fraction of the power dissipated on a general purpose CPU goes towards performing the operations needed by the algorithm [Horowitz, 2014]. The large majority of power goes towards supporting programmability and data movement. As an example, consider that the energy spent to perform a 32bit integer add on a modern CPU is 0.1pJ. That is only a small fraction of the 70pJ that are necessary, on average, to execute an instruction. Most of this power goes towards fetching instructions (25pJ), accessing the register file (6pJ), decoding, control, etc. Speculation in modern out of order CPUs also adds to the power budget, sometimes without providing any performance benefit (consider a mispeculated branch). Finally, energy spent performing operations pales in comparison to the energy necessary to fetch data from DRAM, which can be orders of magnitude greater. These sources of inefficiency in CPUs have been widely studied [Hameed *et al.*, 2010; Horowitz, 2014]; this high energy cost per instruction comes from many years of CPU development that aimed solely at obtaining high single core performance.

In contrast, we can define accelerators as any architecture that tries instead to minimize any energy spent in operations that are not strictly necessary to perform the computation. Accelerators generally have very limited programmability, hence any penalty associated in a CPU to fetching and decoding instructions is almost completely eliminated in most accelerator designs. Accelerators do not rely on a memory hierarchy, but instead use explicitly managed memories, called scratchpads, to maximize the number of operations performed on each byte that is fetched. This amortizes the high cost of data movement and avoids overheads of general cache hierarchies such as conflict misses. Furthermore, accelerators usually do not employ any form of speculation that could result in unnecessary computation being performed. Finally, accelerators usually exploit the inherent parallelism of applications by replicating computational blocks and therefore increase performance compared to a software implementation. Since accelerators tend to also have more predictable running times than software, this additional performance manifests itself as both increased throughput and reduced latency.

Accelerators have already been deployed in datacenters. Currently, both Google [Jouppi *et al.*, 2017; Dean *et al.*, 2018] and Microsoft [Fowers *et al.*, 2018] deploy accelerators in their datacenters to accelerate machine learning workloads. These accelerators are currently replacing GPUs as the main computational substrate for machine learning. FPGAs are also used in Microsoft's datacenters to offload software defined networking tasks [Firestone *et al.*, 2018].

In academia, accelerators have been proposed for the most disparate domains, including genome sequencing [Fujiki *et al.*, 2018], graph analytics [Ham *et al.*, 2016], key-value stores [Lim *et al.*, 2013], and, of course, machine learning [Reagen *et al.*, 2016; Chen *et al.*, 2016b]. While most of these works target an entire application or an application domain, there have been proposals for accelerating functions that are common across applications such as memory allocation [Kanev *et al.*, 2017] or garbage collection [Maas *et al.*, 2018]. These cross application overheads take a big toll on datacenter applications. Recent work on profiling applications running in Google datacenter [Kanev *et al.*, 2015] found that almost 30% of the clock cycles are spent in similar operations that the authors named as the "datacenter tax".

Chapter 4 and Chapter 5 will focus on accelerators for analytical query processing. We present proposals in more detail for this domain in Section 2.2.1. Chapter 5 also compares a standard accelerator design against a Coarse Grain Reconfigurable Array. We present these architectures in Section 2.2.2.

2.2.1 Specialized Hardware for Analytical Query Processing

Analytic query processing is a good candidate for acceleration since it is a stable workload, it is widely used, and operates on structured data.

Furthermore, analytical query processing benefits from having a standard language (SQL), intermediate representation for the computation (relational algebra) and a benchmark (TPC-H) that is an industry standard [Boncz *et al.*, 2014].

Kung et al. first suggested using systolic arrays[1] to accelerate database workloads [Kung and Lehman, 1980] in 1980. However, high performance improvement of CPUs year after year made the development of ASICs for this kind of computation unappealing [Boral and DeWitt, 1983].

The end of Dennard scaling spurred the more recent interest in this topic. Key computational kernels such as partitioning [Wu *et al.*, 2013], nested loop joins [Teubner and Mueller, 2011; Cao *et al.*, 2017], and hash-joins [Kocberber *et al.*, 2013] have been evaluated for acceleration as ASICs with high expected benefits. The Q100 that will be the focus of our later analysis in Chapter 4 aims instead to accelerate full analytical queries. A similar architecture to the Q100 has also been proposed for FPGAs [Chung *et al.*, 2013] targeting the LINQ query language. Other notable approaches for the use of specialized hardware in analytic query processing include modifications to the storage subsystem to increase the DBMS performance [Jun *et al.*, 2015] as well as near storage compute [Woods *et al.*, 2014].

Industry too has examined database acceleration. Oracle, for example, proposed a many core architecture [Agrawal *et al.*, 2017]. It differs from our design as its cores have a standard Von-Neumann architecture and data is received from a DMA engine rather than passed in a dataflow manner from one PE to the other. Baidu recently showed results from an FPGA database accelerator that closely resembles the Q100 [Ouyang *et al.*, 2016].

2.2.2 Coarse Grain Reconfigurable Architectures

Course Grain Reconfigurable Architectures/Arrays (CGRAs) 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 [Sutton *et al.*, 1998; Mirsky and DeHon, 1996; Goldstein *et al.*, 2000; Taylor *et al.*, 2002; Sankaralingam *et al.*, 2003; Swanson *et al.*, 2007;

[1]Systolic arrays are a predecessor of Coarse Grain Reconfigurable Architectures which will be the focus of the next section

Govindaraju *et al.*, 2012; Yu *et al.*, 2008; Bohnenstiehl *et al.*, 2017; Parashar *et al.*, 2013; Prabhakar *et al.*, 2017; Nowatzki *et al.*, 2017; Repetti *et al.*, 2017]. While the term CGRA can encompass a large selection of different architectural proposals, these architectures have a couple of defining characteristics: a large number of relatively simple CPUs – often called processing elements (PEs) – that operate at the word level, and communicate over a mesh network (Figure 2.3).

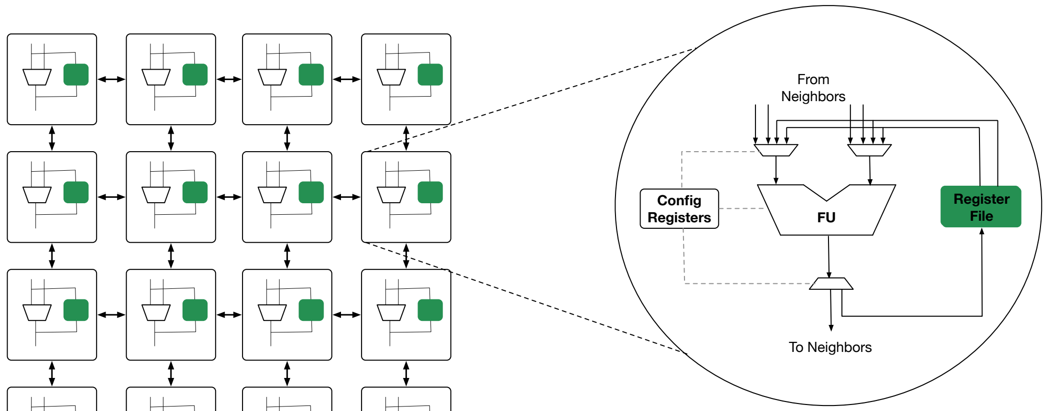


Figure 2.3: Architecture of a generic CGRA.

CGRAs constitute a middle ground between GPUs and FPGAs. Their architecture is more efficient than GPUs in the presence of control flow divergence and, by operating at the word level, CGRAs also avoid the cost of bit level programmability that FPGAs offer, but is often time unnecessary in warehouse scale applications [Taylor, 2012; Kuon and Rose, 2006; Falsafi *et al.*, 2017]. As an example, recent work has shown how CGRAs can outperform SIMD architectures as well as an FPGA implementation for computer vision algorithms [Vasilyev *et al.*, 2016].

CGRA architectures differ from one another mainly in their means of scheduling operations on the various PEs. Some architectures rely on all operations and data movement being scheduled globally on a cycle-by-cycle basis [Sutton *et al.*, 1998; Mirsky and DeHon, 1996; Goldstein *et al.*, 2000]. Others map circuit-switched dataflow networks onto an array of PEs with local control consisting only of if-then-else predication [Sankaralingam *et al.*, 2003; Swanson *et al.*, 2007; Govindaraju *et al.*, 2012]. Finally, some designs give PEs complete local autonomy to execute sequential routines covering their designated portion of a spatial program [Taylor *et al.*, 2002; Yu *et al.*, 2008; Bohnenstiehl *et al.*, 2017; Parashar *et al.*, 2013].

Stream-dataflow [Nowatzki *et al.*, 2017] and Triggered Instructions [Parashar *et al.*, 2013] are the closest proposals to the homogeneous architecture we propose in Section 5.2. Stream-dataflow operates explicitly on memory streams fed into and read from the reconfigurable fabric by external stream readers and stream writers [Nowatzki *et al.*, 2017]. Our homogeneous PEs use triggered-instruction-like operand-availability as a criteria for instruction scheduling [Parashar *et al.*, 2013] and can thus tolerate arbitrary data latencies. The tags that are added to each data channel perform a similar function to our independent control plane. Neither of these architectures is optimized for database applications in the same way as our design.

Chapter 3

Benchmarking Video Transcoding at Scale

Video sharing represents a growing fraction of internet traffic. For example, in the November 2016 Facebook earnings presentation, Mark Zuckerberg described Facebook's evolution into a "video first" company [facebook, 2016]. The 2016 Sandvine Global Internet Phenomena report [Sandvine Intelligent Broadband Networks, 2016] places audio and video at 71% of evening traffic in North America and projects that figure will grow to 80% by 2020. Video processing plays a pivotal role in virtual and augmented reality (Oculus Rift, HoloLens), video surveillance (Nest), cloud gaming (GeForce Now, PlayStation Now), and other emerging applications.

To keep up with growing usage, video on demand providers such as Netflix, YouTube, and Facebook maintain large video serving infrastructures. All these services perform a large number of transcoding operations [Xin *et al.*, 2005], i.e., decoding a compressed video into raw frames and re-encoding it in a new compressed format. Each uploaded video is transcoded at least once before it is sent to viewers. This ensures that videos that are malformed are not distributed. Even more importantly, each upload must be converted to a range of resolutions, formats, and bitrates to suit varied viewer capabilities, i.e., screen resolution, codecs supported, and available network bandwidth. In every transcoding operation, there is a trade-off between compressed video size, fidelity to the original video,

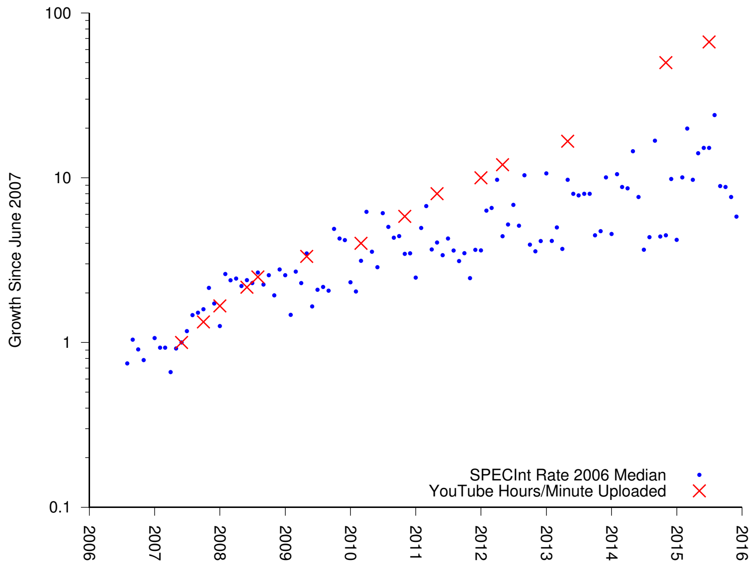


Figure 3.1: Many hours of video are uploaded to YouTube every minute [Insights, 2015]. The uploads are growing more rapidly than CPU performance (as measured on SPECRate2006), which creates a growing burden on video sharing infrastructures.

and transcoding time. For example, reducing video size may reduce visual quality, but encourages smooth playback, thus potentially improving the overall quality of experience.

Within a transcode operation, the decoding step, which converts a compressed video stream into a sequence of frames to be displayed, is deterministic and relatively fast. In contrast, the encoding step has to make many decisions that can not be exhaustively explored, so encoders perform a heuristic search over this decision space. Increasing the space searched, also known as the *effort level*, increases the likelihood of finding a better transcode, i.e., a compressed video with less distortion, or lower bitrate. The reader can find a more detailed explanation of transcoding workloads in Section 3.1.

Transcoding is ripe for optimization. As Figure 3.1 depicts, demand is outstripping CPU performance, and within Google, the cycles spent transcoding have grown by 10x in the last two years. Research is needed in order to advance areas like hardware video transcoding, evaluation of new video codecs, and related technologies. However, *there is no well-defined way to compare transcoding solutions*. In the video processing community, encoders are evaluated in terms of visual quality and bitrate. Large scale studies [Cock *et al.*, 2016] show increases in compression rate without hurting video quality. Computation time, however, is not typically measured and thus also increases: as new codecs introduce new knobs and parameters, the search space grows. Our case study in Section 3.6.2 demonstrates this effect. In the architecture community, two widely used benchmark suites, SPEC [Henning, 2006] and PARSEC [Bienia, 2011], include some video encoding use cases. However, the video content and settings are not representative of a video sharing infrastructure.

To establish some common ground, we developed `vbench`, a video transcoding benchmark that reflects the transcode demands of a video sharing service such as YouTube. YouTube receives videos in thousands of combinations of resolution, framerate, and complexity (entropy). `vbench` uses clustering techniques to select 15 videos of 5 seconds each. This is a small enough number to allow detailed RTL or microarchitecture simulations, but wide enough to cover a significant cross section of the corpus of millions of videos. `vbench` also establishes a set of reference transcode operations against which other transcoder proposals can be compared. These operations are comparable with operations that are performed at providers like YouTube. This ensures a consistent and appropriate baseline. Finally, `vbench` includes five comparison metrics derived from real-world transcoding scenarios. These metrics guide meaningful improvements by enforcing constraints associated with video transcoding at scale.

We demonstrate the value of `vbench` with four use cases. First, we show how `vbench` choice of videos highlights microarchitectural trends and then derive qualitative conclusions from this performance study. Second, we quantify the limits of SIMD vectorization for transcoding: consistent with other studies [Hameed *et al.*, 2010], we find that SIMD instructions can provide limited improvements. Next, we evaluate current GPU support for video transcoding, finding non-intuitive tradeoffs in their behavior. While GPUs are a clear win for live transcoding tasks, they sacrifice compression and quality for video archival. Lastly, we find that while GPUs today cannot meet the strict quality and compression targets for popular videos, newer and more complex software encoders can. Collectively, these studies demonstrate the relevance of our benchmark and the importance of having a curated set of videos, meaningful baselines, and encoding scenarios to evaluate new transcoding solutions. Moreover, by showing the possible pitfalls of using non-curated video sets and the significant differences in computational requirements for different production transcoding pipelines, our results presented here support our thesis that design space exploration requires benchmarks reflective of production workloads. In the remainder of this chapter we are going to introduce the video transcoding workload (Section 3.1) as well as the video sharing infrastructures that employ many video transcoding pipelines that will be the subject of our analysis (Section 3.2). Related work in video benchmarking is presented in Section 3.3. Section 3.4 describes `vbench` and the methodology we used to construct it. Section 3.5 and Section 3.6 present the main insights we gained using `vbench`. Finally Section 3.7 concludes the chapter by summarizing the contributions of this work and how they support our thesis.

3.1 Video Transcoding Background

This section provides some background on video transcoding techniques, how they are evaluated, and the video sharing infrastructures where they play a crucial role.

To understand the importance of transcoding, consider that a raw Full-HD frame (1920×1080 pixels) is roughly 3MB. Streaming uncompressed video at 30 frames/second would require 90 MB/s, or 700 Mb/s, exceeding the capabilities of most home broadband installations.

Since streaming raw video is not practical, it must always be compressed. A video *transcoder* is a device or program that decodes a compressed input video into a raw, uncompressed format and then re-encodes it in a new compressed format. It is equivalent to a decoder and encoder chained together. While video decoding simply follows the interpretation rules for the bitstream of the video format, video encoding has a number of degrees of freedom to decide how the raw video should be compressed. Video encoding formats, like H.264/AVC [Wiegand *et al.*, 2003], H.265/HEVC [Sullivan *et al.*, 2012], or VP9 [Mukherjee *et al.*, 2013], are usually referred to as *codecs*.

3.1.1 Video Transcoding

Video encoders exploit properties of human perception as well as spatial and temporal redundancy in the video content. Humans perceive changes in luminosity more than changes in color, therefore video processing is performed on the YUV color space, rather than in RGB. YUV separates luminosity signal (luma) from color information (chroma) allowing encoders to dedicate more bits for the luma plane than the chroma plane (a process called chroma subsampling). They also rely on the fact that blocks of pixels in a frame are usually similar to other blocks of pixels in previous (and future) frames. Encoders take advantage of this similarity by expressing part of a frame as a function of blocks in other *reference frames*.

Video encoders generally adhere to the following template: First, video frames are decomposed in square blocks of pixels called *macroblocks*. For each macroblock, the encoder searches temporally neighboring frames for similar macroblocks (*motion estimation*). This initial *motion estimation* is usually the most computationally onerous step [Hameed *et al.*, 2010]. Once a suitable reference block is found, the encoder computes the difference (the *residual block*) and stores only the relative location (the *motion vector*). Residual blocks are then encoded like a regular image [Rabbani and Jones, 1991]: A *discrete cosine transform* (DCT) is used to convert blocks of pixels to the 2D spatial frequency domain. Then the matrix of coefficients is *quantized*, i.e. divided point-wise by another matrix (the *quantization matrix*) to introduce zeroes[1]. Quantization zeroes out the high frequency components (quick pixel transitions) which are less noticeable to the viewer. Quantization is the only lossy part of the process. The more zeroes introduced this way, the more effective the final compression step, in which each frame is losslessly compressed via a sequence of *run length encoding* and *entropy encoding*. Examples for the latter are *e.g.* Context Adaptive Binary Arithmetic Coding (CABAC) or Context Adaptive Variable Length Coding (CAVLC) [Marpe *et al.*, 2003].

To allow playback from arbitrary points in the video stream, and to allow recovery after a transmission error, index frames (I-frames, also called *keyframes*) have to be inserted at regular intervals. I-frames do not contain motion vectors, therefore it is possible to decode these independently of other frames.

New codecs introduce new compression tools and algorithms, like the H.264 *deblocking filter*, which removes artifacts that can appear at the boundaries between macroblocks.

[1]Potentially the entire residual block can be discarded if equal to zero after quantization.

Denoising is another optional operation that can be applied to increase video compressability by reducing high frequency components [Kokaram *et al.*, 2012].

3.1.2 Encoding Effort

Video encoding requires the user to specify a target quality. If the user specifies a constant rate factor (CRF), the encoder will try to sustain the same quality level for all video frames, using as many bits as necessary. Alternatively, to make the video size predictable, the user can specify a target bitrate (bits per second); The encoder will try to fit the video in the allocated space, but may sacrifice quality to do so.

When encoding to a target bitrate, 2-pass encoding can optimize the allocation of bits to the more complex parts of the video. On the first pass, the encoder records how complex each frame is and uses that information in the second pass to budget fewer bits for simple frames, and more for complex frames.

The Rate Distortion Optimizer (RDO) decides how to gracefully degrade visual quality in order to meet the target bitrate. A sample RDO decision would be the post-DCT quantization strength. More complex decisions include how to decompose the frames into macroblocks or whether to perform sub-pixel motion estimation.

The difficulty of the RDO's job is input dependent. Videos with static images, such as slideshows or animations, are easily compressed since motion vectors describe most of the frames with precision. On the other hand, videos with high motion and frequent scene changes will require more time for motion search, and other optimizations to fit the frames in the allowed bitrate.

It is possible to specify an encoder effort level that affects the RDO decisions. RDO decisions at each stage of encoding entail difficult-to-predict tradeoffs between quality and bitrate. As a consequence, the whole encoding process resembles a heuristic search. Performing more computation, i.e. covering more combinations in the encoding space, ensures that better transcodes are found. The effort level restricts the parameters (motion search range, number of reference frames, etc.) used in search for a better encoding. Higher effort will achieve higher quality at the same bitrate, at the expense of longer encoding time.

3.1.3 Transcoding Metrics

Video transcoding must be evaluated in three dimensions: visual quality, video size, and transcoding speed.

Visual quality is measured by comparing the original uncompressed frames with the same frames in the encoded version. *Peak signal-to-noise ratio* (PSNR) captures the ratio between the maximum error per pixel and the actual error per pixel, so larger values indicate higher quality. Given an initial raw frame F and its transcoded version T, both of $m \times n$ pixels, PSNR is obtained by computing the mean square error (MSE):

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(F(i,j) - T(i,j))^2$$

The MSE is then compared against the maximum pixel value of the frame, typically 255 for standard 8 bit pixels.

$$PSNR = 10\log_{10}\left(\frac{255}{\sqrt{MSE}}\right)$$

This process can be repeated for all planes, luma (Y) and chroma (Cb,Cr) of all frames and averaged to compute the average YCbCr PSNR. We use average YCbCr PSNR in the remainder of the paper as a measure for transcoding quality.

There are several alternative "perceptual quality" metrics such as Structural Similarity (SSIM [Wang *et al.*, 2004]), and those recently proposed by Netflix [Li *et al.*, 2016] and Google [Chen *et al.*, 2016a]. These metrics try to capture the specifics of human visual perception into an analytic method. They all assume that the original video is uncompressed. However, YouTube uploads generally arrive already encoded and thus potentially distorted. Furthermore, there is no consensus in the video processing community as to which one of these metrics works best. Therefore, we rely on the "objective" PSNR for the rest of this work.

Video size is usually measured by *bitrate*, the number of bits per second of video. While actual video file size depends on the length of the video, bitrate is a video-length-normalized metric. Decreasing the bitrate of a video stream decreases the likelihood of re-buffering

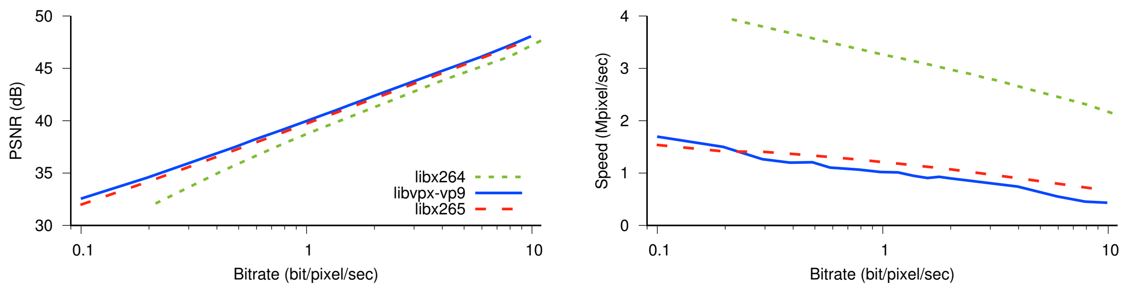


Figure 3.2: Video transcoding is usually compared on video quality vs. video bitrate, but that leaves out a critical third dimension: the transcoding speed.

events, i.e. video data packets not delivered on time for playback. To compare videos at different resolutions, we report bitrate normalized by the number of pixels in each frame (bits per pixel per second).

Transcoding speed, like bitrate, is normalized against the length of the video, and the resolution of each frame. We multiply the number of frames transcoded in a second by the number of pixels in a frame and report the number of pixels transcoded per second.

3.1.4 Evaluating a Transcoder

Since video quality is clearly related to the bitrate made available to the video encoder, the video community compares video transcoders using *PSNR curves* that plot video quality as a function of the video bitrate. Figure 3.2 (left) shows PSNR curves for three different software encoders on one HD video[2].

The curves show that libvpx-vp9 achieves slightly better quality than libx265, and both achieve substantial improvements over libx264 for all target bitrates. That would indicate that libvpx-vp9 is a superior transcoder, since it can always provide better quality at the same video size, or smaller videos at the same quality.

However, when transcoding speed is factored in Figure 3.2 (right), we observe that the advantage of libvpx-vp9 over libx265 corresponds to a decrease in transcoding speed, and that both of them require 3-4x more computation than libx264. It is no longer obvious

[2]The first 1000 frames of Big Buck Bunny [Blender Foundation, 2002], used in SPEC 2017.

which one is the best encoder.

The answer depends on the use case. For large video sharing infrastructures a fast transcode is needed when streaming a live event, so it is necessary to trade bitrate and/or quality to ensure that streaming is fluid. Conversely, when a video is expected to be played many times, it is worth using an advanced encoder, since the cost of producing a smaller video at equivalent perceptual quality is amortized, and the bitrate savings are multiplied, across the many playbacks. A video sharing infrastructure (Section 3.2) is designed to efficiently manage all these decisions.

3.2 Video Sharing Service Architecture

A video streaming service such as Netflix, YouTube, or Facebook allows users to view, rate, upload, and comment on videos. They serve media to a wide variety of devices, from web browsers to mobile applications, with media content ranging from movies, television programs, music videos, video games, and documentaries, to animations, slideshows, and screen capture tutorials.

These services incur three primary costs: storage, network, and compute. The storage cost is proportional to the size of the corpus of videos stored in a central repository, including duplicates in various resolutions and formats, as well as replication across a Content Distribution Network (CDN) for faster service [Buyya *et al.*, 2008; Rafetseder *et al.*, 2011]. The network cost is determined by the egress traffic from the central repository and/or CDN to users and, to a lesser extent, by the ingress traffic of new uploads. The compute cost is incurred each time a video is transcoded.

To optimize these costs, video streaming services make multiple video transcoding passes on each video, as illustrated in Figure 3.3. Videos are uploaded in a wide variety of combinations of codec, container, color space, resolution, frame rate, etc. [Engineering and Blog, 2016]. To apply a uniform process, all originals are first transcoded to an universal format that functions as an intermediate representation for the system.

From there, videos are transcoded to a wide variety of formats and resolutions to suit the capabilities of the network and the different client platforms. Depending on whether

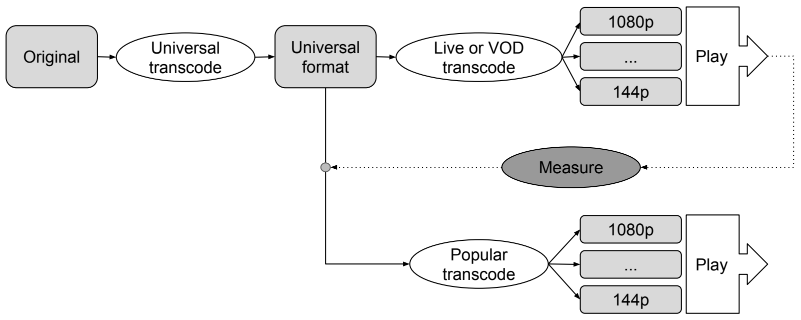


Figure 3.3: Video transcoding passes in a video sharing infrastructure. Higher effort is invested in popular videos watched many times.

the video is being forwarded directly to clients, i.e. live streaming, or transcoded offline and stored to be viewed later, i.e. video on demand or VOD, this encoding can be single pass for low latency, or two pass for higher quality and smaller size. Newly uploaded videos must be available for playback as soon as possible, especially for live streaming, so the latency of these transcoding steps must be bounded.

Video popularity follows a power law distribution with exponential cutoff [Cha *et al.*, 2009]: most of the watch time concentrates in a few popular videos, while there is a long tail of rarely watched videos. It would be wasteful to invest much compute effort on the long tail, but when a video is observed to be popular, services will spend the extra compute. Those videos are transcoded a second time at higher effort levels to produce high quality compressed replicas that optimize user experience, storage and network costs. The extra compute time is amortized across many playbacks of the video, while the savings are multiplied across playbacks.

3.3 Related Work

While benchmark suites have been developed for various classes of datacenter workloads, such as personal assistants [Hauswald *et al.*, 2015], web applications [Zhu *et al.*, 2015], big data analytics [Wang *et al.*, 2014] as well as datacenter workloads at large [Ferdman *et al.*, 2012], there is no suite specifically targeting video transcoding. When transcoding does appear in a benchmark, the videos are not necessarily representative, and sometimes the specific operations to be performed are not specified at all.

Several popular CPU benchmarks contain some video transcoding workload. For example, SPEC 2006 [Henning, 2006] includes the H.264 reference encoder with two low-resolution videos. SPEC 2017 uses the libx264 encoder and two segments of a HD video. PARSEC [Bienia, 2011] includes a pthread parallelized version of the libx264 encoder and one input video. Since video transcoding microarchitectural profile is highly dependent on the type of inputs, these are clearly not enough input sequences to provide good coverage.

Netflix released a dataset of 9 videos from an internal suite of 34 video clips from popular TV and movies from their catalog. This curated data set has been used in a study proposing a perceptual quality metric, as opposed to signal fidelity metrics like PSNR [Li *et al.*, 2016]. These sequences are collected using a qualitative metric ("a wide range of high level features: animation, indoor/outdoor, camera motion, $\cdots$") while videos in vbench are selected according to their entropy, a quantitative metric which we formally define in Section 3.1. The Alliance for Open Media is using Derf's HD video collection from Xiph.org [Xiph.org, 2016] for the development of the AV1 codec [Foundation, 2017]. The Derf collection contains 41 videos from 480p to 4K resolution but the rationale for inclusion is not clear. Both Xiph and Netflix workloads are not intended for performance benchmarking and there is no indication on how to use them.

HD-VideoBench [Alvarez *et al.*, 2007] is a notable previous attempt at benchmarking video transcoding that however lacks the diversity in both input sequences (only 4 videos obtained from the MPEG-Test Sequences archive) and scenarios (single pass constant quality encode only) that characterize a video sharing infrastructure. All their video sequences are included in the Xiph.org collection.

Prior architectural work in video transcoding does not use rigorously selected videos [Magaki *et al.*, 2016; Hameed *et al.*, 2010; Qadeer *et al.*, 2013], or compares against unoptimized encoders [Henning, 2006; Hameed *et al.*, 2010] that underperform compared to state of the art software solutions. As a consequence, it is difficult to translate their insights for video sharing infrastructures, since the implications on the quality of user experience, storage, and network costs can not be predicted from the reported results. As an example, Fouladi et al. recently implemented a system [Fouladi *et al.*, 2017] to perform low latency transcodes using AWS Lambda. Their evaluation was performed using two movies from the Blender Foundation [Blender Foundation, 2002] (Sintel and Tears of Steel) and no rationale for this choice is stated. Zhang et al. investigate how to reduce video decoding energy consumption in mobile devices [Zhang *et al.*, 2017]. A mix of video sequences is used for the evaluation with no explicit rationale for inclusions. `vbench` would be more representative of the content that mobile devices receive from video sharing infrastructures.

There has been recent work documenting how large video sharing infrastructures operate [Huang *et al.*, 2017] or optimize for popular videos [Tang *et al.*, 2017]. Results obtained using `vbench` should also apply to these systems.

Magaki et al. explore the possibility of introducing ASICs into the datacenter to process large scale workloads with a lower Total Cost of Ownership (TCO) [Magaki *et al.*, 2016]. Their analysis names video transcoding as a possible candidate. However, before building an ASIC, a performance analysis of the workload to accelerate is paramount.

Overall, we find that the state of the art is not conducive to controlled comparisons between transcoding solutions. This lack of data and common benchmarks hampers innovation in this critical area.

3.4 `vbench`: a Video Transcoding Benchmark

In this section, we describe `vbench`, our video transcoding benchmark consisting of input videos (Section 3.4.1), scoring functions (Section 3.4.2), and reporting rules (Section 3.4.3). All of the videos, reference data, and scripts are publicly available on the `vbench` website *http://vbench.net.*

3.4.1 Video Selection

The input videos must achieve a complex trade-off between representativeness and coverage. They must be representative so that the results match what is observed on a production system, but provide coverage to expose trends and not ignore corner cases that may become increasingly important in the future. Moreover, the number of videos must be constrained to facilitate adoption: while real machines can easily transcode many videos, that is not feasible for microarchitectural or RTL simulation.

Video Feature Selection

From the many features describing a video, we determined three to have the greatest impact on transcoding:

- *resolution*, because higher resolution frames will require a higher bitrate to encode, and will also require more time to encode,
- *framerate*, because high framerate videos (> 30 frames/s) will require a higher bitrate to encode, and
- *entropy*, because videos with high motion, or frequent scene transitions will require a higher bitrate to encode and higher effort (for motion search and other tools), or will incur quality losses.

While resolution and framerate are simple to understand and measure, entropy requires some explanation. Throughout the paper, we use bits/pixel/second when encoded using libx264 at *visually lossless* quality (Constant Rate Factor CRF 18) as a measure for video entropy[3]. As described in Section 3.1.2, when an encoder is asked to generate a fixed target quality, it will use as many bits as needed to do so, and thus the number of bits used by the encoder in this setting reflects the inherent entropy of the video content.

From these three characteristics, we define a video *category* as the set of videos that have the same resolution, measured in Kpixels/frame ($\frac{width \times height}{1000}$, rounded to integer), framerate (frames per second, rounded to integer), and entropy (bits per pixel per second when encoded using libx264 at constant quality – constant rate factor 18 – rounded to one decimal place).

Selecting Video Categories

From logs of all the video transcoding operations at YouTube from January to June 2017, we accumulate the total transcoding time spent on each video category. This yields over

[3]libx264 Constant Rate Factor (CRF) goes from 0 to 51. CRF 0 is lossless compression, CRF 23 is the default value, and CRF 18 is generally considered *visually lossless* [project, 2017].

3500 video categories with significant weights (40+ resolutions and 200+ entropy values).

We use k-means clustering to select a small set of categories – particular triplets (resolution, framerate, entropy) – from that 3-dimensional space. Prior to clustering, we linearize resolution using the base two logarithm. This ensures that the large distance between standard resolutions does not bias the clustering algorithm. We also use the base two logarithm of the entropy to quantify the relative difference between videos: videos of entropy 1 and 2 bit/pixel/s are much more different than videos of entropy 20 and 21 bit/pixel/s. Lastly, we normalize all dimensions to a [-1, +1] range. We then apply weighted k-means clustering to find a pre-defined number of centroids, with weights determined by the time spent transcoding for each category of videos. Since each centroid covers multiple categories, we select the category with the highest weight in the cluster – i.e., the mode – as the cluster representative.

This process achieves both representativeness, since we select the mode as cluster representative, and coverage, since all videos must be associated with a cluster.

Selecting actual videos

The k-means clustering defines a reduced set of ideal video categories. We then select a random video from the YouTube corpus belonging to each selected category. To ensure our benchmark is redistributable, we restrict the selection pool to videos that were uploaded with a Creative Commons Attribution 3.0 (CC BY) license [CreativeCommons.org, 2007].

Finally, we split the full-length original videos into non-overlapping 5-second chunks, and select the chunk with the bitrate that best matches the average bitrate of the video. We limit videos to 5 seconds since it has been observed to be the optimal duration for subjective video quality assessment [Mercer Moss *et al.*, 2016]. We verify that removing the creative commons restriction creates no significant difference in our results or insights. The videos that compose `vbench` are summarized in Table 3.2.

Coverage

Our process ensures that the chosen videos are representative, with each sequence covering a significant fraction of the entire corpus. However, not all categories can be covered. We

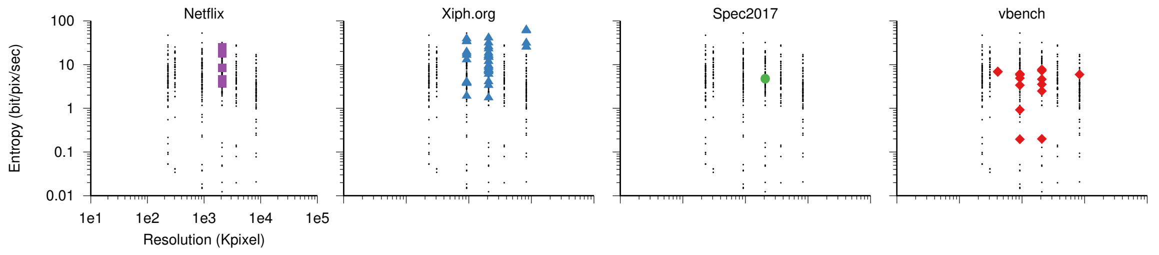


Figure 3.4: Black dots are a uniform sample of videos in the various resolutions and complexities uploaded to YouTube. Colored dots show how public video sets cover only a fraction of the space.

therefore compare our benchmark with an internal YouTube coverage set that collects 11 uniformly distributed entropy samples from the combination of the top six resolutions and the top eight framerates. These 36 resolution and framerate combinations account for more than 95% of the YouTube uploads. Figure 3.4 shows one black dot for each video in this set, and overlays the different public video sets – plus our own, `vbench` – on top to evaluate coverage.

Note that the entropy range is four orders of magnitude wide, from still images and slideshows (entropy < 1) to high motion videos with frequent scene transitions (entropy $>$ 10). In contrast, the Netflix and Xiph datasets focus only on high entropy videos (entropy $\geq$ 1) as they are intended for visual analysis. Furthermore, the Netflix dataset contains a single resolution (1080p). As we show in Section 3.5.1, the lack of low entropy videos introduces significant bias in the results using this video set. SPEC'06 and the latest SPEC'17 contain only two video sequences. This is clearly not enough for a video benchmark. Moreover, the resolution of SPEC'06 videos is not representative (too small). This is improved in SPEC'17, however the two videos used in this case have almost identical entropy (Figure 3.4) as they are obtained from the same animation. `vbench` achieves better coverage in both resolution and entropy than all of these other alternatives, and has fewer and shorter videos than Xiph.org to facilitate adoption.

3.4.2 Transcoding Scenarios

To capture the nuances of the various video processing pipelines outlined in Section 3.2, `vbench` distinguishes five scoring scenarios. Each scenario reflects the *constraints* and *priorities* of its corresponding real-world scenario: (1) uploading a new video, (2) live streaming, (3) archiving for video on demand, (4) optimizing popular videos, and (5) optimizing the hardware platform.

For each scenario we provide reference measurements, namely speed (in Mpixel/sec), bitrate (in bits/pixel/sec), and quality (in dB), all normalized to video resolution and duration to allow comparison across videos. The measurements for each of these scenarios are taken using ffmpeg with libx264 on a Intel Core i7-6700K CPU @ 4.00GHz with 4 cores and 8 threads. Each of these reference transcoding operations is a measuring stick, grounded in real-world video sharing infrastructure, with which to compare transcoding solutions. All ffmpeg parameters used are reported in the `vbench` website (*http://vbench.net*).

The `Upload` reference is single pass with a constant quality target, allowing the encoder to use as many bits as needed to maintain the quality of the original. The `Live` reference is single pass, low latency, with a fixed bitrate target; the encoder effort is lower for higher resolution videos to ensure that the latency constraints are met. The `VOD` reference is the average case and is the same as the `Platform` reference: two-pass encoding with a fixed bitrate target. Finally, the `Popular` reference is high-effort two-pass encoding. The reference measurements are **scientifically essential**. They ensure that `vbench` results reported by different groups are directly comparable, and that the baseline is meaningful.

Users of the benchmark will try to improve on the reference transcoding operations provided. `vbench` uses ratios (speedups) between a new system and a reference transcode to indicate improvement. Values greater than 1 indicate the new solution is *better* in that dimension.

$$S = \frac{Speed_{new}}{Speed_{ref}} \qquad B = \frac{Bitrate_{ref}}{Bitrate_{new}} \qquad Q = \frac{Quality_{new}}{Quality_{ref}}$$

Since video transcoding entails a trade-off between speed, size, and quality, it is unlikely

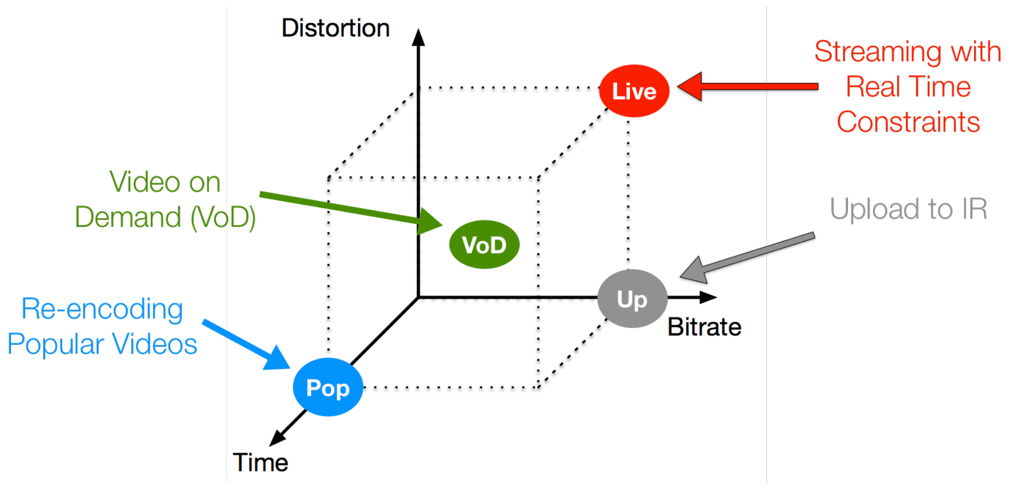


Figure 3.5: Different transcoding scenarios impose different tradeoffs. Therefore, a transcoding solution should be evaluated differently for each scenario. The ideal transcoder (zero time, infinite compression, no distortion) corresponds to the origin of the plot.

that a new solution will Pareto dominate the reference transcodes on all three dimensions. However, a new solution has only to be evaluated for the scenario in which it is going to be deployed. Figure 3.5 depicts graphically how different scenarios included in `vbench` prioritize one metric over another. As an example, Live streaming has to prioritize speed over bitrate or visual quality. Each `vbench` scenario has an associated scoring function where one dimension is eliminated via a strict Quality of Service constraint, leaving ratios for the remaining two dimensions. These constraints are reflective of the particular priorities of the transcoding scenario targeted. It is then possible to condense each video down to a score by multiplying the two ratios, similar to an energy-delay product [Gonzalez and Horowitz, 1996]. These scores, summarized in Table 3.1, are easy to compare yet reflective of nuanced real-world constraints and trade-offs.

The `Upload` transcoding pass requires speed and quality: the video should be available for further processing as soon as possible, while not degrading the quality of the uploaded original. On the other hand, bitrate can be almost arbitrarily large because it is only a temporary file. We therefore require the bitrate be no larger than 5x the reference ($B > 0.2$) when reporting `Upload` scores of $S \times Q$.

`Live` streaming must happen in real time, so transcode must not lag behind the pixels

Scenario	**Constraint**	**Score**
`Upload`	when $B > 0.2$	$S \times Q$
`Live`	when $S_{new} \geq outputMpixel/s$	$B \times Q$
`VOD`	when $Q \geq 1$ or $Q_{new} \geq 50dB$	$S \times B$
`Popular`	when $B, Q \geq 1$,$S \geq 0.1$	$B \times Q$
`Platform`	when $B, Q = 1$	S

Table 3.1: `vbench` scoring functions and constraints.

per second of the output video. The `Live` score is then $B \times Q$.

In the `VOD` scenario, one cannot degrade quality compared to the reference, as this would have negative effects on user experience. However, provided quality is maintained ($Q \geq 1$) or the transcode is visually lossless ($Q_{new} \geq 50dB$) one can report a `VOD` score of $S \times B$.

High-effort optimizations for `Popular` videos should always produce smaller videos of higher quality. Improvements on visual quality and reduction in network bandwidth will improve user experience, while extra compute cost of re-transcoding popular content will be amortized across many playbacks of these popular videos. In this case, we report bitrate and quality: $B \times Q$ (if $B \geq 1$ and $Q \geq 1$). While speed is not critical in this scenario, it should still be bounded to a 10x slowdown ($S \geq 0.1$).

The final `vbench` score captures the case where the encoding algorithm and settings are constant and only the `Platform` changes. Innovations evaluated in this scenario are the same as SPEC benchmarks: compilers (icc vs gcc), architecture (x86 vs PPC), and microarchitecture (cache size, branch prediction, etc.). The scoring function assumes that bitrate and quality will be unaffected, and thus the two platforms can be compared by reporting S (if $B = 1$ and $Q = 1$).

3.4.3 Reporting Results

For each scenario, a complete run of the benchmark requires a transcode operation for each input video sequence that is compared against the reference. Each transcode operation results in three values – speed, bitrate, and quality – reported individually. For each video, if the constraints specific to the scenario are satisfied, scoring metrics described in the previous section can be computed. Given the diversity of the videos, results should not be aggregated into averages as significant information would be lost. Each video reflects some segment of the video sharing workload, so that providers, who know the specifics of their corpus and service costs can weigh the information accordingly, similar to what is done today for SPEC.

We demonstrate how benchmark results should be reported in the next section.

3.5 Bridging the Performance Gap for VOD

In this section we analyze the performance of different video transcoding solutions on the `VOD` scenario, looking for opportunities to improve performance, and understanding the tradeoffs that they represent. Throughout the section, we will use the coverage corpus described in Figure 3.4 as a golden reference, comparing the trends, correlations, and insights obtained with it to those of `vbench`.

3.5.1 CPU Performance

First, we examine how video transcoding exercises the CPU microarchitecture. We found that the microarchitectural profile of video transcoding is very sensitive to the input video, which reinforces the need of a validated benchmark. Furthermore, its performance on general purpose CPUs is better than the typical datacenter workload, e.g. websearch, with respect to retiring rate, frontend stalls and bad speculation stalls [Kanev *et al.*, 2015].

Figure 3.6 shows how L1 instruction cache misses, branch mispredictions, and last level cache misses correlate with video entropy[4]. Each plot shows two sets of data: black dots for the coverage corpus and colored dots for the various benchmark suites.

Our results show that transcoding of complex videos incurs more icache misses, and more branch mispredictions per kilo instructions. As videos become more complex, the

[4]Measurements for Figures 5 to 7 are reported on a Google corporate machine different from the `vbench` reference: a Xeon E5-1650v3 with 32 GB of DDR4. This was necessary to not distribute user data that was not Creative Commons.

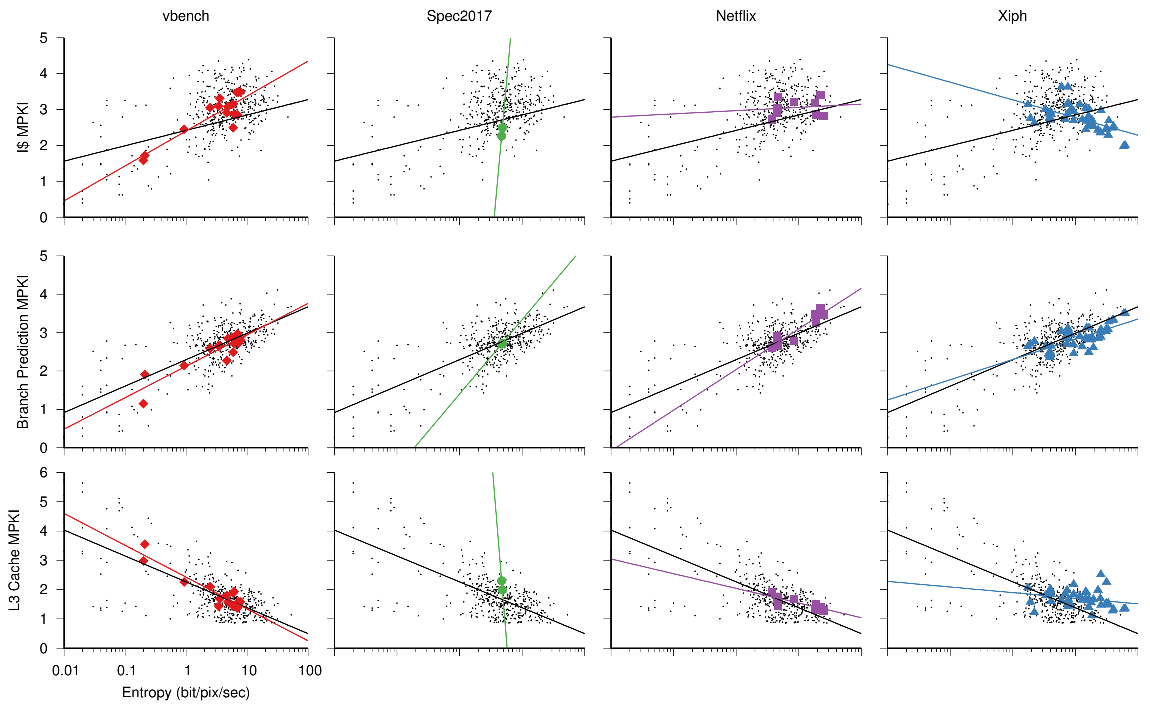


Figure 3.6: These plots overlay the videos from the different benchmarks (colored dots) on the coverage set (black dots). In both the coverage set and `vbench`, videos with higher entropy have worse frontend behavior, and reduced last level cache miss rates. The lack of low-entropy videos in the Xiph.org and Netflix datasets leads to different microarchitecture performance trends: lower branch misprediction for high entropy videos, no correlation between video entropy and LLC misses. Logarithmic interpolation ($y = a * \log(x) + b$) is used to obtain trends.

encoder needs to use more advanced compression tools in order to meet the bitrate constraint without degrading quality. This requires exercising more code, which leads to worse frontend performance. At the same time, complex videos incur a lower LLC miss rate. The memory footprint of a video depends only on its resolution, not on video entropy. Transcoding more complex videos will execute more instructions on the same data, leading to higher temporal reuse, hence the lower cache miss rate. In all cases, the trend observed using the `vbench` video suite matches the trend exposed by the much larger coverage corpus.

Figure 3.6 also reveals how the choice of the video set can lead to different microarchitecture trends. Since these trends are most visible if low entropy videos are present, the high

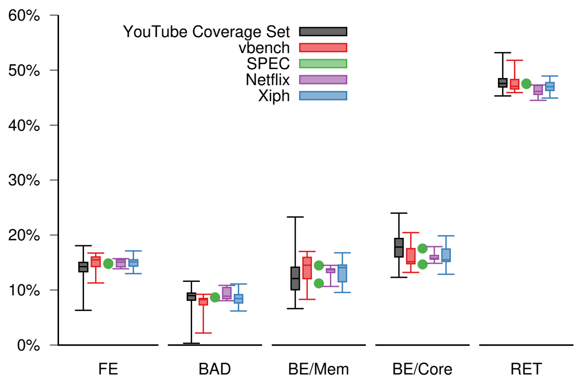


Figure 3.7: Statistical distribution of the fraction of time spent on frontend, bad speculation, memory, backend, and retiring instructions. 60% of the time is either retiring instructions or waiting for the backend functional units.

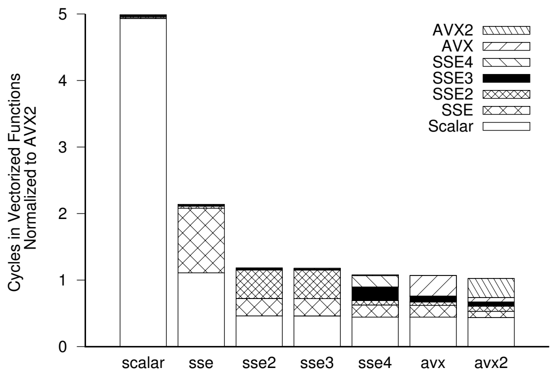


Figure 3.8: Time breakdown for H.264 transcoding across different SIMD instruction sets. The fraction of time spent in scalar code remains constant and becomes increasingly dominant. New vectorization extension only provide limited benefits.

entropy of the videos in Netflix and Xiph.org biases results. The Xiph.org set shows the opposite trend on icache misses, while the Netflix set shows no correlation between icache MPKI and video entropy. A similar error appears on LLC misses, where the Xiph.org set shows no correlation between them and video entropy.

Figure 3.7 translates these microarchitecture event counters to performance using the Top-Down methodology [Yasin, 2014]. For all video sets, we show boxplots (minimum, maximum, first and third quartiles, and median values) for the % of time spent on frontend (FE), bad speculation (BAD), waiting for memory (BE/Mem), waiting for the backend (BE/Core), or retiring instructions (RET).

Our results show that for all sets 15% of the time is spent on frontend stalls, 10% on bad speculation, 15% of the time is spent waiting for memory (decreasing for higher entropy videos, lower LLC misses), with the remaining 60% spent retiring instructions or waiting for functional units. Except for maximum and minimum values, the results observed with `vbench` closely match those obtained with the coverage corpus.

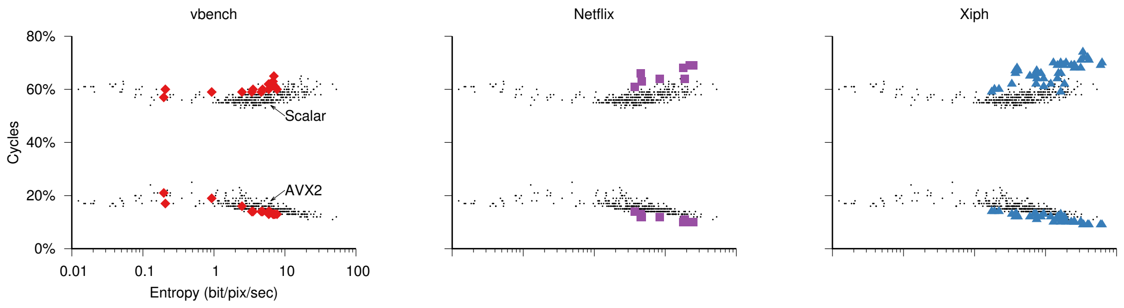


Figure 3.9: Fraction of time spent in scalar (non-vector) and AVX2 (long vectors) instructions as a function of video entropy. Over half the executed code is not suitable for SIMD acceleration, and less than 20% of the code would benefit from longer vectors. The high entropy videos in Netflix and Xiph show slightly higher ratios of scalar code.

3.5.2 SIMD Analysis

The high fraction of time retiring instructions or waiting for backend resources indicates an opportunity for performance improvement by increasing the number of functional units and their data width. Exploiting data-level parallelism with wider SIMD functional units does both at the same time.

Media processing was a major reason for the introduction of SIMD in consumer processors in the 1990s. Indeed, video transcoding is amenable to SIMD acceleration because most of its kernels (DCT, quantization, motion estimation, etc.) are performed on blocks of adjacent pixels. However, not all the video transcoding process can be vectorized. Figure 3.9 shows the fraction of scalar instructions, and of AVX2 vector instructions as a function of video entropy.

Scalar code represents close to 60% of the instructions on all videos, regardless of their entropy. Focusing only on videos with entropy greater than 1, we observe a slight increase in the scalar fraction as entropy increases and a corresponding reduction of AVX2 instructions. Non-vectorizable functions include all the decision making logic, e.g. the frame reference search for motion estimation which averages 9% of the time, or functions that are strictly sequential and control dominated, e.g. entropy encoding which averages 10% of the time.

Figure 3.8 shows the fraction of the execution time in the different instruction sets as we progressively enable newer SIMD extensions in libx264. Our results show that the scalar fraction has been stable since the introduction of SSE2. New ISA extensions have accelerated the already vectorized fraction of time, but have not increased code coverage. Moreover the performance improvement from SSE2 – an ISA introduced more than fifteen years ago – is only 15%.

Furthermore, our results show that AVX2, which doubles vector width to 256 bits with respect to AVX, only partially replaces it and represents only 15% of the runtime. The remaining vectorized code does not benefit from 256-bit wide SIMD registers due to the width of macroblocks being smaller than the AVX2 vector length. Amdahl's Law limits the potential impact of a 2x wider SIMD extension to less than 10%, even if we assume that time spent in AVX2 instructions scales perfectly with vector size.

We conclude that performance of video transcoding on CPUs is limited by the scalar fraction of the code not suitable for data-level parallelism. To achieve significant speedups, processors could be enhanced with special functional units targeting increased code coverage [Qadeer *et al.*, 2013; Hameed *et al.*, 2010], and 2-dimensional SIMD extensions that exploit data-level parallelism across the entire macroblock [Corbal *et al.*, 1999]. Otherwise, we must resort to full implementations of video transcoding in hardware.

Notice that suites other than `vbench`, since they contain only high entropy videos, have a slightly larger fraction of time spent in scalar code, and a lower fraction in AVX2 code than what we observe in the coverage corpus. In both Xiph and Netflix sets, only 11% of the time is spent in AVX2 code compared to 14% and 15% for vbench and the coverage set, respectively. This predicts an even lower benefit from vectorization.

3.5.3 Hardware Accelerators

Contrary to SIMD extensions, end-to-end video transcode solutions are not limited by Amdahl's Law because they cover the entire algorithm, including the control flow and bitstream manipulation. In addition, they can exploit functional level partitioning and parallelism across different stages of the algorithm.

Our results show that hardware encoders provide significant improvements in terms of speed at the cost of increased bitrate. Hardware transcoders need to be selective about

Resolution (Kpixel)	Name	Entropy (bit/pix/sec)
854x480	cat	6.8
(410 Kpixel)	holi	7.0
1280x720	desktop	0.2
(922 Kpixel)	bike	0.9
	cricket	3.4
	game2	4.9
	girl	5.9
	game3	6.1
1920x1080	presentation	0.2
(2074 Kpixel)	funny	2.5
	house	3.6
	game1	4.6
	landscape	7.2
	hall	7.7
3840x2160	chicken	5.9

Table 3.2: `vbench` videos' description.

NVENC			QSV		
S	B	VOD Score	S	B	VOD Score
5.74	0.76	4.36	9.27	0.80	7.38
5.04	0.76	3.83	7.95	0.80	6.38
2.41	0.40	0.96	3.90	0.18	0.72
4.05	0.62	2.52	6.68	0.73	4.91
8.91	0.83	7.39	13.22	0.70	9.32
7.72	0.64	4.97	12.94	0.71	9.20
8.51	0.93	7.88	14.29	0.80	11.46
9.22	0.52	4.81	11.32	0.80	9.05
3.58	0.35	1.24	4.35	0.48	2.09
9.63	0.43	4.10	11.17	0.83	9.30
14.29	0.93	13.34	16.75	0.96	16.02
14.87	0.57	8.50	15.89	0.72	11.42
15.05	0.88	13.26	18.50	0.94	17.36
13.68	1.14	15.58	18.64	0.94	17.51
19.12	0.85	16.31	20.00	0.83	16.58

Table 3.3: VOD score improves with video resolution for NVIDIA NVENC and Intel QSV on the VOD scenario.

NVENC			QSV		
Q	B	Live Score	Q	B	Live Score
1.01	1.09	1.09	1.02	1.14	1.16
1.00	1.21	1.21	1.01	1.28	1.29
1.06	1.03	1.09	1.88	0.16	0.30
1.03	1.31	1.35	1.25	0.48	0.59
1.00	1.29	1.29	1.01	1.14	1.16
1.00	1.20	1.20	1.02	1.30	1.32
1.01	1.16	1.17	1.01	1.45	1.47
1.01	0.96	0.97	1.01	1.28	1.29
1.05	0.79	0.83	1.34	0.31	0.42
1.01	1.01	1.02	1.00	1.69	1.69
1.00	1.53	1.54	1.01	1.68	1.70
1.03	1.19	1.22	1.01	1.57	1.59
1.01	1.19	1.21	1.01	1.26	1.27
1.02	1.28	1.31	1.01	1.45	1.46
1.01	2.10	2.12	1.01	2.42	2.44

Table 3.4: Intel QSV and NVIDIA NVENC achieve real-time performance on the Live scenario.

LIBVPX-VP9			LIBX265		
Q	B	Pop. Score	Q	B	Pop. Score
1.00	1.47	1.48	1.02	1.17	1.19
1.00	1.06	1.06	1.01	1.12	1.13
1.01	0.67		1.00	0.87	
1.00	1.06	1.06	1.01	1.11	1.12
1.01	0.97		1.02	0.86	
1.00	1.33	1.33	1.01	1.03	1.04
1.01	1.06	1.06	1.02	0.81	
1.01	1.09	1.10	1.01	0.80	
1.00	1.86	1.86	1.00	1.13	1.13
1.00	1.37	1.37	1.00	1.06	1.06
1.01	1.06	1.07	1.01	0.97	
1.00	1.20	1.20	1.00	1.28	1.28
1.01	1.47	1.48	1.02	1.30	1.32
1.01	1.49	1.51	1.01	1.11	1.13
1.01	1.57	1.58	1.01	1.17	1.19

Table 3.5: libx265 and libvpx-vp9 can reduce bitrate significantly while being iso-Quality on the Popular scenario.

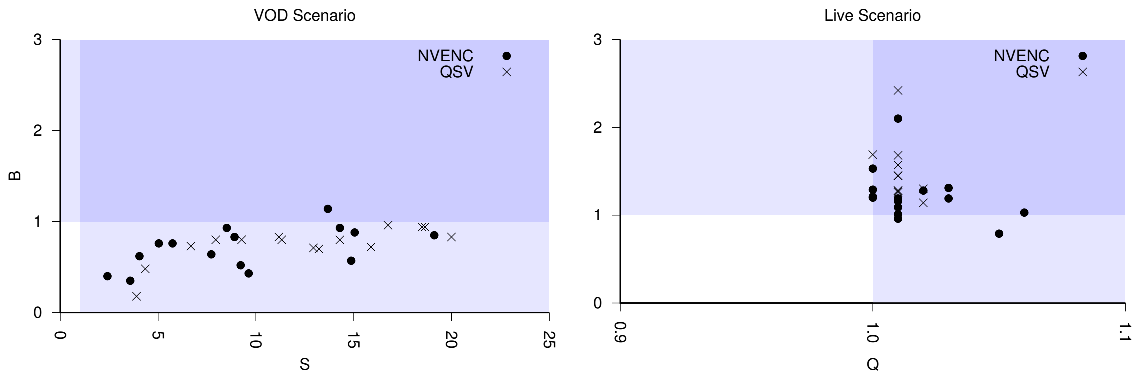


Figure 3.10: NVIDIA NVENC and Intel QSV results on the `VOD` (left), and `Live` (right) scenarios. The shaded areas indicate gains. While GPU adoption for `VOD` entails tradeoffs (speedups offset by losses in compression) it is an unqualified win for `Live` transcoding.

which compression tools to implement, in order to limit area and power. For example, enabling sub-pixel precision in motion search or increasing the motion search range will greatly increase the area of an implementation while providing only marginal compression improvements.

Table 3.3 reports the speed (S) ratios, bitrate (B) ratios, and `VOD` scores for two GPUs: the Intel QuickSync Video (QSV) [Corp., 2017] featured in the Intel core i7-6700K CPU, the NVIDIA NVENC [Wilhelmsen *et al.*, 2014] found in the GTX 1060 GPU. This data is also depicted in Figure 3.10. To obtain these results we used the highest effort settings on both GPUs and varied the target bitrate using a bisection algorithm until results satisfy the quality constraints by a small margin. The results show that the QSV scores are generally higher than the NVENC scores; this is mostly due to the higher speed ratios since bitrate ratios are comparable in the two platforms. Unfortunately, we cannot offer much deeper explanation for the difference as the GPUs do not allow software to inspect intermediate results, effectively creating a black box.

Both GPUs show higher speed improvements for higher resolution videos, since they better amortize the data transfer overheads, and enable higher parallelism across the macroblocks. They also show higher speedups for more complex videos, since they perform all the operations in parallel, while the software needs to run for a longer time to apply the more complex encoding tools: Having a curated video set is important here. Using Xiph or Netflix dataset – both containing only high resolution, high entropy videos – would overestimate the benefits of GPU transcoding.

Both GPUs show significant speed benefits that compensate the losses in video bitrate. Their higher speed would allow a significant downsizing of the transcoding fleet at a video sharing infrastructure. However, they need to compensate with an increase in storage and network costs. The precise balance between compute costs, storage, and network will depend on the specifics of the service, reinforcing the need to report metrics separately in addition to the score.

Given the speedups achieved by these hardware implementations, future hardware video transcoders might implement more advanced encoding tools to trade slower speed for higher video quality at lower bitrate, enabling service providers to tune them to their specific use cases.

3.6 Live and Popular Analysis

That was the `VOD` scenario. We now evaluate how GPUs perform on the `Live` and `Popular` scenarios. Both are key use cases in services like YouTube and Facebook.

3.6.1 Live Streaming

While speed is important for `VOD`, it is critical for the `Live` scenario. GPUs here shine as low latency transcoding is their intended application, while software encoders have to significantly decrease effort levels in order to meet the real-time speed constraint. In fact, to meet the real time constraint our reference transcodes have an effort level that is inversely proportional to the resolution of the input video. This explains the positive GPU `Live` results seen in Table 3.4. When real time speed is required, software encoders degrade the transcode quality much more than hardware.

There are a number of configurations for these GPUs that would have met the `Live` scenario constraints. For this experiment, we chose to maintain reference quality, which creates an interesting comparison with `VOD`. Contrary to what we have observed in the

VOD scenario, using GPUs in this case generally incurs no tradeoffs. Our results show that hardware encoders achieve the same quality as our reference while **also** reducing the transcode bitrate. The only exceptions being low entropy videos, for which the GPUs struggle to degrade quality and bitrate gracefully. Had the benchmark not included low entropy videos and different scenarios, such insights would not be visible.

Even on the Live scenario, where hardware encoders do not incur sacrifices in bitrate and quality, we find that hardware encoders exceed real-time and thus are potentially faster than they need to be. As with VOD this again raises the possibility that the excess speedups seen on GPUs might be better spent finding higher quality transcodes.

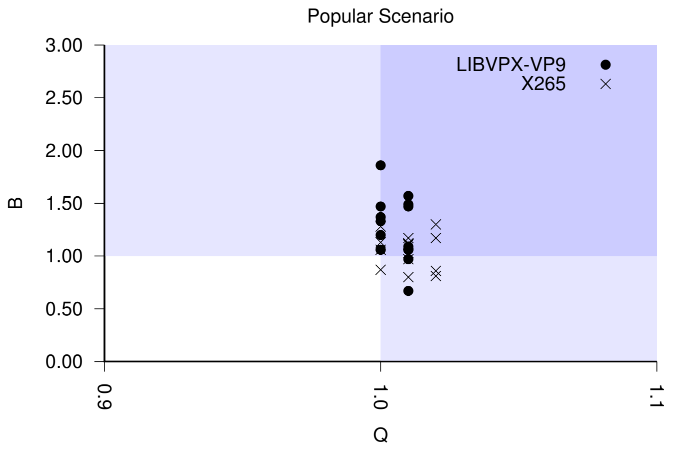


Figure 3.11: NVIDIA NVENC and Intel QSV results on the Popular scenario. Newer encoders such as Libvpx-VP9 and x265 can produce better transcodes in most cases when compared to our Popular reference, albeit at a higher computational cost.

3.6.2 Popular Videos

The Popular scenario deals with very high effort transcoding for videos that receive the most playbacks. As we saw in the VOD scenario, the hardware video transcoders require additional bitrate to match the quality of the reference transcodes in the VOD scenario. Given that the reference quality of the Popular scenario is higher than VOD, it was impossible for either of the GPUs to produce a single valid transcode for this scenario. GPUs are valuable in the VOD and Live scenarios because of their speed. However, speed is the least valuable metric in the Popular scenario. Software encoders are the best option today for optimizing bitrate and quality on highly popular videos, where the effort will be amortized across many playbacks.

Table 3.5 shows the benchmark scores for the recent libx265 and libvpx-vp9 encoders when transcoding `Popular` videos. This data is also depicted in Figure 3.11. For both, we selected a fixed effort level such that all videos can be encoded within the speed constraint[5]. Our results confirm what was seen earlier in Figure 3.2: both libx265 and libvpx-vp9 encoders are superior to the reference libx264 when transcoding speed is not considered. Both encoders are capable of significantly reducing bitrate for most videos at no quality loss, especially for HD and higher resolutions. Notice that the reference transcode operations in this scenario use the highest quality setting in libx264. This reflects how newer codecs (H.265, VP9) and relative encoders keep improving video compression, a trend that is expected to continue with the release of the AV1 codec by the end of the year [Foundation, 2017].

3.7 Conclusions

We have described `vbench`, a video transcoding benchmark that has an algorithmically curated set of representative videos, metrics, and reporting guidelines that reflect complex real world tradeoffs between quality, compression, and speed. `vbench`'s quantitatively selected dataset improves on existing video collections assembled based on qualitative visual properties. With the same methodology we can update `vbench` videos over time to reflect changes in upload trends. The reference transcoding operations reflect those of large video sharing infrastructures and are thus a useful baseline. The metrics guide improvement over this baseline in meaningful directions.

Our studies of video transcoding reveal a microarchitectural sensitivity to inherent video entropy. There are limits on the benefits of vectorization for this workload while the viability of hardware support for transcoding depends strongly on the context; current GPUs are well suited for live streaming yet unable to meet the quality and compression demands of popular videos. Each video and scenario combination is unique and highlights an important part of the video sharing infrastructure workload. If the benchmark did not capture all this variability incorrect conclusions could have been made in the process of designing an accel-

[5]cpu-used 0 for libvpx-vp9 and -preset veryslow for libx265. A empty score indicates that either the bitrate or quality constraints are not met (highlighted in red)

erator for this workload. Therefore, the insights gained from developing and using `vbench` confirm part of our thesis that *Design Space Exploration requires benchmarks reflective of production workloads.*

We expect this benchmark and the insights it will produce will promote much needed innovation in video transcoding, a key warehouse scale workload that is at the core of all video sharing infrastructures.

Chapter 4

Design Space Exploration of Accelerators for Analytical Query Processing

Data mining is the process of generating new insights from data. In the contemporary age of Big Data [McAfee and Brynjolfsson, 2012], the performance and energy efficiency of these workloads has become critical since they are the main driver in creating value out of user data. However, they have to perform efficiently in order to be viable and keep up with data growth.

Analytical query processing is a mature and critical data mining workload, and it is also often used as an intermediate step for more complex data mining pipelines, e.g. as a preprocessing step before training a machine learning model. It is therefore not surprising that there have been multiple academic proposals to augment Database Management Systems (DBMS) with specialized hardware [Woods *et al.*, 2014; Chung *et al.*, 2013; Jun *et al.*, 2015], as well as recent industrial designs that implement such solutions [Ouyang *et al.*, 2016; Agrawal *et al.*, 2017]. In this chapter, we are going to focus on an accelerator architecture proposed by our group, the Q100 [Wu *et al.*, 2014; Wu *et al.*, 2015]. We will analyze how design space exploration affects this class of accelerators for analytical query processing.

The Q100 contains a heterogeneous set of fixed-function processing elements; *tiles* in

Q100 terminology. Each tile implements a relational operator such as a join or filter. These processing elements operate on streams of data corresponding to columns of the database. The Q100 architecture can readily exploit both pipeline parallelism (performing different operations on the elements of a column), and data parallelism (performing the same operation on multiple columns). This allows the Q100 to process queries faster, using far less power than a general purpose system running a software DBMS.

We are going to show how design space exploration has a great effect on performance. First we will show how the mix of hardware modules that compose the final accelerator design has to be tailored to the workload. This analysis, together with an introduction of the Q100 is presented in Section 4.1. The less intuitive result that we present in greater detail is how it is possible to tailor the internal Network on Chip (NoC) of an accelerator to significantly increase performance given an area budget. To this end, we developed an interconnect synthesis algorithm that is able to produce specialized topologies.

We compare out method with two other NoC synthesis algorithms found in the literature [Koibuchi *et al.*, 2012; Ogras and Marculescu, 2006]. Our algorithm improves over these other state of the art approaches by being more resource conscious. As others do, we expose a parameter than can be tuned—effectively the degree of specialization—that allows a designer to produce networks that fall between general-purpose and fully specialized. However, unlike the state of the art, we introduce dedicated, low-contention links for the most important communication pairs and connect *the remainder* of the ports through a generic topology. In Section 4.3, we present our algorithm and prove it generates deadlock-free networks. Our experiments show that we produce networks which are faster and smaller than competing algorithms. We put more of the high-volume communication on dedicated links while reducing the size of the standard interconnect that serves the less-important traffic.

In Section 4.5, we use our algorithm to generate a custom interconnect for a Q100 instance. This interconnect allows the device to execute the TPC-H benchmark only 1.24× slower than a fat tree topology while consuming only 82% of the area of a double ring. Compared with NoCs produced by competing algorithms [Ogras and Marculescu, 2006; Koibuchi *et al.*, 2012], our algorithm reduces NoC area by 25% while increasing system

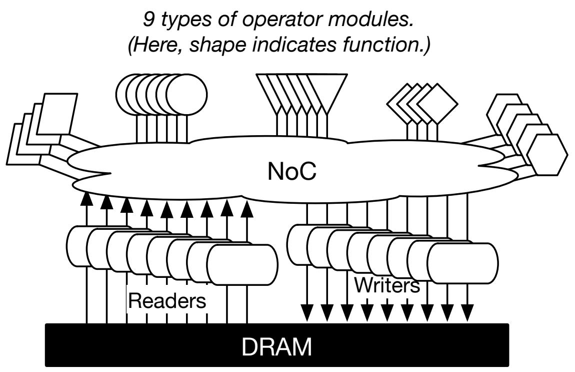

9 types of operator modules.
(Here, shape indicates function.)
NoC
Readers
Writers
DRAM

Figure 4.1: The Q100 [Wu *et al.*, 2014] 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 that offers all-to-all connectivity.

performance by 1.21×.

In this chapter, we assume that the reader is familiar with basic Database concepts [Ramakrishnan and Gehrke, 2003]. To represent the workload, we use the industry standard TPC-H benchmark [Boncz *et al.*, 2014].

4.1 The Q100 Analytical Query Processing Accelerator

The Q100 [Wu *et al.*, 2014; Wu *et al.*, 2015] 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. We describe here the key properties of the architecture and its toolchain. This architecture is fully detailed in our papers [Wu *et al.*, 2014; Wu *et al.*, 2015] which also include a comparison against a software DBMS system.

4.1.1 Architecture

The accelerator contains a heterogeneous collection of hardware modules, called *tiles* in Q100 terminology. In all, there are 9 types of Q100 tile and each implements a partic-

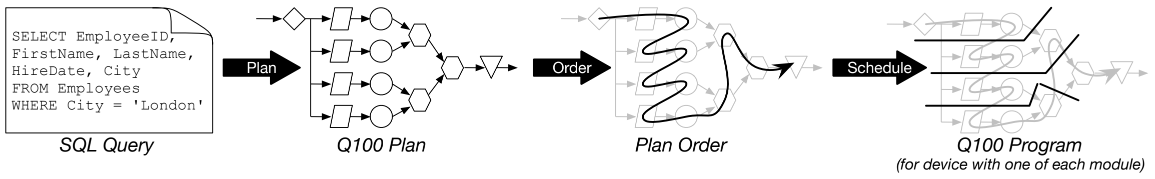


Figure 4.2: Our toolchain is able to automatically compile SQL queries into query plans that can execute on the Q100. As a first step for execution, operations are re-ordered with the objective of producing better schedules that group operations with similar latency. This reordering must still satisfy 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.

ular relational algebra operator. Those types are *Boolgen*, *Colfilter*, *Case*, *ALU*, *Joiner*, *Aggregator*, *Sorter*, *Shifter*, and *Merger*. The minimal Q100 has one of each tile, but the desirable designs will have multiple instances of each (Figure 4.1). The modules communicate directly between producer and consumer via an on-chip network which streams columns of data record by record. This network provides all-to-all connectivity amongst processing elements, but as we will see later in this chapter, it can be tailored to exploit the communication patterns of the workload. In the Q100, only dedicated *Reader* and *Writer* modules can speak to memory. To the other modules, *Readers* and *Writers* are standard data producers and consumers respectively.

4.1.2 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 operations not supported by the Q100.

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. 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.

We have developed an automated SQL compiler that generates Q100 query plans (Figure 4.2). This compiler works around limitations of the Q100, e.g. every Join must be a Sort-Merge Join to be implemented in a streaming fashion. For the same 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 reordering of joins still apply in this flow.

4.1.3 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 producer of a column must execute before or in parallel to a column's consumer. Moreover, none of the steps can exceed the available resources of the target Q100 instance. If the target contains five sorter modules, the scheduler can not schedule six sorts in one step. If an instruction in the plan operates on more input columns than the device supports, the scheduler will automatically split it into multiple instructions. Lastly, the scheduler maps each operation to a specific physical module for execution.

Like most DBMS query planning logic, this scheduler must execute in real time; scheduling the next time step while the Q100 device is operating on the current one. This is necessary to produce accurate estimates of column sizes that in turn affect scheduling decisions. Exhaustive approaches [Nowatzki *et al.*, 2013] are thus not viable. Instead the scheduler uses a greedy "Longest-Job-First" heuristic which performs on average within 5% of the best schedule obtained via a Montecarlo method[1] for the TPC-H benchmark. When mapping instructions to modules, the scheduler selects the module that requires the fewest hops to route all of the input data streams.

[1]In this method we evaluate roughly 100000 schedules and pick the best one.

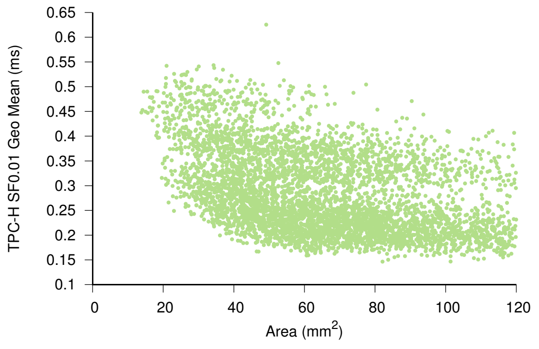


Figure 4.3: Tile mix has a significant impact on the Q100 performance. Pareto optimal designs have a tile mix that is consistent with the application requirements and are able to provide greater performance given an area budget.

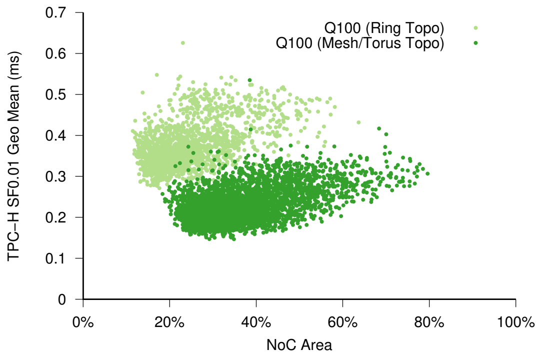


Figure 4.4: The NoC constitutes a large fraction of the area of a Q100 device. Different NoC topologies provide a different performance/area tradeoff.

4.1.4 Design Space Exploration of Compute Resources

The mix of tiles that are instantiated in a Q100 device are a clear target for design space exploration. In order to test the effects of the tile mix on the performance of the system we evaluated more than 2000 different designs with different tiles combinations[2]. In order to connect the various tiles in each design we use two standard NoC topologies: a ring and a mesh. The results of this analysis are shown in Figure 4.3. We can clearly see the great impact that tile mix has on performance. This shows how design space exploration can exploit application properties, in this case the frequency of the different computational kernel in input queries, to significantly improve performance.

4.2 Motivation for Interconnect Specialization

The correlation between tile mix and performance that we presented in the previous section is fairly intuitive. Performing design space exploration on the tile mix allows us to pick

[2]Performance values are obtained using our in house simulator while the design area is computed combining the area of each tile when synthesized using TSMC 65nm standard libraries

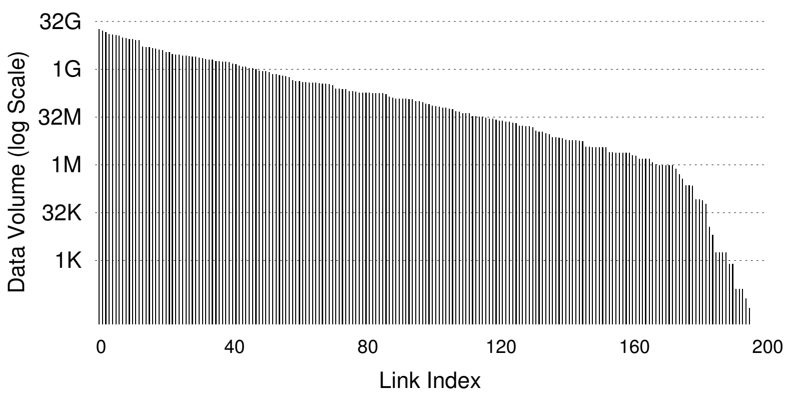


Figure 4.5: Motivation for limited specialization: the volume of link traffic falls off exponentially from the busiest links, suggesting only a handful of specialized links are necessary. (TPC-H on the Q100)

designs that are pareto optimal in terms of compute resources. In order to "extract" additional performance out of these designs we focus our attention to the NoC. Qualitatively it should be clear that the NoC is the brain behind the Q100 (Section 4.1.1) as it allows different queries to be executed by implementing the producer consumer relations between tiles. More importantly though, the NoC greatly affects the performance and cost of a Q100 device. Figure 4.4 shows how a large fraction of the area of a Q100 device is occupied by the NoC. Moreover, by changing the NoC topology it is possible to trade performance per area. A simple NoC like a double ring will impose a performance loss but at the same time reduce the area footprint of a Q100 device when compared to a larger mesh NoC.

Ideally, we would want the system to perform as if a full crossbar were present while devoting minimal area to the NoC. The fat tree and the double ring are good examples of high performance and economical NoC topologies, respectively. Moreover, under a fixed area budget, a small, performant NoC creates room for more processing tiles and potentially increased performance.

We picked one of the pareto optimal designs for further inspection. Again, we focused our attention on the application properties – producer consumer relationships between tiles in this case – and try to exploit these when designing its NoC. We found that relatively few edges carry the vast majority of data in our application, which motivates our approach to custom network design. Figure 4.5—our results from profiling the TPC-H query plans execution on the Q100—illustrates this. These patterns arise from the workload, as many tiles communicate preferentially with other tiles to implement the computation expressed by query plans. In particular, data is only ever sent over 196 of the 896 possible edges (i.e., 28 output ports times 32 inputs).

Hoping existing algorithms could exploit such patterns, we first tried applying two NoC synthesis algorithms found in the literature [Koibuchi *et al.*, 2012; Ogras and Marculescu, 2006] to the Q100 system. Since these methods are strictly additive to the base NoC topology, their solutions will be strictly larger than the base topology, and also slower due to the increased degree of the routers in the resulting NoC. We developed a network synthesis algorithm that remedies this.

4.3 Network Synthesis Algorithm

Our algorithm operates in two phases: it begins by building a partial, specialized network by considering a user-specified number of high-traffic edges and building custom, point-to-point links and routers to carry their load. After doing this, it connects all remaining edges through a generic "fallback" network. We describe this below; Figure 4.6 shows pseudocode.

4.3.1 Specialization

The algorithm starts with an empty network consisting only of ports that must be connected and, like prior NoC synthesis algorithms, a communication graph. Each node in the graph represents a type of port (e.g., the output of a filter tile or the input to the sorter tile).[3] Each directed, weighted edge indicates the relative amount or importance of the communication between the source and destination nodes. We will use the term *edge* to denote an abstract connection in the communication graph while a *link* will indicate a physical connection in a NoC.

[3]Because we target systems that allow multiple instances of each type of tile, a node in our communication graph represents a *set of interchangeable physical ports*, i.e., the same kind of port on identical tiles. This represents a slight extension over prior network synthesis algorithms that assume edges in the communication graph to have a one to one correspondence to NoC ports.

```
# Synthesize a partially specialized network
# endpoints       List of ports in the NoC, each with a type
# edges           Priority queue of communication graph edges
# to_specialize   Number of edges to specialize
# fallback        Ring, mesh, torus, or fat_tree
specialize(endpoints, edges, to_specialize, fallback):
    noc = NoC(endpoints) # Create a network of just endpoints
    r   = noc.create_router() # Add fallback router
    while edges is not empty:
        src_type, dest_type, load = edges.pop() # Get busiest edge
        # Locate the least loaded endpoints
        src  = lightest(endpoints, outgoing, src_type)
        dest = lightest(endpoints, incoming, dest_type)
        # Count how many tiles exist for both source and destination
        src_count = $|\{e : e \in \text{endpoints}, e.\text{type} = \text{src\_type}\}|$
        dest_count = $|\{e : e \in \text{endpoints}, e.\text{type} = \text{dest\_type}\}|$
        if to_specialize > 0: # Create specialized link
            min_count = min(src_count, dest_count)
            noc.add_link(src, dest, load/min_count) # May add router
            # Put the edge back after adjusting its load
            edges.push(src_type, dest_type, load − load/min_count)
            to_specialize = to_specialize − 1
        else: # Create unspecialized link to the fallback router
            noc.add_link(src, r, load)
            noc.add_link(r, dest, load)
    # Replace the fallback router with a fallback network
    r.transform_to_network(fallback)
    return noc

# Return the lightest loaded endpoint of a particular type
# endpoints List of ports in the NoC, each with a type
# direction Incoming or outgoing
# type      Endpoint type to consider
lightest(endpoints, direction, type):
    # Consider only endpoints of a certain type
    weights = $\{e.\text{direction.weight} : e \in \text{endpoints}, e.\text{type} = \text{type}\}$
    return the endpoint ∈ weights with minimum weight
```

Figure 4.6: Our algorithm for NoC synthesis.

During specialization (i.e., while *to_specialize* > 0 in Figure 4.6), we select the highest-traffic edge from the communication graph and introduces a link in the physical network to serve it. Since a Q100 device may have multiple physical replicas of a tile to exploit parallelism between operations on different columns, we select the least heavily loaded instances (as determined by *lightest* in Figure 4.6) of both the source and destination port. It then annotates the new physical link with an expected load, which is the total load on the communication edge, divided by the minimum number of instances of producer or consumer ports. If the least-loaded source or destination already has an outgoing or incoming link (respectively), this step may require introducing a router to share the port. Once a physical link is introduced, the algorithm deducts this expected load on the newly added physical link from the communication graph's edge.

Figure 4.7 illustrates this process for a simple example. The top row depicts the communication graph in which each node is labeled with a port type and count. In this example, there are two type-A ports, two type-B's, and a single C. The edges indicate data flowing from one type of port to another with a weight indicating the edge's importance. In this example, communication between A-type and B-type ports is the most important, followed by that from A to C and C to B. The bottom row of Figure 4.7 depicts the physical network under construction. Here, each physical port is represented explicitly (e.g., A_1 and A_2 are the two type-A ports), routers are introduced (the black circles), and each edge represents a physical point-to-point link with an associated expected load.

Each step from left to right in Figure 4.7 illustrates a single specialization step. In step 1, the physical link between A_1 and B_1 is marked with a load of 50 because the edge from A to B has a weight (load) of 100 and there are two instances of A and B in the target architecture. Had there been three instances of both port B and A, the physical link would have an expected load of 33.

In step 2 of Figure 4.7 we see the algorithm choosing the least loaded port when deciding which physical instance of a port should serve an edge. There the algorithm connects A_2 and B_2 because they are unused; A_1 and B_1 already have an expected load of 50. If either the source or destination port already has an incident link, we introduce a router to serve both logical communication streams, as shown in step 3 of Figure 4.7.

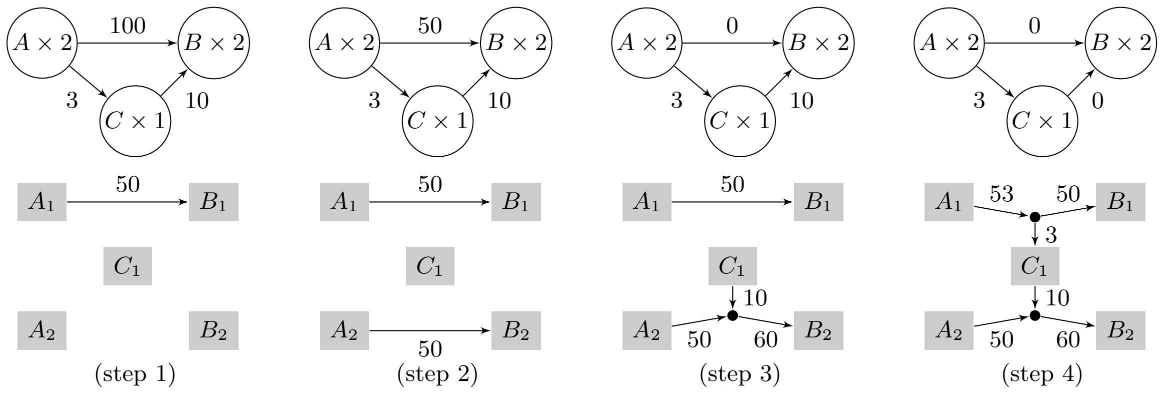


Figure 4.7: Our algorithm performing four specialization steps on a system with two A processing elements, two B's, and one C. The top graphs depict the communication patterns as observed in simulation; the bottom graphs depict the structure of the synthesized network. In step 1, our algorithm selects the highest-weight edge ($A \rightarrow B$) and adds the point-to-point connection $A_1 \rightarrow B_1$ with load 50 because there are two A's and two B's sharing the load of 100. In step 2, $A \rightarrow B$ has been reduced to 50 but remains the highest so $A_2 \rightarrow B_2$ is added. $C \rightarrow B$ is highest in step 3, but B_2 already has a connection so a router is added with a link from C_1. Finally, $A \rightarrow C$ is selected and another router is added connecting A_1 to C_1, although would also have been possible to add another link from the existing router to C_1.

The algorithm proceeds greedily for the desired number of specialization steps, at each one selects and removes the highest-weight edge from the graph. While a larger number of specialization steps produces a more customized network, more specialization is not always better because it can produce more irregular networks that are either too large, too slow, or both. By exposing the degree of specialization (the *to_specialize* parameter in Figure 4.6) we can quickly generate many networks with differing degrees of specialization, allowing the designer to evaluate and select the best option.

4.3.2 Generalization

After specialization, the remaining unconnected ports are connected to each other and the network by a standard "fallback" interconnect. In Figure 4.6, our algorithm first introduces

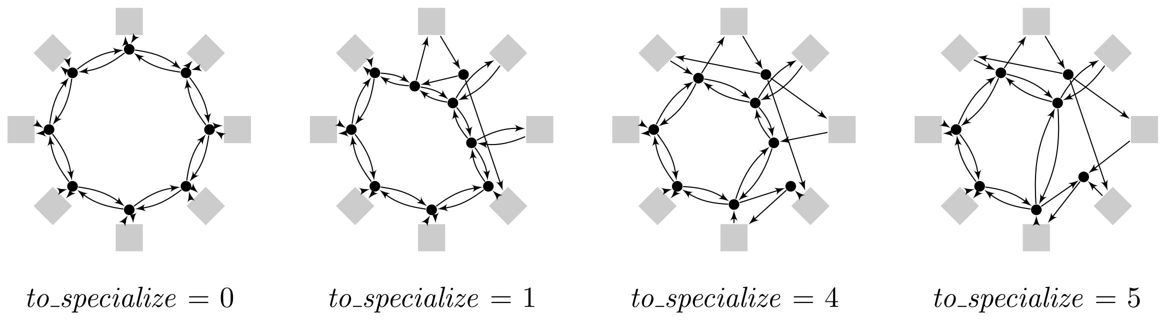


Figure 4.8: Networks produced by our algorithm employing a ring fallback for a certain system with 8 ports. More specialization steps produces a more customized network and a smaller ring.

the *fallback* router, which serves as a placeholder for the generic network and to which all edges that were not specialized during the specialization phase are connected. Once every edge has been added to the network, the *fallback* router is replaced with a standard network: we currently support ring, mesh, torus, and fat tree topologies, although others would be possible.

Figure 4.8 depicts the output of our algorithm using a ring fallback after 0, 1, 4, and 5 specialization steps. As more specialized links are introduced, the *smaller* the fallback network becomes.

4.3.3 Proof of Deadlock Freedom

Our algorithm generates networks that are free from deadlock or livelock. We show this by relying on the argument of Dally and Seitz [Dally and Seitz, 1987]: there is an ordering of channels in the generated network such that every path will traverse channels in descending order.

We rely on the fallback network already having this property. Known deadlock-free routing algorithms exist for each type of fallback network we currently support. For example, a mesh can be made free of deadlocks by using dimension ordered routing or the turn model [Glass and Ni, 1992]. As a result, we can safely treat traffic that crosses our fallback network as going through a deadlock-free black box; we can treat it as a single edge in the ordering argument.

The generated part of the network use minimal destination based routing with no adaptivity. Specifically, data follows exactly two kinds of routes. For paths considered during the specialization phase of our algorithm, our network sends data through at most two routers: one connected to the source tile and one connected to the destination tile. For all other paths, data enters at most one router, then traverses the fallback network, then traverses at most one router to reach its destination. Once data emerges from the fallback network, it never reenters.

Our algorithm enforces this property by construction. During the specialization phase, each time a new link is added, the source router's routing table is updated to steer data sent to the destination through the newly added link. For every other path, the relevant routers are instructed to send data to the fallback network instead. Furthermore, these paths do not interfere with each other and the routing tables remain fixed throughout the system's execution. Thus, every path traverses each of the following links, in order, no more than once:

source → *specialized router* → *fallback router* →
path within the fallback network → *fallback router* →
specialized router → *destination*

This is the total order on link types that Dally and Seitz's argument demands; our generated networks are deadlock-free.

4.4 Experimental Methodology

We evaluate interconnect topologies using a Q100 with 18 tiles and 16 input and output ports to memory.[4] Because a tile may have multiple inputs or outputs, the network will have 66 input ports and 76 output ports. The slight skew is due to the fact that the tiles in the Q100 design tend to have more inputs (network outputs) than outputs (network inputs). Each link is 32 bits wide.

To evaluate a network, we consider its performance relative to its size. We use CONNECT [Papamichael and Hoe, 2012] to produce FPGA-optimized, synthesizable RTL from

[4]Three *Aggregator*, two *Boolgen*, one *Sorter*, five *Colfilter*, two *ALU*, two *Joiner*, one *Merger*, two *Case*

a network description. We then synthesize the RTL using Quartus (targeting a mid-range Altera Stratix 5SGXEA7N1F45 FPGA) to obtain the network size and maximum clock frequency. In order to minimize the NoC area we fix the buffer count to the minimum amount allowed by the tool (4 entries), use Input Queued routers, and turn off all pipelining options. To calculate the overall performance of a network, we simulate TPC-H on our cycle-level Q100 simulator using the network in question. This produces a total cycle count for the workload which, multiplied by the clock period from Quartus, produces the total runtime. We then limit the frequency of the overall design to the frequency of the slowest tile in our implementation: the *merger* tile which operates at a frequency of 145MHz.

In all fallback networks we use minimal destination-based routing. The only exceptions are the fat trees which require dynamically changing routing tables to ensure non-blocking communication [Dally and Towles, 2003]. CONNECT does not support dynamic routing, so while the dynamic policy is accounted for in our simulation, the area and frequencies derived from CONNECT correspond to simpler static routers and thus should be considered lower bounds on fat tree area and upper bounds on fat tree frequency. This does not impact our final conclusions as the data show that even with these allowances, the fat trees are the largest interconnects.

To gather the communication graph, we simulate 19 (out of 22) TPC-H queries and register the amount of data flowing across between each combination of of tiles' ports; the three queries that are left out contain operators that are not supported by our current compiler infrastructure. TPC-H is the standard benchmark for analytic query processing workloads [Boncz *et al.*, 2014]. We run the queries on a database with a scaling factor of 1, meaning the whole database is 1GB.

For scheduling query plans to the finite resources of the Q100 device we employ a greedy "longest job first" heuristic that schedules the longer latency operations first on the processing element which would require the smallest number of network's hops for all its input operands. We found that, on average, this simple greedy heuristic produces schedules that are only 5% slower than the best schedule out of 10000 random valid schedules.

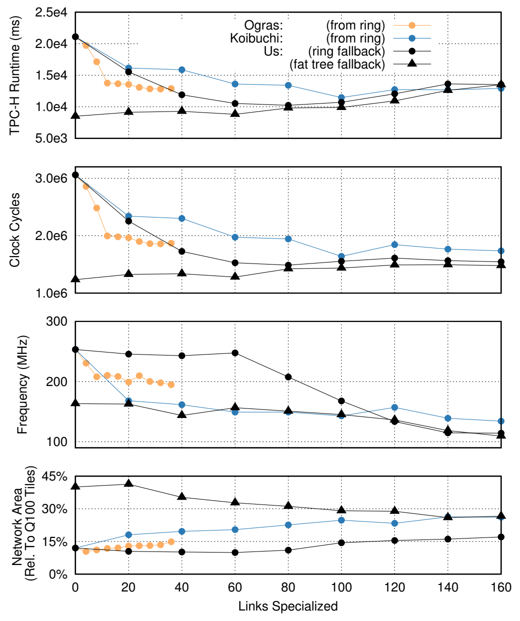


Figure 4.9: The best NoC configuration we found came from specializing 60 edges and employing a double ring fallback. This network is superior to those from other methods because our NoCs are more resource conscious.

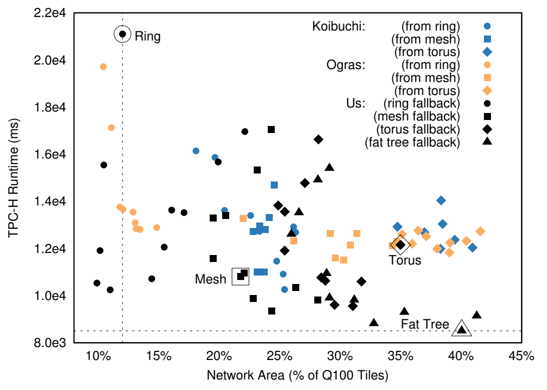


Figure 4.10: Design space exploration of semi-specialized NoC topologies. We compare our algorithm with Ogras et al. [Ogras and Marculescu, 2006], Koibuchi et al. [Koibuchi *et al.*, 2012], and four generic topologies. We find that our approach to network specialization is the most effective, producing designs that approach the performance of a fat tree and have a smaller resource-cost than a simple double ring, which are indicated by the dashed lines in the plot.

4.5 Experimental Results

In this section, we compare with two other network specialization algorithms developed by Ogras et al. [Ogras and Marculescu, 2006] and Koibuchi et al. [Koibuchi *et al.*, 2012]. The idea behind both methods is to start with a standard network topology to which dedicated links are added for important connections. Ogras, at each step, exhaustively consider all possible pairs of non-adjacent nodes and greedily select the one which reduces a cost function the most. The cost function they use is the free packet delay of each communication – a metric proportional to the hop count – weighted by the amount of traffic it carries. Koibuchi et al. instead produce a fixed number or randomly augmented graphs – they found 100 to be a reasonable number – from which they select the one with the smallest diameter. Although the original paper by Ogras et al. only considered a mesh base topology, we apply their method to specialize a ring and torus as well. Like us, both these strategies seek to balance the benefits of specialized and generic network topologies. However, whereas their specialization is strictly additive to the generic network starting point, we start with no network, introduce dedicated links, and lastly fill in with a generic network.

For all algorithms, we sweep the degree of specialization from no specialization (i.e., the base network topology) to fully specialized. Figure 4.10 plots the performance and area trade-off of the resulting networks for different degrees of specialization. It is clear that the most desirable networks are bidirectional rings that are slightly specialized. This is consistent with the observed patterns of communication for the Q100 accelerator (Figure 4.5) where a small set of edges carry most of the traffic. Koibuchi et al. also observe how ring topologies provide the most benefits with their approach. However, the networks produced by our synthesis algorithm outperform both standard topologies and the specialized topologies produced by the other algorithms. Figure 4.10 shows how all points in the Pareto frontier, other than the fat tree, are NoCs produced by our algorithm.

To develop our intuition as to why our method outperforms the others, in Figure 4.9 we analyze the impact of growing specialization on all figures of merit: completion time, clock cycles, frequency, and area. In these plots x links specialized for Ogras and Koibuchi's method means x dedicated links added, and for us x specialization steps. Note that Ogras does not specialize beyond 40 links due to the method's constraint that each router can have no more than one long-distance link.[5] We plot the data for the strongest base topology, the ring, as well as the fat tree which was the least congested at all times[6].

Starting with no specialization, Figure 4.9 shows that a ring topology will have the smallest area, but performs poorly. As we increase specialization, the number of clock cycles

[5]We experimented with relaxing this constraint to create a fairer comparison to our method. However, this did not improve performance and since this was not part of the original algorithm, we do not report results for it.

[6]We do not apply Ogras or Koibuchi's method starting from a fat tree since it is already non-blocking and therefore will not benefit from extra links.

tends to drop, but the frequency does too as the specialization breaks the regularity of the network. For the same reason the area of the ring will increase. Therefore, performance will not necessarily increase with specialization. This is exactly what limits the other techniques we are comparing against. Networks produced by Koibuchi or Ogras' method might outperform ours when considering raw clock cycle count and a given specialization target. However, they are much larger and can only operate at slower frequencies with respect to our more resource conscious NoCs. It is also clear that using information about the traffic patterns helps ours and Ogras' method produce networks that are better than what the random approach can achieve. Finally, notice how a dense fallback network like a fat tree "slims down" as we remove incoming and outgoing connections. However, because the specialized network is now less versatile, congestion slowly increases as well. For all methods frequency drops rapidly after 60 links are specialized, emphasizing the importance of regularity in the NoC structure.

4.6 Related Work

Networks-on-Chip, or NoCs, are a *de facto* standard for System-on-Chip integration [Dally and Towles, 2001; Benini and De Micheli, 2002]. Application-specific network synthesis techniques have been widely investigated to improve and automate the design of networks for SoCs. The typical approach considers a fixed communication graph with clearly defined endpoints.

We compare against the two recent network specialization techniques of Koibuchi et al. [Koibuchi *et al.*, 2012] and Ogras et al. [Ogras and Marculescu, 2006]. Both approaches start with a standard NoC topology then augment it with additional links to bypass congestion. The criteria for when a new link is introduced differ between the two methods. Koibuchi et al. produce a new NoC by adding random links in different ways then picking the resulting NoC that shows minimum diameter. This approach is meant to reduce congestion by reducing the amount of hops for all paths regardless of the amount of traffic they carry. In contrast, Ogras et al. use traffic traces to decide which link to add.

By contrast, we begin with an empty network, build links for a user-specified number of high-traffic paths, and finally connect all remaining paths with a network of standard topology. In Section 4.5 we compare the quality of the network our algorithm produces with both Koibuchi and Ogras's method, and find that our approach exploits specialization in a more resource-conscious way.

Notice that neither our method, nor the other two methods we compare against [Koibuchi *et al.*, 2012; Ogras and Marculescu, 2006] consider communication timing [Ho and Pinkston, 2003].

A fully generative approach to interconnection synthesis never uses a standard topology. With such techniques, since the space of possible solutions is much larger than for additive techniques, additional constraints are necessary for these methods to find a solution in a reasonable time. Murali et al. [Murali *et al.*, 2006] provide an algorithm that hierarchically partitions the communication graph. Pinto et al. [Pinto *et al.*, 2002] propose an algorithm relying on weighted unate covering solvers to generate a NoC. Srinivasan et al. introduce an approach that specifies the synthesis problem as an ILP formulation [Srinivasan *et al.*, 2006]. All these methods use floorplanning and frequency constraints to reduce the space of admissible solutions. In our case we operate at a higher level of abstraction, with no restrictions on the placement of the NoC endpoints. Therefore, these generative methods can not be readily applied.

Lastly, there are techniques to map computation into a standard topology such as a mesh or a torus (e.g. NETCHIP [Bertozzi *et al.*, 2005]). These solutions are orthogonal to our work and could be applied to further improve performance of the standard fallback networks we utilize.

4.7 Summary and Conclusions

The Q100 is a promising class of accelerators that target analytical query processing, a ubiquitous and mature application. We have shown how Domain Space Exploration – both of computation and communication resources – has a great impact on performance for this class of accelerators. First, we presented how it is possible to tailor the mix of computational resources to the relative frequency of relational algebra operators in the input queries in order to improve performance. Then, we presented an algorithm that allows us to tailor the NoC to the communication patterns between hardware modules that compose the accelerator.

The algorithm we presented allows rapid network design space exploration by sweeping the number of specialized links. It finds networks that perform better per unit area than standard topologies and custom networks obtained by other algorithms for NoC synthesis. In particular, our algorithm found networks that improve application performance by 1.21× and reduce area by 25% relative to state-of-the-art network synthesis techniques.

Our work shows the importance of optimizing the interconnect of an accelerator by exploiting the pattern of communication between the accelerator hardware blocks.

Chapter 5

Levels of Specialization for Accelerators

In the previous chapter, we have shown how it is possible to exploit application properties to drive design space exploration of accelerators. While the approaches we presented are very effective at increasing performance given an area budget, they do suffer from some qualitative limitations.

First of all, applications tend to change over time. As an example consider the constant development of new video codecs that we discussed in Section 3.6. It is important that accelerators do not slow down software development by imposing constraints on what can be accelerated. After an accelerator has been optimized for a certain computation and fabricated as an ASIC, it won't be able to adapt to changing computational demands.

Moreover, applications running on different datacenters might have different computational requirements. This would lead to different implementations of the same domain specific accelerator depending on the type of computation and inputs that the different cloud providers observe in production.

Finally, the techniques presented in the previous section – especially the interconnect synthesis – greatly increase the heterogeneity of the system. This affects physical implementation, e.g. more complex place and route, as well as verification of each different accelerator configuration.

In this chapter, we are going to evaluate an alternative homogeneous Coarse Grain Reconfigurable Architecture (CGRA) for acceleration of analytical query processing. We compare and contrast the Q100 class of accelerators with our alternative design, that we will refer to as the homogeneous design. Despite the latter being preferable given the increased programmability, we found that the performance of the two architectures is comparable.

We explain this counterintuitive result showing that the heterogeneity in Q100 has adverse effects. While each hardware module is individually efficient relative to a software implementation, low usage of each component causes performance loss. Root causes include a mismatch between the available modules and the resources needed by different query plans or congestion of the NoC that routes data between modules. In contrast, in the homogeneous system, each Processing Element (or PE) has equal computational capabilities, reducing mismatches between available and required resources at runtime. For the same reason, communication generally happens between neighbors making a resource conscious circuit switched network a viable alternative to the packet switched network of the Q100. The increased usage in the homogeneous design recoups the power and area overheads of the homogeneous PE relative to the Q100 modules. We find that both designs perform comparably given an area/power budget, with the homogeneous design outperforming the Q100 when this budget is small.

For this study, we have extended the SQL compiler for the Q100 with a new backend targeting the homogeneous design. The benefits of a full software infrastructure are twofold: 1) it demonstrates that developing software support for these custom architectures is feasible, and 2) it establishes a controlled comparison between the two architectures, minimally influenced by software differences. The compiler for the two differs only as strictly necessary in the backend. The frontend – importantly including the query planner – is the same. Since we are mostly interested in the inefficiencies of specialization, we do not measure the benefits of programmability, for example how a homogeneous architecture could support additional operations and increase application coverage. We limit our analysis to the operations supported by the Q100 on both architectures.

Software complexity increases in the homogeneous case due to the increased degrees of freedom when scheduling operations. We implemented a greedy scheduler to demonstrate a viable solution, but highlight a big performance gap that more sophisticated heuristics could reduce. This would make the homogeneous designs performance far superior than that of the Q100.

While previous work has contended with the relationship between programmability and efficiency [Hameed *et al.*, 2010; Nowatzki *et al.*, 2016; Qadeer *et al.*, 2013], 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 in depth analysis of accelerators for a computational domain that highlights pure performance degradation due to "excessive" specialization.

5.1 Motivation to De-Specialize

We have identified a number of ways in which the Q100 does not operate with high efficiency.

Despite careful tuning of Q100 modules, utilization is low. While the Q100 design compares favorably to software implementations, we find that the runtime utilization of the Q100 modules is low. Figure 5.1 shows the average amount of Q100 tile area idle for each TPC-H query. Since a Q100 design can be configured with arbitrary tile mixes we show here a set of pareto-optimal designs, hence the error bars. The reason for this low utilization can be found in variability between SQL queries. Table 5.1 further classifies the root causes of low utilization. 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 require fewer inputs than the hardware tile provides, leaving a fraction of the tile idle.

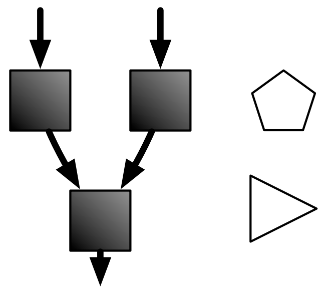
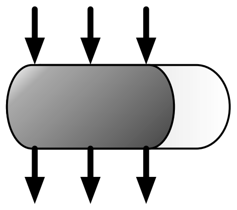
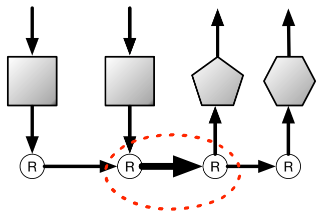
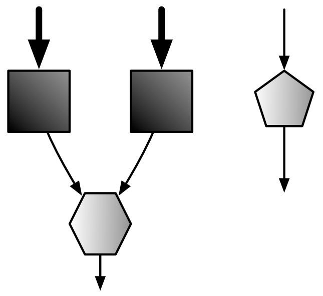
	**Idle tiles** occur when the query does not need some of the available tiles. This results from variation between queries or phases of a query. For example, analytical queries tend to filter inputs at the start and aggregate at the end, leaving aggregators idle at the start and filters idle at the end.
	**Input mismatch** happens when an operator processes fewer inputs than are provisioned in the Q100 tile. For example, a Q100 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 in Q22, which operates on a small number of columns from each table.
	When a link in the interconnection is oversubscribed, 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 easily ripple out and degrade performance. Interconnect congestion tends to occur when there are many active tiles. Note in Figure 5.2 that 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 5.1: Four root causes of low computational resource utilization in the Q100. In the cartoons, darker shading indicates higher utilization, while arrow thickness indicates data volumes.

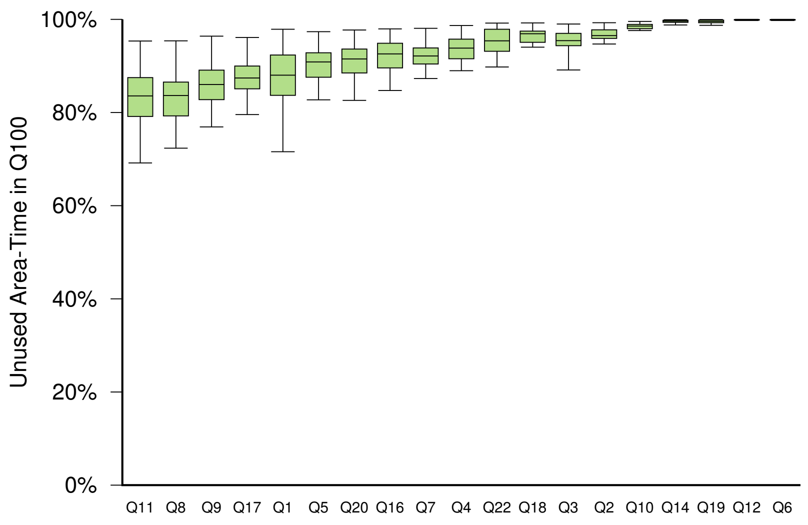


Figure 5.1: A large fraction of the Q100 area is idle when executing TPC-H queries.

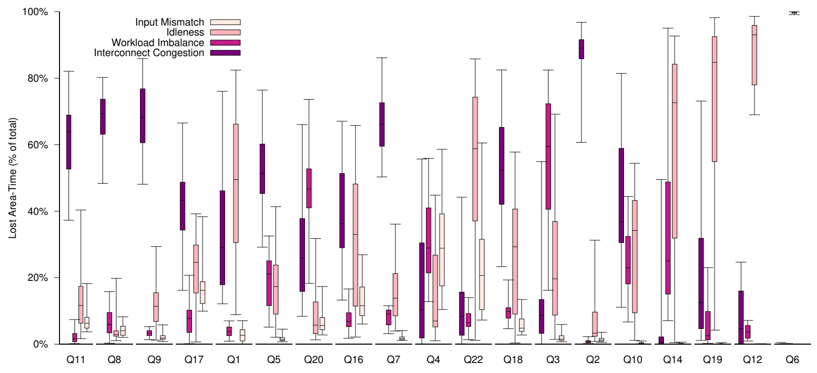


Figure 5.2: Area-Time product lost to different causes in the Q100.

In Figure 5.2 we can see all these factors at play on the various queries that compose TPC-H. Each query will exhibit a different mix of inefficiencies as it exercises the Q100 system in a different way. The big error bars are due to different Q100 tile mix and NoC combinations reacting in a different way to the same set of operations executed.

While clock and power gating can recoup most of the power associated with unused area, it still represents a costly resource that is not contributing to performance.

Despite careful tuning of the Q100 interconnect, it is expensive. Because the Q100 design assumes all-to-all communication amongst operator modules, a large fraction of the area goes to the interconnect that supports this communication (30-50% for most reasonable tile mixes). As shown in Table 5.1, links can become oversubscribed and therefore slowdown the entire computation. Interconnection specialization techniques that are presented in Chapter 4 can reduce interconnect size and/or conflicts. For this study we rely only on standard topologies for two reasons. First, these specialization techniques insert long bypass links which might affect place and route significantly. Since we are evaluating a large number of designs, placing and routing each one would be prohibitive. Second, these techniques tie a given Q100 implementation to a specific benchmark. Changes in the workload might call for a different NoC, hence a different ASIC tapeout with all the costs associated. These limitations are the main motivation for the development of the homogeneous design that we described at the beginning of this Chapter.

Some SQL operators are not supported on the Q100. Queries may require operators that are not supported by the Q100, most notably regular expression matching and non-equi-joins, such as hash joins. More generally, operations that result in random memory accesses are not supported because operator modules can access memory only via the *Reader* and *Writer* modules. Using the CPU as a fallback for such operations lowers efficiency, as intermediate data spills to and from memory on transfers between CPU and accelerator. The modules are restricted in other ways as well. For example, the *Aggregator* supports just four basic aggregation functions, SUM, COUNT, MIN, and MAX. Ultimately these specialization choices eat away at Amdahl's coverage and limit overall speedup.

There is an opportunity with common computational patterns Despite these drawbacks, the Q100 modules also suggest an opportunity. When implemented in hardware, relational algebra operators share similar datapaths. For example, aggregators contain an adder that is also shared with the ALU, the compare and swap logic on a sort module is similar to the one in a merge. This suggests that a compound module configurable to perform all operations might be a viable design strategy. This is the approach we take with the homogeneous system described in the next section.

5.2 Homogeneous Architecture

We have developed a homogeneous spatial, programmable dataflow architecture, designed to address the limitations of the Q100 listed above. While the basic processing element could easily be extended to support more operations, this design does not do so, as the coverage gains relative to Q100 are not readily quantified. Instead, we will weigh the merits of each architecture on precisely the same workload.

The homogeneous system, depicted in Figure 5.3 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 sit at the top of the array, and a row of *Writer* elements at the bottom. These units behave much like their Q100 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 5.3 shows the microarchitecture 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 output data tokens and two output control tokens. The compute core is configurable, executing essentially one instruction continuously until the device is re-programmed. 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.

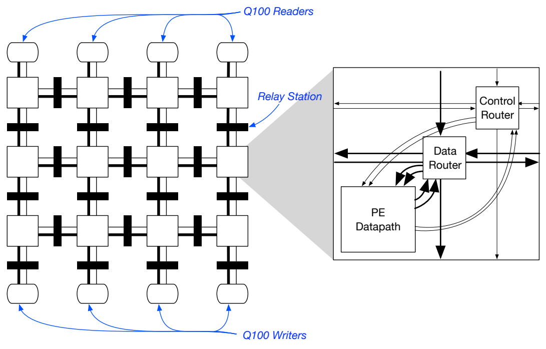


Figure 5.3: Homogeneous system architecture. Each processing element operates independently. Within each PE compute and communications modules are also independent. The memory subsystem is equivalent to the one used in the Q100.

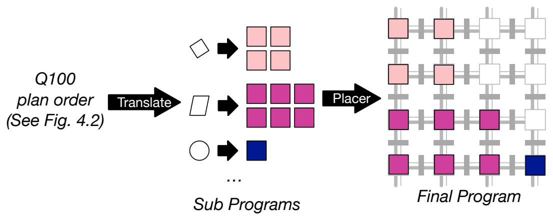


Figure 5.4: Compilation for our homogeneous design re-uses the Q100 frontend (Figure 4.2). After an ordering is found for the Q100 instructions each one is compiled into a graph of micro-ops. These are then mapped – over multiple time steps – to the processing elements in the system.

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 outputs until the design is reconfigured.

This design addresses the concerns listed in Section 5.1 as follows:

1. 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, or a combination of PEs. Furthermore, only the necessary number of PEs will be used for each operation while a Q100 module might be provisioned for a fixed number of input columns.

2. Since each PE is equivalent instructions will be scheduled over 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 area overhead and 3.8X power overhead on average).

3. PEs can be extended with more operations and therefore cover a larger fraction of a query plan compared to the Q100. While not quantified in this study, this would avoid pipeline breaks and consequent roundtrips between the CPU, memory, and the accelerator.

5.3 Homogeneous Compilation Backend

To ensure a controlled comparison between the Q100 and the homogeneous design, we re-use the front end of the Q100 compiler for both systems. The homogeneous backend, depicted in Figure 5.4 picks up from the point where the Q100 query plan has been sorted according to the Longest Job First heuristic (Section 4.1.3). Each operator that appears in a Q100 query plan is translated into a graph of instructions for the homogeneous PEs that we will call micro-ops. Instructions will continue to mean Q100 operations. Similarly to the Q100 instruction these micro-ops have 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 an 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 (Algorithm 1). 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 execution is terminated. 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 routing rules that must be added. 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 routing rules.

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. 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, possibly facilitating routing and reducing cost. After a predetermined number of mappings are evaluated, the least expensive valid mapping is selected and the PEs used marked as such.

Lastly, the homogeneous scheduler must test for deadlock, which can occur if there are reconvergent paths in the layout. Deadlock is avoided if there are more buffer slots in the shortest branch of the reconvergent path than there are hops in the longest branch. This condition must be tested after each instruction is scheduled, and if the test fails the instruction must be de-scheduled. In practice we find this happens rarely so it should not affect performance in a noticeable way.

We recognize that by considering only one instruction at a time, this algorithm may find only a 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 5.5.1 how much performance is left on the table due to sub-optimal mapping.

Algorithm 1 Homogeneous Micro-op Mapping

```
1: procedure MAP($inst$, $device$)
2:     $inst\_graph \leftarrow inst\_to\_graph(inst)$
3:     $best\_map \leftarrow null$
4:     $best\_cost \leftarrow Inf$
5:     for $i \leftarrow 1, max\_tries$ do
6:         $map \leftarrow null$
7:         if $i = 1$ then
8:             $map \leftarrow first\_map(inst\_graph, device)$
9:         else
10:            $map \leftarrow tweak\_map(best\_map, device)$
11:        end if
12:        if $map = null$ then
13:            break
14:        else
15:            $tries \leftarrow tries + 1$
16:            $cost \leftarrow compute\_cost(map, device)$
17:            if $cost < best\_cost$ then
18:                $best\_map \leftarrow map$
19:                $best\_cost \leftarrow cost$
20:            end if
21:        end if
22:    end for
23:    return $best\_map$
24: end procedure
```

5.4 Methodology

We developed an RTL implementation of the Q100 modules and the homogeneous PE. For the Q100 NoC, we use an open-source router implementation [Becker, 2018] while for the homogeneous NoC we implemented a circuit switched network router. NoCs for the Q100 are packet switched and use reverse credit flow [1] while the NoC for the homogeneous design uses a single circuit switched plane each for its data and control networks. While the different NoCs may at first appear to complicate the comparison, our objective is to pick the best network for each architecture. Because the two designs are so different, a single network design is liable to be a bad choice for one or the other. The Q100 assumes all-to-all connectivity between tiles. Since NoC congestion is already a significant source of inefficiency in the Q100, a circuit switched network such as the one in the homogeneous design would only exacerbate it. With a circuit switched network only a few circuits could be established before all of the physical links are occupied, thus preventing utilization of the available tiles. Similarly, a packet switched NoC is unnecessary for the homogeneous design as there is less chance of link contention since communication tends to happens between endpoints that are spatially close.

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.

[1] Reverse credit flow and additional buffer slots are used in the Q100 to avoid application deadlock when different modules share a link.

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

All performance numbers are obtained using an in house cycle-level simulator. The scheduler described in the previous sections is embedded in the simulator infrastructure and governs the execution of each query plan. For all homogeneous experiments the mapping algorithm described in Section 5.2 attempts at most 100 diverse mappings. Mapping of instructions is complex in the homogeneous system due to the many degrees of freedom when mapping an instruction to a set of PEs. Since our algorithm is not exhaustive we also consider an "ideal" homogeneous mapper. The ideal mapper only takes into account 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 [2]. While we could have performed a more thorough HW/SW codesign between the number of physical planes and the complexity of the mapping algorithm in the homogeneous architecture we feel this is outside the scope of this dissertation. Even with this rudimentary software support homogeneous performance is comparable to the Q100 and therefore proves our point.

All our tests are performed on TPC-H, the most widely used benchmark for query processing analytic workloads [Boncz *et al.*, 2014], 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 we are interested in evaluating each accelerator's merits.

[2]This upper bound is equivalent to assuming to have 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.

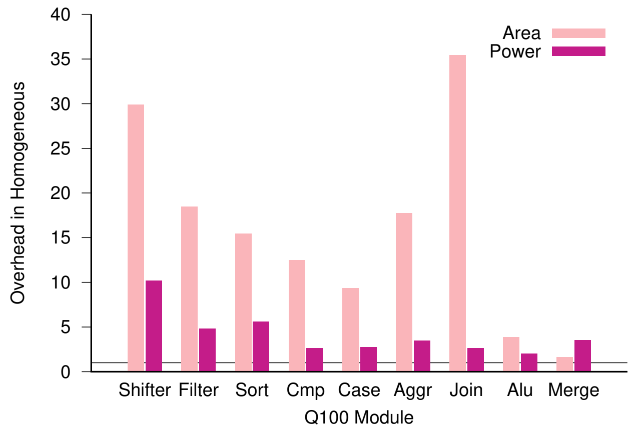


Figure 5.5: On a per operator basis there is a penalty associated with using a homogeneous PE instead of the Q100 module. Q100 modules are sorted from smallest on the left to largest on the right.

For all systems under consideration we assume the same memory subsystem. Columns are stored in memory and are streamed by a configurable number of *Reader* and *Writer* modules. These are capable to produce/consume a data token per clock cycle. For a Q100 system the number of input/output streams can be configured independently of the number of compute modules. On the other hand, for a homogeneous system *Readers* are attached to the top row of processing elements, one for each PE. Similarly, *Writers* are attached to the bottom row of PEs. Since all systems under consideration are configurable we assume that they will be paired with a powerful enough memory subsystem to drive them. Since it is always possible to scale down the size of either accelerators, and therefore their memory requirements, we assume that a balanced system can always be achieved. Therefore, the memory subsystem performance is not a variable accounted in this study.

5.5 Comparison of Q100 with Homogeneous Design

In this section we want to understand which accelerator design has better performance given an area/power budget and why. Figure 5.5 shows how, taken singularly, each Q100 module is significantly cheaper than a homogeneous PE. This is of course an intuitive result since each homogeneous processing element can be programmed to perform any tile operation.

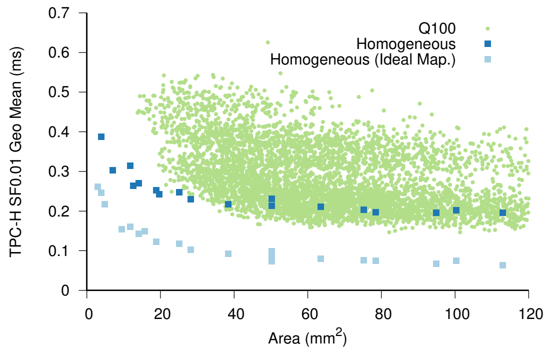


Figure 5.6: Performance/Area comparison of homogeneous and Q100 implementations

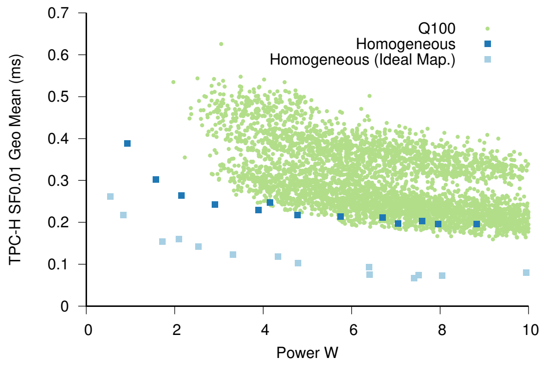


Figure 5.7: Performance/Power comparison of homogeneous and Q100 implementations.

Furthermore, the Q100 maximum operating frequency is higher (1100MHz) than that of the homogeneous design (950MHz).

However, when looking at the system holistically, performance is similar. The homogeneous design can recoup performance thanks to increased utilization and more efficient communication.

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 values to be reasonable ranges for an accelerator targeting the datacenter especially 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). Furthermore, 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 only considers a single accelerator die, other techniques that have already been presented [Magaki *et al.*, 2016] could be used to derive an ideal arrangement of different accelerator dies into a board and then combine these into a final system that further reduces Total Cost of Ownership (TCO).

From Figure 5.6 we can see that the homogeneous is comparable to Q100 Pareto optimal designs in terms of performance given an area budget. Furthermore, it outperforms the Q100 when that area budget is small. The advantage at low area is understandable since a Q100

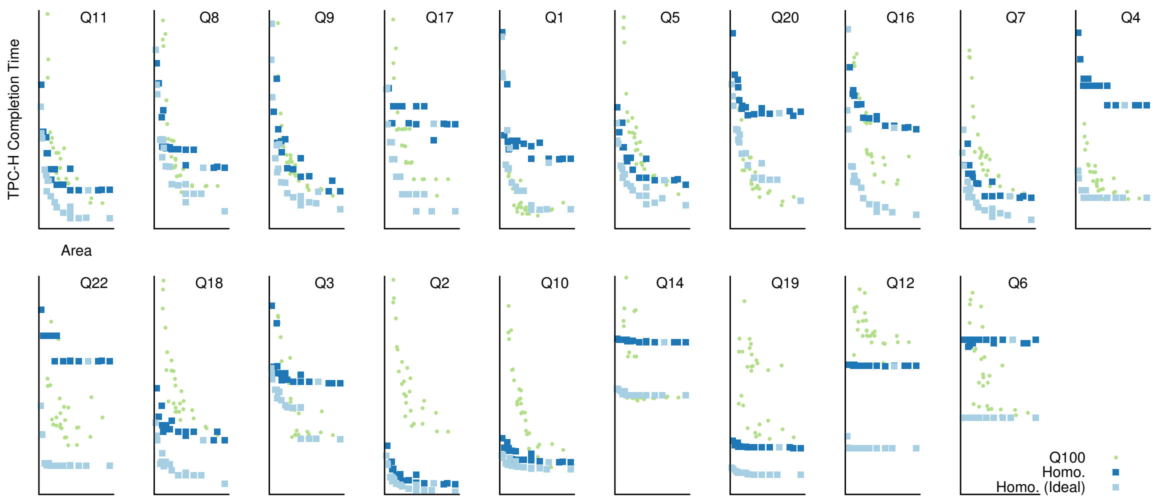


Figure 5.8: Performance gains of the homogeneous design depend on the query examined. In all cases using the ideal scheduler will cause the homogeneous system's performance to surpass that of the Q100.

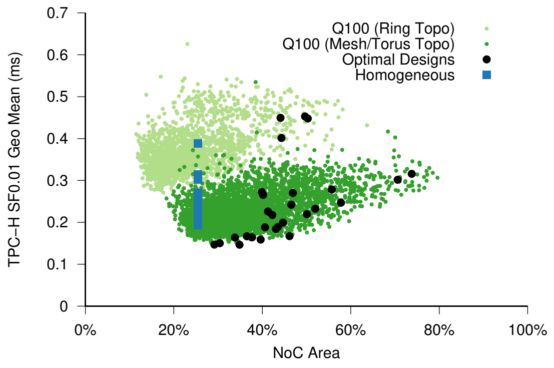
device will have to contain at least one of each tiles (plus the NoC to connect these) while a homogeneous design can shrink to a small size more gracefully as each PE is interchangeable.

In Figure 5.8 we can see a more detailed comparison on a query by query basis between different homogeneous design and Pareto optimal Q100s. Benefits depend on the query being considered.

Similar results are obtained when power is considered as a cost metric (Figure 5.7). Again homogeneous provides comparable results across the board and significant improvements for a small power budget.

In our analysis we want to understand where does the Q100 falls short. How can the homogeneous design perform just as well even though the PEs that comprise it are larger than the average Q100 module? How does the homogeneous system not incur in the same pitfalls?

Lower Cost of Communication In Figure 5.9 we can see that is possible to cluster all Q100 designs depending on whether they are using a ring, mesh, or torus interconnect. Devices that include a ring tend to be smaller but incur penalties in terms of running time.



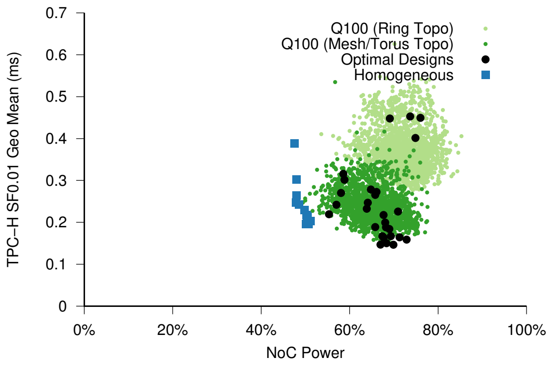


Figure 5.9: Fraction of area spent in the NoC. Pareto designs for the Q100 devote a larger fraction of their area to the NoC compared to homogeneous designs.

Figure 5.10: Homogeneous designs consume a smaller fraction of their power in the NoC.

While it is possible to construct a Q100 device in a wide range of NoC/Tile area ratios, designs on the Pareto frontier have a significant fraction of their area occupied by the NoC. In particular, almost all Pareto designs will invest a larger fraction of their area in the NoC than a homogeneous system. Remember that for homogeneous design this ratio is fixed since a single router is associated to each processing element.

In Figure 5.10 we can see that this reduction in NoC area in homogeneous also corresponds to a lower ratio of power spent in this component. Since the NoC can be considered as performing no "actual" work we can look at Figure 5.9 and Figure 5.10 and conclude that this is a factor in favor of homogeneous. This is not surprising if we consider that Q100 routers are more expensive and tend to support longer distance communication links.

Higher Utilization The homogeneous design is able to use a larger fraction of its area at any given time. Figure 5.11 shows that utilization is on average higher on a homogeneous device than on Q100. This result would be clear if we think back at the reasons behind low utilization in the Q100 (Section 5.1). The homogeneous design does not suffer from three out of four inefficiencies that affect the Q100 (depicted in Table 5.1). Since connections are all circuit switched no cycles can be lost due to the NoC, rather instruction might have to be scheduled in different temporal instruction if a valid routing is not found. Similarly, only

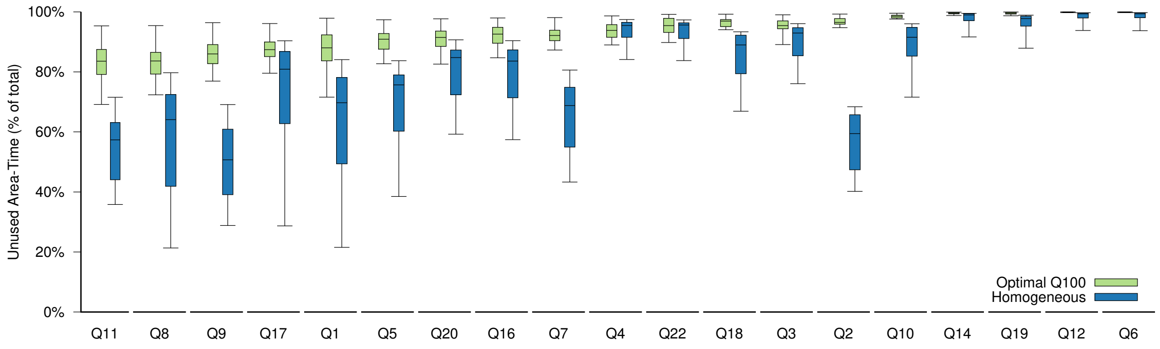


Figure 5.11: Unused Area-Time by both devices under exam. We can see that homogeneous makes a better use of the available resources.

routing restricts scheduling of instructions on homogeneous while on the Q100 an instruction can only be scheduled on its specific tile. There is no mismatch between an instruction number of inputs and its hardware implementation since instruction are compiled at runtime onto a set of PEs. Imbalance between instruction complexity affect homogeneous the same as this is a function of the workload rather than the architecture.

In conclusion, the homogeneous design is more efficient both in terms of area and power consumption, as its programmability and regularity makes the architecture adaptable to the workload.

5.5.1 Instruction Mapping in the Homogeneous Architecture

Our results highlight how the mapping algorithm is a critical component of the homogeneous system. If an ideal mapping of micro-ops to PE was attainable at every time step, utilization would be even higher, and performance of the system would greatly surpass that of Q100.

Figure 5.12 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 covered with software techniques rather than hardware. This is unlike the Q100 where limitations stem from the architecture of the system.

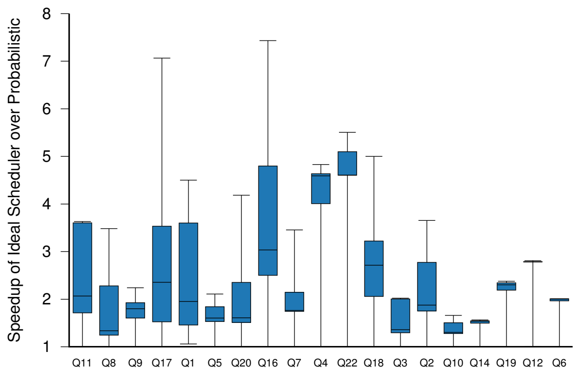


Figure 5.12: Comparison of our probabilistic algorithm with an ideal scheduler. Significant performance improvements could be obtained by better software scheduling.

5.6 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 [Hameed *et al.*, 2010]. 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 [Qadeer *et al.*, 2013]. Nowatzki et al. performed a study re-targeting proposed accelerators to a custom accelerator substrate they designed [Nowatzki *et al.*, 2016]. This work shows a significant cost (4x area and power) to pay for programmability and they do not analyze the sources of inefficiencies in the baseline accelerator as we do for the Q100.

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 in a hopefully modest penalty compared to an ASIC implementation. To the best of our knowledge we are the first to highlight a raw performance detriment due to "excessive" specialization. While the approach taken in the Q100 architecture is perfectly valid – the original paper [Wu *et al.*, 2014] shows significant improvements over software DBMSs – the orchestration of all the accelerator blocks coupled with the irregular nature of the application cause a detriment in performance.

There have been successful deployments of ASICs in the datacenter targeting Machine Learning inference [Jouppi *et al.*, 2017] or training [Dean *et al.*, 2018]. These are ideal workloads for acceleration due to the high compute per byte and their regular control flow regular structure. On the other hand, it has been shown that no single workload dominates in a datacenter and a long tail of datacenter workloads exist [Kanev *et al.*, 2015]. Among these workloads high compute per byte is the exception rather than the norm. These might include application 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 here. Therefore, we believe these application might perform more efficiently on a programmable non Von Neumann architecture such as the homogeneous design we presented here. Notice also how video transcoding at datacenter scale, which we presented in Chapter 3, also seems to have enough variability that current hardware accelerators cut corners and impose quality/performance tradeoffs.

5.7 Conclusions

To the best of our knowledge, this is the first work to highlight the dangers of over-specialization and analyze in details its causes on a complete system. In our analysis we found that the Q100 [Wu *et al.*, 2014] can be outperformed by a homogeneous programmable architecture. While the latter underperforms compared to the Q100 on any relational algebra operator in isolation, it is superior when considering entire queries. We blame the root cause of this counter-intuitive result on the Q100 excessive specialization that hinders its ability to adapt to an irregular workload such as analytical query processing. Our results suggests that there exists a tipping point where an application has sufficient variety in computational kernels and dependencies between them that a programmable architecture becomes preferable purely on a performance perspective.

This work supports our thesis by showing how application “regularity” can dictate the ideal level of specialization for an accelerator targeting such application. Counterintuitively, pushing specialization as much as possible does not necessarily provide the best results. We believe the results presented here motivate further research on the development of other non Von-Neumann architectures and their software stack.

Chapter 6

Conclusions

As more accelerators are deployed in datacenters to contend with the end of Dennard scaling, it is critical that their performance is optimized for the workload at hand. In this dissertation, we have shown the effectiveness of Design Space Exploration in deriving good accelerator designs given an area/power budget. Furthermore, this dissertation presents a thorough analysis on how to derive a benchmark to guide design space exploration for a production datacenter workload.

6.1 Summary of Contributions

The study presented in Chapter 3 highlights how video transcoding is a critical datacenter workload whose computational needs are outpacing performance growth of commercial general purpose CPUs (Figure 3.1). While vectorization can improve performance, it is not a long term solution (Section 3.5.2). Alternative solutions, including accelerators, are necessary to keep up with the growth of video uploads.

More importantly though, this study highlights how user facing cloud applications tend to have a large variety of use cases as well as input characteristics. This variety has to be captured in the benchmarks that will subsequently be used for design space exploration of accelerator solutions. The production use cases that we replicated in `vbench` – the `Popular`, `Upload`, `Live`, and `VOD` scenarios – all have different requirements. Therefore, accelerator solutions targeting this workload are unlikely to be applicable to all scenarios. Consider the GPU solutions that we have evaluated in Section 3.5.3. While a potential solution for the `Live` scenario[1], GPUs are not applicable to the `Popular` scenario, and impose tradeoffs for the `VOD` scenario. As another example consider the microarchitectural analysis that we performed in Section 3.5.1. This study shows the correlation between input videos and the microarchitectural profile of video transcoding. Correctly capturing the diversity of input video sequences becomes paramount in order to avoid incorrect assumptions about the workload that would sway optimization efforts.

The study in Chapter 4 shows how Design Space Exploration can significantly improve the performance of the Q100 class of accelerators given an area/power budget. In Section 4.1.4, we show how the number of tiles of each type in a Q100 instance can be tailored to the computational requirements of the SQL queries being executed. More interestingly though, we also show how it is possible to tailor the communication resources to the communication patterns between hardware modules of the accelerator. To this end, we present an algorithm that is able to automatically generate NoC topologies that are application-specific given a communication graph. We use this algorithm to derive an optimized NoC – more performant for a given area budget – for an instance of the Q100 accelerator. This study highlights how specializing communication in accelerators can be as important as specializing its computational blocks. Furthermore, we provide an algorithm that is domain agnostic and therefore could be applied to other accelerator designs. This study shows how application properties drive Design Space Exploration: tile mix as a function of input queries' operations, and NoC topology as a function of communication patterns.

Finally, Chapter 5 examines a critical aspect of accelerator design, their ideal level of specialization. Increased programmability is always desirable as it generally allows an accelerator to cover more computation, and decreases the likelihood of obsolescence. Intuitively, programmability comes at a cost and most of recent work on domain specific architectures presents programmability as contrary to performance. However, we show that when the application being accelerated has dependencies between its computational kernels that are a function of its inputs, programmability provides a performance benefit. For analytical

[1]Applicability of a solution is not only a function of their performance but also of their TCO that we did not compute.

query processing, the frequency of relational algebra operators and their data dependencies is a function of the input query being executed. In a highly specialized design – one that is not able to adapt to the variability of the computation – this leads to inefficiencies including low silicon utilization and a high cost of communication between hardware modules. In contrast, a programmable architecture can more easily adapt to varying computational demands and therefore outperform the more specialized design. We believe that these conclusions should extend to other application and therefore motivate the development of other non Von Neumann architectures, together with the software tools necessary to efficiently map applications unto them.

6.2 Future Work

This dissertation provides three contributions towards more effective design space exploration of accelerators for datacenters. However, there is much more work to do in order to significantly increase the energy efficiency of future datacenter systems.

Mapping the Datacenter Despite wide interest in the computer architecture community for machine learning workloads, there are many other datacenter applications that should receive commensurate interest. Previous work has shown how there is a long tail of applications in the datacenter that constitute a large fraction of all computation [Kanev *et al.*, 2015]. All these applications are poorly documented, in particular when we consider how datacenter applications can be sensitive to the variability of user inputs (Chapter 3). To the best of our knowledge, video transcoding [Lottarini *et al.*, 2018] and websearch [Ayers *et al.*, 2018] are the only workloads that received such attention. More benchmarks are necessary to fully engage the academic community and create common open source solutions. Solutions that hopefully could be applied across different cloud providers.

New Programmable Accelerator Architectures GPUs, FPGAs, and ML ASICs are now deployed in datacenters and can be considered success stories. However, we believe that to tackle the long tail of datacenter applications other solutions are necessary. Purely from a TCO perspective, ASICs – especially when targeting the latest, most expensive technology node – might not be the best solution [Magaki *et al.*, 2016; Khazraee *et al.*, 2017]. On the other hand, FPGAs leave some performance on the table [Falsafi *et al.*, 2017; Kuon and Rose, 2006; Vasilyev *et al.*, 2016].

We believe CGRAs constitute an interesting middle ground. Research efforts until now have been too scattered though. They should converge on few promising hardware prototypes as well as tacking what we believe is the hardest problem in this space: compilation to CGRAs.

Software Support for Accelerators Compiling to CGRAs is harder than normal software compilation. Compiler optimizations for software mostly deal with re-ordering instructions. On top of this time allocation of instructions, compilers for CGRAs have also to consider allocation in space. We encountered this problem when mapping Q100 workloads to our proposed CGRA architecture (Section 5.3). While previous work has tried to tackle compilation for CGRA architectures [Smith *et al.*, 2006] we still need definitive solutions in this space.

We believe that leveraging the recent popularity of Domain Specific Languages might simplify this task. Limiting the compilation to a domain specific language, together with selecting a good intermediate representation for the domain [Brain, 2017] – one which successfully abstracts away differences between accelerator implementations – will significantly reduce the compiler complexity and increase the quality of results.

Support Heterogeneity in the Datacenter All these heterogeneous computing resources that are now (or will be) deployed in datacenters have to be managed. Applications will generally prefer to be scheduled on their accelerator of choice, but when that is not possible, fallback to other architectures should be seamless and application performance should degrade gracefully. Efficient scheduling and virtualization of accelerator resources is necessary in order to achieve this goal [Gupta *et al.*, 2011]. Furthermore, when work is scheduled to an accelerator, we have to understand how this accelerator should be controlled. In case this control comes from a host CPU, how can we ensure that accelerators' performance does not degrade when such host CPU utilization is high [Zhu *et al.*, 2019]?

General Purpose Computing is still Relevant Efficiency in general purpose computing is still going to be critical, despite of the rise of accelerators. At least for the near future, we should expect the latest version of an application to target general purpose CPUs first. We have observed this phenomenon in the `Popular` scenario of `vbench` (Section 3.6.2). On the other hard, accelerators will relieve general purpose CPUs of the most compute heavy tasks. This could open up interesting research directions for developing datacenter specific general purpose CPUs. One possibility is to tackle what previous work has called the "datacenter tax" [Kanev *et al.*, 2015] by reducing the overhead of recurrent computation, e.g. memory allocation [Kanev *et al.*, 2017]. Another interesting direction is to use machine learning not only as an application but also to drive computer architecture design [Hashemi *et al.*, 2018].

There is no doubt that the computing landscape is going to become more and more heterogeneous in the near future. This opens up many exciting avenues for research as new systems will have to be developed to cope with the end of Dennard scaling and the slowdown of Moore's law. Similarly to what we presented in this dissertation, thorough benchmarking of applications at the warehouse scale, and the subsequent development of new architectures or accelerators will become a key component in increasing energy efficiency for the datacenter.

Bibliography

[Agrawal *et al.*, 2017] Sandeep R Agrawal, Sam Idicula, Arun Raghavan, Evangelos Vlachos, Venkatraman Govindaraju, Venkatanathan Varadarajan, Cagri Balkesen, Georgios Giannikis, Charlie Roth, Nipun Agarwal, and Eric Sedlar. A many-core architecture for in-memory data processing. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2017.

[Alvarez *et al.*, 2007] M. Alvarez, E. Salami, A. Ramirez, and M. Valero. HD-videobench: A benchmark for evaluating high definition digital video applications. In *IEEE International Symposium on Workload Characterization*, IISWC, 2007.

[Ayers *et al.*, 2018] G. Ayers, J. H. Ahn, C. Kozyrakis, and P. Ranganathan. Memory hierarchy for web search. In *IEEE International Symposium on High Performance Computer Architecture*, HPCA, 2018.

[Bawden, 2016] Tom Bawden. Global warming: Data centres to consume three times as much energy in next decade, experts warn, 2016.

[Becker, 2018] Daniel Becker. Efficient microarchitecture for network-on-chip routers. 07 2018.

[Benini and De Micheli, 2002] Luca Benini and Giovanni De Micheli. Networks on chips: A new SoC paradigm. *IEEE Trans. on Computers*, January 2002.

[Bertozzi *et al.*, 2005] Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna Tamhankar, Stergios Stergiou, Luca Benini, and Giovanni De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. Parallel Distrib. Syst.*, 2005.

[Bienia, 2011] Christian Bienia. *Benchmarking Modern Multiprocessors.* PhD thesis, Princeton University, 2011.

[Blender Foundation, 2002] Blender Foundation. *Blender - a 3D modelling and rendering package*, 2002.

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

[Bohr, 2007] M. Bohr. A 30 year retrospective on dennard's mosfet scaling paper. *IEEE Solid-State Circuits Society Newsletter*, 2007.

[Boncz *et al.*, 2014] Peter Boncz, Thomas Neumann, and Orri Erling. *TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark*, pages 61–76. Springer International Publishing, 2014.

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

[Borkar and Chien, 2011] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54(5), May 2011.

[Brain, 2017] Google Brain. Xla - tensorflow, compiled, 2017.

[Buyya *et al.*, 2008] Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali. *Content Delivery Networks.* Springer Publishing Company, 1st edition, 2008.

[Cao *et al.*, 2017] Bingyi Cao, Kenneth A. Ross, Stephen A. Edwards, and Martha A. Kim. Deadlock-free joins in db-mesh, an asynchronous systolic array accelerator. In *Proceedings of the International Workshop on Data Management on New Hardware*, DAMON, 2017.

[Caulfield *et al.*, 2016] 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. A cloud-scale acceleration architecture. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2016.

[Cha *et al.*, 2009] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Transactions on Networking (TON)*, 17(5), 2009.

[Chen *et al.*, 2016a] Chao Chen, Mohammad Izadi, and Anil Kokaram. A perceptual quality metric for videos distorted by spatially correlated noise. In *Proceedings of the ACM Multimedia Conference*, 2016.

[Chen *et al.*, 2016b] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture*, ISCA, 2016.

[Chung *et al.*, 2013] Eric S. Chung, John D. Davis, and Jaewon Lee. Linqits: Big data on little clients. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2013.

[Cock *et al.*, 2016] Jan De Cock, Aditya Mavlankar, Anush Moorthy, and Anne M. Aaron. A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications. In *Applications of Digital Image Processing XXXIX*, 2016.

[Corbal *et al.*, 1999] Jesus Corbal, Roger Espasa, and Mateo Valero. MOM: A matrix SIMD instruction set architecture for multimedia applications. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, SC, 1999.

[Corp., 2017] Intel Corp. Intel Quick Sync Video. `https://www.intel.com/content/www/us/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html`, 2017.

[CreativeCommons.org, 2007] CreativeCommons.org. Creative commons attribution 3.0 license. `https://creativecommons.org/licenses/by/3.0/legalcode`, 2007.

[Dally and Seitz, 1987] William J. Dally and C.L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Comput.*, 36(5):547–553, May 1987.

[Dally and Towles, 2001] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the Annual Design Automation Conference*, DAC, 2001.

[Dally and Towles, 2003] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[Danowitz *et al.*, 2012] Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz. Cpu db: Recording microprocessor history. *Commun. ACM*, 55(4), April 2012.

[Dean *et al.*, 2018] J. Dean, D. Patterson, and C. Young. A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, 38(2), Mar 2018.

[Dennard *et al.*, 1974] Robert H. Dennard, Fritz H. Gaensslen, Hwa nien Yu, V. Leo Rideout, Ernest Bassous, Andre, and R. Leblanc. Design of ion-implanted mosfets with very small physical dimensions. *IEEE J. Solid-State Circuits*, 1974.

[Engineering and Blog, 2016] Youtube Engineering and Developers Blog. A look into YouTube's video file anatomy. `https://youtube-eng.googleblog.com/2016/04/a-look-into-youtubes-video-file-anatomy.html`, 2016.

[Esmaeilzadeh *et al.*, 2011] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2011.

[facebook, 2016] Facebook Inc. third quarter 2016 earnings call (transcript). `https://seekingalpha.com/article/4018524-facebook-fb-q3-2016-results-earnings-call-transcript`, 2016.

[Falsafi *et al.*, 2017] Babak Falsafi, Bill Dally, Desh Singh, Derek Chiou, Joshua J. Yi, and Resit Sendag. Fpgas versus gpus in data centers. *IEEE Micro*, 37(1), January 2017.

[Ferdman *et al.*, 2012] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2012.

[Firestone *et al.*, 2018] 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. Azure accelerated networking: Smartnics in the public cloud. In *USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2018.

[Fouladi *et al.*, 2017] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2017.

[Foundation, 2017] Mozilla & The Xiph.Org Foundation. Progress in the alliance for open media. `https://people.xiph.org/~tterribe/pubs/lca2017/aom.pdf`, 2017.

[Fowers *et al.*, 2018] 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. A configurable cloud-scale dnn processor for real-time ai. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2018.

[Fujiki *et al.*, 2018] Daichi Fujiki, Aran Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. Genax: A genome sequencing accelerator. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2018.

[Glass and Ni, 1992] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 1992.

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

[Gonzalez and Horowitz, 1996] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9), 1996.

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

[Gupta *et al.*, 2011] Vishakha Gupta, Karsten Schwan, Niraj Tolia, Vanish Talwar, and Parthasarathy Ranganathan. Pegasus: Coordinated scheduling for virtualized accelerator-based systems. In *USENIX Annual Technical Conference*, USENIX ATC, 2011.

[Ham *et al.*, 2016] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2016.

[Hameed *et al.*, 2010] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2010.

[Hashemi *et al.*, 2018] Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. In *International Conference on Machine Learning*, ICML, 2018.

[Hauswald *et al.*, 2015] Johann Hauswald, Michael A. Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G. Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, and Jason Mars. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2015.

[Hennessy and Patterson, 2017] John L. Hennessy and David A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6th edition, 2017.

[Henning, 2006] John L Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4), 2006.

[Ho and Pinkston, 2003] Wai Hong Ho and T. M. Pinkston. A methodology for designing efficient on-chip interconnects on well-behaved communication patterns. In *IEEE International Symposium on High Performance Computer Architecture*, HPCA, 2003.

[Horowitz, 2014] M. Horowitz. Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, ISSCC, Feb 2014.

[Huang *et al.*, 2017] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito, IV, Xifan Yan, Maxim Bykov, Chuen Liang, Mohit Talwar, Abhishek Mathur, Sachin Kulkarni, Matthew Burke, and Wyatt Lloyd. SVE: Distributed video processing at Facebook scale. In *Proceedings of the Symposium on Operating Systems Principles*, SOSP, 2017.

[Insights, 2015] Tubular Insights. 500 hours of video uploaded to YouTube every minute. `http://tubularinsights.com/hours-minute-uploaded-youtube/`, 2015.

[Jouppi *et al.*, 2017] 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. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2017.

[Jun *et al.*, 2015] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. Bluedbm: An appliance for big data analytics. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2015.

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

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

[Khazraee *et al.*, 2017] Moein Khazraee, Lu Zhang, Luis Vega, and Michael Bedford Taylor. Moonwalk: Nre optimization in asic clouds. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2017.

[Kocberber *et al.*, 2013] Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim, and Parthasarathy Ranganathan. Meet the walkers: Accelerating index traversals for in-memory databases. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2013.

[Koibuchi *et al.*, 2012] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, D. Frank Hsu, and Henri Casanova. A case for random shortcut topologies for HPC interconnects. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2012.

[Kokaram *et al.*, 2012] Anil Kokaram, Damien Kelly, Hugh Denman, and Andrew Crawford. Measuring noise correlation for improved video denoising. In *IEEE International Conference on Image Processing*, 2012.

[Kung and Lehman, 1980] H. T. Kung and Philip L. Lehman. Systolic (vlsi) arrays for relational database operations. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD, 1980.

[Kuon and Rose, 2006] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. In *Proceedings of the international ACM/SIGDA symposium on Field programmable gate arrays*, FPGA, 2006.

[Li *et al.*, 2016] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward a practical perceptual video quality metric. `http://techblog.netflix.com/2016/06/toward-practical-perceptual-video.html`, 2016.

[Lim *et al.*, 2013] Kevin Lim, David Meisner, Ali G. Saidi, Parthasarathy Ranganathan, and Thomas F. Wenisch. Thin servers with smart pipes: Designing soc accelerators for memcached. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2013.

[Lottarini *et al.*, 2017] Andrea Lottarini, Stephen A. Edwards, Kenneth A. Ross, and Martha A. Kim. Network synthesis for database processing units. In *Proceedings of the Annual Design Automation Conference*, DAC, 2017.

[Lottarini *et al.*, 2018] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. vbench: Benchmarking video transcoding in the cloud. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2018.

[Maas *et al.*, 2018] Martin Maas, Krste Asanović, and John Kubiatowicz. A hardware accelerator for tracing garbage collection. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2018.

[Magaki *et al.*, 2016] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. Asic clouds: Specializing the datacenter. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2016.

[Marpe *et al.*, 2003] Detlev Marpe, Heiko Schwarz, and Thomas Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 13(7), 2003.

[McAfee and Brynjolfsson, 2012] A. McAfee and E. Brynjolfsson. Big Data: The management revolution. *Harvard Business Review*, October 2012.

[Mercer Moss *et al.*, 2016] Felix Mercer Moss, Ke Wang, Fan Zhang, Roland Baddeley, and David R. Bull. On the optimal presentation duration for subjective video quality assessment. *IEEE Transactions on Circuits and Sysystems for Video Technology*, 26(11), 2016.

[Mirsky and DeHon, 1996] Ethan Mirsky and Andre DeHon. Matrix: a reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*, FCCM, 1996.

[Moore, 2006] G. E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 2006.

[Mukherjee *et al.*, 2013] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jinghing Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. The latest open-source video codec VP9 - an overview and preliminary results. In *Picture Coding Symposium*, PCS, 2013.

[Murali *et al.*, 2006] Srinivasan Murali, Paolo Meloni, Federico Angiolini, David Atienza, Salvatore Carta, Luca Benini, Giovanni De Micheli, and Luigi Raffo. Designing application-specific networks on chips with floorplan information. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, ICCAD, 2006.

[Nowatzki *et al.*, 2013] Tony Nowatzki, Michael Sartin-Tarm, Lorenzo De Carli, Karthikeyan Sankaralingam, Cristian Estan, and Behnam Robatmili. A general constraint-centric scheduling framework for spatial architectures. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI, 2013.

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

[Nowatzki *et al.*, 2017] Tony Nowatzki, Vinay Gangadhar, Newsha Ardalani, and Karthikeyan Sankaralingam. Stream-dataflow acceleration. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2017.

[Ogras and Marculescu, 2006] U.Y. Ogras and R. Marculescu. It's a small world after all: NoC performance optimization via long-range link insertion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, July 2006.

[Ouyang *et al.*, 2016] Jian Ouyang, Wei Qi, Yong Wang, YichenTu, Jing Wang, and Bowen Jia. SDA: Software-defined accelerator for general-purpose distributed big data analysis system. In *IEEE Hot Chips Symposium*, HCS, 2016.

[Papamichael and Hoe, 2012] Michael K. Papamichael and James C. Hoe. CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs. In *Proceedings of the international ACM/SIGDA symposium on Field programmable gate arrays*, FPGA, 2012.

[Parashar *et al.*, 2013] 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. Triggered instructions: A control paradigm for spatially-programmed architectures. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2013.

[Pinto *et al.*, 2002] Alessandro Pinto, Luca P. Carloni, and Alberto L. Sangiovanni-Vincentelli. Constraint-driven communication synthesis. In *Proceedings of the Annual Design Automation Conference*, DAC, 2002.

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

[project, 2017] The FFmpeg project. Encode/H.264. `https://trac.ffmpeg.org/wiki/Encode/H.264`, 2017.

[Putnam, 2017] Andrew Putnam. Fpgas in the datacenter: Combining the worlds of hardware and software development. In *Proceedings of the on Great Lakes Symposium on VLSI*, GLSVLSI, 2017.

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

[Rabbani and Jones, 1991] Majid Rabbani and Paul W. Jones. *Digital Image Compression Techniques.* Society of Photo-Optical Instrumentation Engineers (SPIE), 1st edition, 1991.

[Rafetseder *et al.*, 2011] Albert Rafetseder, Florian Metzger, David Stezenbach, and Kurt Tutschku. Exploring YouTube's content distribution network through distributed application-layer measurements: A first view. In *Proceedings of the International Workshop on Modeling, Analysis, and Control of Complex Networks*, CNET, 2011.

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

[Reagen *et al.*, 2016] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the International Symposium on Computer Architecture*, ISCA, 2016.

[Repetti *et al.*, 2017] Thomas J. Repetti, João P. Cerqueira, Martha A. Kim, and Mingoo Seok. Pipelining a triggered processing element. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2017.

[Sandvine Intelligent Broadband Networks, 2016] Sandvine Intelligent Broadband Networks. 2016 global internet phenomena: Latin America & North America. `https://www.sandvine.com/trends/global-internet-phenomena/`, 2016.

[Sankaralingam *et al.*, 2003] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W Keckler, and Charles R Moore. Exploiting ilp, tlp, and dlp with the polymorphous trips architecture. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA. IEEE, 2003.

[Simonite, 2016] Tom Simonite. Moore's daw is dead. now what? *MIT Technology Review*, 2016.

[Smith *et al.*, 2006] A. Smith, J. Burrill, J. Gibson, B. Maher, N. Nethercote, B. Yoder, D. Burger, and K. S. McKinley. Compiling for edge architectures. In *International Symposium on Code Generation and Optimization*, CGO, 2006.

[Srinivasan *et al.*, 2006] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear-programming-based techniques for synthesis of network-on-chip architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, April 2006.

[Sullivan *et al.*, 2012] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 2012.

[Sutton *et al.*, 1998] Roy A Sutton, Vason P Srini, and Jan M Rabaey. A multiprocessor dsp system using paddi-2. In *Design Automation Conference, 1998. Proceedings.* IEEE, 1998.

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

[Tang *et al.*, 2017] Linpeng Tang, Qi Huang, Amit Puntambekar, Ymir Vigfusson, Wyatt Lloyd, and Kai Li. Popularity prediction of Facebook videos for higher quality streaming. In *USENIX Annual Technical Conference*, 2017.

[Taylor *et al.*, 2002] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE micro*, 22(2), 2002.

[Taylor, 2012] Michael B. Taylor. Is dark silicon useful?: Harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the Annual Design Automation Conference*, DAC, 2012.

[Teubner and Mueller, 2011] Jens Teubner and Rene Mueller. How soccer players would do stream joins. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD, 2011.

[Vasilyev *et al.*, 2016] A. Vasilyev, N. Bhagdikar, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz. Evaluating programmable architectures for imaging and vision applications. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, 2016.

[Venkatesh *et al.*, 2010] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2010.

[Wang *et al.*, 2004] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 2004.

[Wang *et al.*, 2014] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent Zhan, Xiaona Li, and Bizhu Qiu. BigDataBench: A big data benchmark suite from internet services. In *IEEE International Symposium on High Performance Computer Architecture*, HPCA, 2014.

[Wiegand *et al.*, 2003] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), 2003.

[Wilhelmsen *et al.*, 2014] Martin Alexander Wilhelmsen, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Asgeir Mortensen, Ragnar Langseth, Carsten Griwodz, and Pål

Halvorsen. Using a commodity hardware video encoder for interactive video streaming. In *IEEE International Symposium on Multimedia*, ISM, 2014.

[Woods *et al.*, 2014] Louis Woods, Zsolt István, and Gustavo Alonso. Ibex: An intelligent storage engine with support for advanced SQL offloading. *Proc. VLDB Endow.*, 2014.

[Wu *et al.*, 2013] Lisa Wu, Raymond J. Barker, Martha A. Kim, and Kenneth A. Ross. Navigating big data with high-throughput, energy-efficient data partitioning. In *Proceedings of the Annual International Symposium on Computer Architecture*, ISCA, 2013.

[Wu *et al.*, 2014] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. 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, 2014.

[Wu *et al.*, 2015] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. The Q100 database processing unit. *IEEE Micro*, May 2015.

[Xin *et al.*, 2005] Jun Xin, Chia-Wen Lin, and Ming-Ting Sun. Digital video transcoding. *Proceedings of the IEEE*, 93(1), 2005.

[Xiph.org, 2016] Xiph.org. Derf's test media collection, 2016.

[Yasin, 2014] Ahmad Yasin. A top-down method for performance analysis and counters architecture. In *IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS, 2014.

[Yu *et al.*, 2008] 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. Asap: An asynchronous array of simple processors. *IEEE Journal of Solid-State Circuits*, 43(3), 2008.

[Zhang *et al.*, 2017] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T. Kandemir, Ravi Iyer, and Chita R. Das. Race-to-sleep + content caching + display caching: A recipe for energy-efficient video streaming on handhelds. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2017.

[Zhu *et al.*, 2015] Yuhao Zhu, Daniel Richins, Matthew Halpern, and Vijay Janapa Reddi. Microarchitectural implications of event-driven server-side web applications. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2015.

[Zhu *et al.*, 2019] Haishan Zhu, David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Mattan Erez. Kelp: Qos for accelerated machine learning systems. In *IEEE International Symposium on High Performance Computer Architecture*, HPCA, 2019.