My research focuses on opportunities at the intersection between computer architecture and datacenter applications. This is a rich area to explore today for two reasons. First, our industry is driven by deploying new, often computationally-expensive features in applications, and while Moore's law continues to give us more transistors, the end of Dennard scaling means we can't benefit from them as we used to [7]. Second, fundamental ratios in our systems are changing: cluster networks are providing higher bandwidth and low latency, storage devices are following a similar trend, and energy usage is now a primary concern at scale. These factors are motivating us to rethink our abstractions; old assumptions about common cases are holding us back. My work shows there are big opportunities in specializing for key applications and use cases, both in hardware and in software.

Over the 15 years I've spent in industry and academia, I've designed both hardware and low-level software, and worked in computer architecture, storage, networking and high-performance computing. My goal now is to do research and advanced development for faster, more efficient large-scale systems. I want to be in an environment with access to production-scale profiling data and real-world datasets, where there is a clear path from prototype to production for the right idea.

Prior work

Latency-tolerant distributed shared memory. My thesis work, Grappa [13], achieved an order of magnitude more performance than existing systems for some data-intensive applications by rethinking two old ideas in the context of modern systems and workloads: software distributed shared memory and latency tolerance.

Grappa provides a distributed shared memory abstraction tuned for data-intensive applications. It is inspired by the Tera MTA [2, 1], a supercomputer providing large-scale multithreading in hardware. Like the MTA, instead of relying on *locality* to reduce the cost of memory accesses, Grappa depends on *parallelism* to keep processor resources busy and hide the high cost of inter-node communication. Grappa adopts the shared-memory, fine-grained parallel programming mindset from the MTA. To support fast fine-grained messaging on commodity networks, Grappa includes an overlay network that transparently combines small messages together into larger physical network packets. The system is built in user-space, utilizing modern programming language features to provide the global address space abstraction.

While Grappa has been a technical success, I have also worked to make it a success of technology transfer. I've done this in four directions. The first (and easiest) was transferring the technology out of our lab to other academics: collaborators at other institutions have published multiple papers using Grappa. The second direction was a startup: my advisors and I formed a company named Konyac to explore applications of the Grappa ideas in the analytics space. We spent a year gathering feedback, developing a minimum viable product, and deploying it in early engagements, but ultimately decided not to proceed. After that, we released the code on GitHub and now have a small but active user base reporting bugs and contributing pull requests. And finally, I have been doing consulting work with companies to integrate Grappa with their proprietary software.

Specialization and approximate computing. Colleagues in my research group pioneered architectural support for the area of *approximate computing*, the idea that computer systems need not waste time and energy providing perfect correctness to applications that don't require it. By relaxing abstractions and exposing the error properties of hardware to applications that can tolerate them, we can gain significant efficiency. I have worked on a few projects in this space.

The first project explored *approximate storage* [16], which exposed the reliability properties of emerging memory technologies such as phase-change memory (PCM) for storing approximate data such as media and sensor data. We explored two techniques: one takes advantage of the "analog" way in which data is stored in PCM's underlying material to issue faster writes for approximate data. The other extends device lifetime by allowing reuse of failed bits for storing approximate data; it adapts the memories' existing error-correcting codes to maximize accuracy when bits wear out.

The second project explored design tradeoffs for an accelerator that uses neural networks to do function-level approximation. This style of accelerator was first proposed as a tightly-coupled execution unit within the CPU's pipeline [8]. Our work instead implemented the accelerator in the FPGA fabric of a commercially available programmable system-on-chip. We used ideas from my work on latency tolerance to overcome the latency of communication between the hard CPU core and the accelerator, giving an average speedup of 3.78x and average energy savings of 2.77x on real hardware. This compares favorably with the paper on which our approach was based, which predicted 2.3x speedup and 3.0x energy savings in the simulated tightly-coupled design.

I have also explored specialization at Google, working on microarchitecture, FPGA, and PCB design for some accelerator hardware.

Future work

What should a near-future warehouse-scale computer look like? I don't think we need radical new technology—my work has shown that there are big opportunities lying at the hardware-software interface in clusters today. With the right abstractions, we can make use of technology that will be deployed in the next few years to build systems that provide significant improvements in throughput and efficiency. Here are ideas for four components of such a computer.

Memory: using emerging non-volatile devices to achieve petabyte-scale working sets. In-memory analytics systems like Grappa and Spark [17] have significant performance benefits compared to those like Hadoop/MapReduce [6] that compute from secondary storage. DRAM's fast fine-grained random-access abilities may yield faster answers to questions about the data, but DRAM is an expensive and power-hungry limited resource in most clusters: processing may be limited to a subset of the data, potentially missing key insights.

Emerging memory technologies are making it practical to build a computer with a petabyte-scale main memory. Devices like memristors, phase-change memory (PCM), and Intel's 3D XPoint promise significantly better bit density, energy usage, and cost per bit than DRAM. (Note that while these technologies are non-volatile, exploiting that property is not the goal of this project.) While we can build a cluster with a petabyte of DRAM today (e.g., 10K nodes of 128GB/node), the scale required leads to significant overhead and complexity for fault-tolerance, combined with high network latency, energy usage, and cost. In a few years, we may be able to build a workgroup-sized cluster with the same total capacity, and a significant fraction of the total random-access bandwidth.

There are many challenges involved in building such a machine. Cell wearout is an issue, but has been extensively studied; there are many solutions to choose from. Access latency is a bigger issue, but the techniques I explored to tolerate network latency in Grappa can be re-purposed if the memory can be designed to handle enough in-flight operations. Address translation is an interesting problem: most existing processors only implement 48 bits of physical address space (256 TB), and other on-chip and in-OS structures are not sized appropriately for very large memories. It will likely be necessary to implement a separate address space as I did in Grappa, and do address translation in the memory controller. I explored some of these issues in a speculative workshop paper [12].

I would explore this first in simulation, building a simple simulator to capture first-order effects (perhaps a Pin tool). Once practical memories are available, I would build a small prototype with an FPGA controller to use in calibrating the model. If the benefit looked significant, I would then push to build a full-scale system.

Communication: Better interfaces for small messages. Network bandwidths continue to increase, while CPU clock rates and core counts are remaining relatively constant. This means that we have fewer and fewer cycles to process each packet. RDMA networks like InfiniBand try to help with this by offloading some of the *computation* involved in guaranteeing reliability and protection to the network card, but the *state* involved must still be accessed, often requiring the movement of multiple cache lines, as well as multiple round trips over the PCI Express bus, to send a single packet.

In Grappa, I aggregate multiple messages into a single packet to amortize this overhead, but that also requires multiple memory operations per messages, adds significant latency, and requires computation on the remote end of the connection. With the right core-to-network-card interface, the same process could be done much more efficiently, with a latency cost orders of magnitude lower than my software implementation. Furthermore, we could aggregate native RDMA operations, allowing the remote network card to execute the operations directly, and keeping the remote CPU out of the mix.

This work would be a natural outgrowth of Grappa; my first step would be to port the aggregation layer to run in a programmable fabric like Catapult [15] and perform an evaluation similar to that in the Grappa paper. In this process, I expect many questions to arise about the proper way to interact with the OS, caches, and other network traffic flows, as in related work like FlexNIC [10], IX [3], and Arrakis [14]; exploring these questions in the context of applications like Grappa may yield further insights.

Better programmability through higher-level atomic RDMA operations. Current RDMA networks support a limited set of primitives: non-atomic reads and writes, along with atomic fetch-and-increment and and compare-and-swap on single words. Programmers must use these low-level concurrency primitives to build higher-level ones: queues, hash tables, and linked data structures like trees and graphs. It is certainly possible to do this with existing primitives,

but with network latencies in the microsecond range, a simple remote queue insert operation built in a lock-free style using CAS operations would have a window for conflicts that is hundreds of thousands of processor instructions wide. These restrictions limit the possible benefit of using RDMA [9].

Many of these restrictions would be avoided if we could send bundles of dependent, conditional RDMA operations. For instance, the lock-free queue insert could be performed by bundling a data write with the CASes to insert it in the queue. However, allowing arbitrary operations in bundles could lead to problems with progress and resource starvation; choosing the right limitations, and the right way of expressing these bundles, will be a significant component of this project.

I would start by identifying some target applications to drive the design; for instance, key-value stores, parameter servers for machine learning, and distributed computation tools like Spark and Hadoop. These high level operations are likely to be expressed in the current implementations RPCs; profiling should allow me to identify worthy operations. I would measure performance first by emulating the RDMA protocol in software on a spare core, and then implementing it on a programmable network interface (from one of the RDMA network vendors, if possible, or on a programmable fabric like Catapult [15]).

Moving computation into the network with user-level software defined networking. The growth of software-defined networking ([11, 4]) has given us the ability to think about datacenter networks as an integrated, programmable system, but so far these techniques have been used primarily for controlling the forwarding plane of the network. This same technology can be extended to accelerate *user-level* computation in the network as well.

Switches have a number of interesting properties that make them a compelling place to run code. Given their highly-connected position in the network, switches could be used exploit rack-level locality. A switch that sees frequent requests for the same data could cache and return it, reducing latency, round trips, energy usage, and congestion. Applications that combine or reduce data from multiple nodes could do so in the switch before sending it to its destination, rather than copying all the unreduced data to a single node through a single network link before doing the reduction. Furthermore, switch hardware and software is generally simpler than in cluster nodes; power dissipation and thus thermal properties are more constant, reducing opportunities for software and hardware failures. Simpler software also reduces opportunities for the creation of tail latency.

But switches are also highly constrained devices: their high port count means that they must be able to process a huge number packets per second in the worst case, and with bounded delay. To accomplish this within a reasonable power budget, switch hardware must be designed to support limited computation and memory per request. Thus, user programs that run on switches must have bounded delay and resource usage. Hardware executing them must support common multi-tenancy requirements like protection and resource isolation.

The core idea of this project is to extend switches with event-driven processors supporting a restricted set of operations; these operations will be chosen to support execution of components of common datacenter applications. I will explore the design of these processors first in simulation, by modifying the P4 infrastructure to count architectural events, allowing me to build a model of the first-order properties of the idea. I will then refine that model by building a prototype cluster and modifying applications to use the switch-local processors; this could be done either by modifying an existing programmable, specialized network like that in Catapult [15], or by rewiring a cluster network so that a fraction of each switch's links are connected to colocated processors (FPGAs, network processors, or "wimpy nodes" depending on the computational and latency requirements discovered in simulation).

The features of these processors will be driven by analyzing common applications in datacenters, ideally leveraging real-world profiling data. At this point, I plan to explore a caching layer for a key-value store (requiring little computation and relaxed consistency), a machine learning parameter server that combines model update vectors in the network (requiring computation with relaxed consistency), and a lock manager in the spirit of Chubby [5] (requiring support for consensus algorithms like Paxos). Once the right in-switch primitives are known, a way to express them safely will be needed; I will look for collaborations with programming language experts to design an appropriate domain-specific language.

References

[1] G. Alverson, R. Alverson, D. Callahan, B. Koblenz, A. Porterfield, and B. Smith. Exploiting heterogeneous parallelism on a multithreaded multiprocessor. In *Proceedings of the 6th international conference on Supercomputing*, ICS '92, pages 188–197, New York, NY, USA, 1992. ACM.

[2] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. In *Proceedings of the 4th International Conference on Supercomputing*, ICS '90, pages 1–6, New York, NY, USA, 1990. ACM.

[3] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. Ix: A protected dataplane operating system for high throughput and low latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 49–65, Berkeley, CA, USA, 2014. USENIX Association.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[5] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[7] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[8] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 449–460, Washington, DC, USA, 2012. IEEE Computer Society.

[9] A. Kalia, M. Kaminsky, and D. G. Andersen. Using RDMA efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 295–306, New York, NY, USA, 2014. ACM.

[10] A. Kaufmann, S. Peter, T. Anderson, and A. Krishnamurthy. FlexNIC: Rethinking network DMA. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, pages 7–7, Berkeley, CA, USA, 2015. USENIX Association.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[12] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin. Pomace: A Grappa for non-volatile memory. In *Non-Volatile Memories Workshop (NVMW)*, 3 2013.

[13] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin. Latency-tolerant software distributed shared memory. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 291–305, Santa Clara, CA, July 2015. USENIX Association.

[14] S. Peter and T. Anderson. Arrakis: A case for the end of the empire. In *Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 26–26, Berkeley, CA, USA, 2013. USENIX Association.

[15] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 13–24, Piscataway, NJ, USA, 2014. IEEE Press.

[16] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage in solid-state memories. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 25–36, New York, NY, USA, 2013. ACM.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.