Research Statement

Hongyu Miao

I'm broadly interested in computer systems, including operating systems, networking, compiler, architecture, and runtime systems. In particular, my research interests are shaped by the emerging applications (e.g., big data analytics and machine learning) and hardware (e.g, multicore CPUs, 3D stacked memory, and advanced embedded devices). My research agenda focuses on the questions of how to design systems that can improve applications' performance by fully exploiting underlying hardware's features. In this statement, I summarize the contributions of my dissertation research, and outline my future research plan.

1 Dissertation research

My dissertation research lies in building systems for real-time big data analytics and machine learning (ML). On the one hand, data is exploding. A large volume of data is continuously being generated by data centers, humans, and internet of things (IoT). In order to get real-time insights, such enormous data must be processed in time with high throughput and low latency. On the other hand, new hardware is emerging. A large body of new hardware is being shipped by vendors, such as multicore CPUs, 3D stacked memory, and AI accelerators, aiming to speed up data processing. However, traditional systems (e.g., operating systems and general data analytics systems), the key layer that bridges high-level data processing applications and low-level hardware, fail to catch up with the fast evolving of data explosion and hardware emergence.

My dissertation work illustrates a possible approach to address this problem: building runtime systems for high-level data processing applications by exploiting low-level modern hardware. In particular, I build systems with three design goals in mind. First, the system must leverage domain-specific knowledge of high-level data processing applications. Second, the system must exploit the unique features of low-level hardware. Finally, the runtime system must be able to maximize the performance of data processing applications by fully utilizing the hardware features. With this approach, I have pursued my research agenda in the following directions:

Speeding up stream processing by exploiting multicore CPUs [5] Stream analytics on real-time events has an insatiable demand for throughput and latency. Most existing distributed stream engines (e.g., Spark Streaming and Apache Beam) are only optimized for tens/hundreds of machines, but leave every single multicore machine underutilized, which wastes resources.

StreamBox exploits the parallelism and memory hierarchy of modern multicore hardware on single machines. StreamBox executes a pipeline of transforms over records that may arrive out-of-order. The key contribution is to produce and manage abundant parallelism by generalizing out-of-order record processing to out-of-order epoch processing, and by dynamically prioritizing epochs to optimize latency. On a 56-core machine, StreamBox processes records up to 38 GB/sec (38M Records/sec) with 50 ms latency, which is an order of magnitude faster than existing stream engines, e.g., Spark and Beam.

Speeding up stream analytics by exploiting hybrid memory [2] Stream analytics has an insatiable demand for memory and performance. Emerging hybrid memories combine commodity DDR4 DRAM with 3D-stacked High Bandwidth Memory (HBM) DRAM to meet such demands. However, achieving this promise is challenging because (1) HBM is capacity limited and (2) HBM boosts performance best for sequential access and high parallelism workloads. At first glance, stream analytics appears a particularly poor match for HBM because they have high capacity demands and data grouping operations, their most demanding computations, use random access.

StreamBox-HBM exploits hybrid memories to achieves scalable high performance. It performs data grouping with sequential access sorting algorithms in HBM, in contrast to random access hashing algorithms commonly used in DRAM. It solely uses HBM to store Key Pointer Array (KPA) data structures that contain only partial records (keys and pointers to full records) for grouping operations. It dynamically creates and manages prodigious data and

pipeline parallelism, choosing when to allocate HBM. It dynamically optimizes for both the high bandwidth and limited capacity of HBM, and the limited bandwidth and high capacity of standard DRAM.

StreamBox-HBM is the first stream engine optimized for hybrid memories. It achieves 110 million records per second and 238 GB/s memory bandwidth while effectively utilizing all 64 cores of Intel's Knights Landing, a commercial server with hybrid memory. It outperforms stream engines with sequential access algorithms without KPAs by 7× and stream engines with random access algorithms by an order of magnitude in throughput.

Optimizing wearables' graphics stack by exploiting irregular displays [3] Computer displays have been mostly rectangular since they were analog. Recently, smart watches running Android Wear have started to embrace circular displays. However, the graphics stack – from user interface (UI) libraries to GPU to display controller – is kept oblivious to the display shape for engineering ease and compatibility; it still produces contents for a virtual square region that circumscribes the actual circular display. To understand the implications on resource usage, we have tested eleven Android Wear apps on a cutting edge wearable device and examined the key layers of Android Wear's graphics stack. We have found that while no significant amount of CPU/GPU operations are wasted, the obliviousness incurs excessive memory and display interface traffic, and thus leads to efficiency loss.

To minimize such waste, we advocate for a new software layer at the OpenGL interface while keeping the other layers oblivious. Following the idea, we propose a pilot solution that intercepts the OpenGL commands and rewrites the GPU shader programs on-the-fly. Through running a handcrafted app, we show a reduction in the GPU memory read by up to 22.4%. Overall, our experience suggests that it is both desirable and tractable to adapt the existing graphics stack for circular displays.

Enabling large neural networks on tiny embedded devices by exploiting external flash [4] To run neural networks (NNs) on microcontroller units (MCUs), memory size is the major constraint. While algorithm-level techniques exist to reduce NN memory footprints, the resultant losses in NN accuracy and generality disqualify MCUs for many important use cases. We investigate a system solution for MCUs to execute NNs out-of-core: dynamically swapping NN data chunks between an MCU's tiny SRAM and its large, low-cost external flash. Out-of-core NNs on MCUs raise multiple concerns: execution slowdown, storage wear out, energy consumption, and data security. We present a study showing that none is a showstopper; the key benefit – MCUs being able to run large NNs with full accuracy/generality – triumphs the overheads. Our findings suggest that MCUs can play a much greater role in edge intelligence.

2 Future research

As illustrated in my research, building systems for high-level applications by exploiting low-level hardware features is powerful to improve performance, reduce resource waste, and enable new use cases that were previously impossible. However, the potential of this approach has not been fully exploited yet, and I see more exciting opportunities, especially in enabling ML on low-power embedded devices (TinyML). There are 250 billion MCUs in the world [1]. Running ML on these MCUs has huge opportunities, but it's challenging due to limited hardware resources. My long-term research goal is to make ML applications ubiquitous on any tiny embedded devices to serve human's lives while requiring minimal human's efforts. In the following paragraphs, I outline my research plan.

Full-stack systems support for embedded intelligence ML on embedded devices is still in an early stage, and one key reason preventing it from becoming a reality is lacking of systems support. While my dissertation work [4] designs systems to enable large inference on embedded devices, I want to extend my research to address additional problems. First, training ML models on embedded devices locally is attractive because it can preserve users' privacy by avoiding sending data out and allow training in offline settings, but there is no system support yet. I want to investigate *how to build systems to enable training ML models on embedded devices by exploiting their whole storage hierarchy*. Second, in addition to MCUs/DSPs, more and more AI accelerators become available on modern embedded devices. I would like to investigate *how to design systems that can speed up ML inference/training on embedded devices by harnessing heterogeneous hardware accelerators*. Third, embedded devices are usually

deployed at large scale in areas that are not easy for human to physically access, e.g., wild areas and farms. I would like to investigate *how to design system tools that allow users to deploy and update ML models on embedded devices easily and remotely*. Fourth, embedded devices often carry users' sensitive data (e.g., surveillance cameras at home), but they are easy to attack. I want to investigate *how to design systems to safeguard users' data by exploiting hardware security features, like crypto accelerators and TrustZone*.

Innovating new use cases for embedded intelligence More than collecting physical data, embedded systems can be more intelligent with the ability of running ML. For sensor networks that are consist of networked embedded devices, I want to investigate its new use cases to make environment intelligent, e.g, ML-based smart homes/buildings/cities, with emphases on leveraging the advance of federated learning to guarantee user privacy. For wearable devices, I would like to investigate new use cases in ML-based medical diagnosing, health monitoring, and disease preventing, with emphases on designing more friendly user-machine interfaces that can collect human body's signals much easier and more accurate.

References

[1] Why tinyml is a giant opportunity. https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/, 2020.

[2] Hongyu Miao, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S McKinley, and Felix Xiaozhu Lin. Streambox-hbm: Stream analytics on high bandwidth hybrid memory. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 167–181, 2019.

[3] Hongyu Miao and Felix Xiaozhu Lin. Tell your graphics stack that the display is circular. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, pages 57–62, 2016.

[4] Hongyu Miao and Felix Xiaozhu Lin. Enabling large neural networks on tiny microcontrollers with swapping. *arXiv:2101.08744*, 2021.

[5] Hongyu Miao, Heejin Park, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S McKinley, and Felix Xiaozhu Lin. Streambox: Modern stream processing on a multicore machine. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 617–629, 2017.