Research Statement

Scott Beamer

Despite the end of Moore's Law on the horizon, there is no end in sight to the rapid growth in the volume of data and applications to make use of it. To use that data, it must be stored, accessed, and moved, and this communication is often more demanding than the computation on that data. Worse yet, inefficient communication can leave a system woefully underutilized, which increases costs to the point of limiting the amount of data that can be practically processed. My research attacks this problem by enabling *efficient communication*, and I leverage my background in computer architecture and systems to consider both hardware and software perspectives to develop the most efficient solutions. My approach is able to greatly improve communication efficiency by creating novel algorithms, holistic system designs, and specialized hardware architectures. Such a multifaceted approach is necessary, as achieving the best communication efficiency will require rethinking the hardware-software interface.

Communication efficiency is paramount for effective data-intensive computing. First, improving communication efficiency can improve performance, particularly for communication-bound workloads. Moving less data and moving data shorter distances can save substantial amounts of energy, as moving data typically consumes orders of magnitude more energy than a single computation on it. For many systems, off-chip bandwidth is one of the most expensive resources. Therefore, building a system that uses available bandwidth more effectively will reduce the cost and size of a system. These benefits from improved communication efficiency will also translate into enhancements in application quality by enabling the use of larger datasets as well as more complex algorithms. As Moore's Law shows signs of slowing down, improving communication efficiency will be critical to reconciling the growing disparity between rapidly growing data processing demands and sluggishly increasing transistor budgets.

To improve communication efficiency, one can either reduce the amount of communication (move less data) or increase the effective rate of communication (utilize more bandwidth). My research applies both techniques, and I have improved communication efficiency via optimized algorithms and redesigned hardware. I reduced communication for graph algorithms by designing a locality-enhancing transformation as well as a novel breadth-first search algorithm. I increased the effective rate of communication for graph algorithms by identifying and ameliorating bottlenecks constraining memory bandwidth. I also investigated how to best utilize a new interconnect technology (silicon photonics) to design systems with improved performance, increased energy efficiency, and improved programmability. I am currently developing methods to accelerate digital logic simulation by improving its communication efficiency with my insights into efficient graph algorithm execution. I am excited to bring my communication-centric approach to more application areas by designing new algorithms and architectures to improve their communication efficiencies.

Using Graph Algorithms to Understand and Improve Communication

Many workloads can benefit from improved communication efficiency, but for my dissertation research, I focused on improving communication efficiency for graph processing for two reasons. First, graphs are a great abstraction to represent many types of data and improved graph processing performance will benefit its many applications. Second, the connection-centric nature of graph processing results in a communication-centric workload that is representative of other data-intensive workloads. Typically within graph algorithms, communication is guided by the graph topology, so by examining a variety of algorithms and graph topologies, we can examine a variety of communication patterns.

Direction-optimizing Breadth First Search
Breadth-first search (BFS) is a fundamental traversal order used within many graph algorithms. A conventional BFS traversal examines every edge in a graph, but by applying a communication-centric perspective, I designed the *direction-optimizing BFS* algorithm which is often able to leave many edges unexamined [5]. The key insight is that in low-diameter graphs such as social networks, vertices typically have many parents. These redundant parents cause redundant communication, since a correct BFS traversal only needs to find a single parent for each vertex. By performing the traversal in the reverse direction (bottom-up instead of top-down), my algorithm is able to skip edge examinations for vertices once they have found parents. By drastically reducing the number of edges examined which in turn reduces the amount of communication (often by an order of magnitude), my algorithm obtains impressive speedups. This BFS innovation shows that there can be more to optimizing communication than simply moving a blob of data quicker. In this case, leveraging insight into how the data will be used led to an algorithm that dynamically determines which subset of the data it needs.

I first introduced the direction-optimizing BFS algorithm within the Graph 500 competition [3]. Graph 500 ranks the world's supercomputers by their BFS traversal rate, much like the Top 500 competition ranks the world's supercomputers by their ability to perform dense linear algebra. Despite only placing 17th, I had used only a single quad-socket server and outperformed clusters of hundreds of servers, Cray XMTs (supercomputer specialized for irregular applications), and Convey HCs (reconfigurable computer). Such a small system outperforming such large systems demonstrates the potential for communication optimizations to substantially improve efficiency. For low-diameter graphs, the direction-optimizing BFS algorithm has become essential due to its performance advantages. It has been implemented by many graph frameworks and Graph 500 contestants, including at least the top 30 finishers in November 2017's Graph 500 rankings.

I collaborated with Aydın Buluç to extended my algorithm to operate on distributed memory. With careful design, we preserved the algorithm's communication efficiency in order to transmit an order of magnitude less data on the network to obtain significant speedups [8]. The efficiency of our implementation encouraged us to scale out, and by using one hundred thousand cores from the Hopper supercomputer, our implementation traversed a graph with half a trillion edges in only a few seconds.

Workload Characterization
A closer analysis of my BFS algorithm revealed that although it obtains its speedup by performing less communication by traversing fewer edges, it is less communication efficient for each edge actually traversed. To understand what limits communication efficiency, I thoroughly characterized graph processing workloads (Best Paper Award from IISWC) [6]. With a diverse suite of different codebases, graph kernels, and input graphs, I used hardware performance counters to identify and quantify architectural bottlenecks for graph processing workloads. Contrary to the popular belief that graph algorithms have a random memory access pattern and are thus memory bandwidth-bound, I found that most of these workloads actually enjoy moderate cache hit rates and are unable to fully utilize the platform's memory bandwidth. Their memory bandwidth utilization is most frequently limited by the processor's inability to generate a sufficient number of concurrent memory requests. In general, I observed the processor is frequently idle and is able to underutilize its compute throughput and memory bandwidth simultaneously. Fortunately, these results suggest that by improving utilization, a new processor architecture or even a new algorithm can improve performance without requiring a costly new memory system.

GAP Benchmark Suite
While creating the workload for analysis, I found it difficult to compare much of the research on improving graph processing performance due to the great diversity in evaluation methodologies, as there is no standard for which graph problems to solve, which graphs to use for input, or how to actually conduct the experiments. Like many other subfields within systems, being able to reliably perform quantitative comparisons is essential. I made the *GAP Benchmark Suite* to solve this problem by standardizing graph processing evaluations [4]. The benchmark specifies the requirements for the evaluation, so any implementation that

is compliant can now be correctly compared. As an exemplar of the benchmark specification, I also released high-performance reference code, which for some of the graph kernels, is also the fastest available. The performance advantages are primarily due to communication efficiency optimizations guided by the characterization work. Since its release, the benchmark has been used for evaluations in papers in top conferences (ASPLOS, ISCA, PACT, and SPAA) as well as being highlighted in the citation for my dissertation award from SPEC [2].

Propagation Blocking
A stream of memory requests with poor spatial locality is a common inefficiency for graph processing workloads. When one of these memory requests misses the cache, it triggers a full cache line transfer when it will use only a single word, which wastes both memory bandwidth and energy. To mitigate this communication inefficiency, I developed *propagation blocking*, an algorithmic transformation that improves spatial locality by using additional memory capacity to semi-sort the data. Although propagation blocking requires more arithmetic operations, it improves spatial locality which significantly reduces the amount of memory communication to provide a net speedup. I used PageRank as a kernel to explain and demonstrate the technique (Best Paper Award from IPDPS), but it can be applied to other graph algorithms and more generally, anything that performs a sparse all-to-all transfer [7].

Domain-specific Language for Graph Algorithms
To make it easy to concisely express graph algorithms without loosing performance, I made GBSP, a simple domain-specific language (DSL) for graph applications [12]. At runtime, GBSP automatically translates the DSL into high-performance parallel native code and executes it. Under the hood, GBSP uses propagation blocking for efficient communication to deliver great performance. GBSP is embedded within Python, so it is easy to use and can integrate with the large volume of existing Python code. GBSP demonstrates that communication optimizations can be made more accessible by including them transparently within a framework.

Using Silicon Photonics to Build Efficient Interconnects

Silicon photonics is an emerging technology with promising communication applications. I used my architecture training to help VLSI and device experts at MIT who were building photonic test chips to decide the best way to utilize photonics for both inter-chip communication and intra-chip communication to improve overall system performance. Photonics is an interesting opportunity for architects, since its advantages of bandwidth density, distance insensitivity, and energy efficiency alter many design tradeoffs, often necessitating much more novel designs. Since we were working with device models for a future technology, we swept the simulated device parameters to motivate which device performance targets would be necessary in order to merit the use of photonics.

We designed interconnects for various parts of the system, including within a single chip [10], between multiple sockets [1], and between a socket and memory [9]. In addition to increasing bandwidth and improving energy efficiency, our new interconnect designs bring other benefits to the system. Our multisocket interconnect leveraged the high bandwidth possible with photonics to provide uniform memory access in a multisocket system (not NUMA) to simplify performance programming and enable disintegrating large chips for a cost reduction. Our memory interconnect allows for greatly increased memory capacity by taking advantage of the distance insensitivity of photonics and our novel optical power guiding technique. In the years since we designed our optical memory link, the device experts succeeded in fabricating a test chip in which a processor operates correctly using optically connected memory, and it was such a radical breakthrough that it was published in *Nature* [13].

Future Work

I am thrilled to continue improving communication efficiency within hardware and software systems, and I am eager to collaborate with others to bring my communication-centric approach to more application areas and to utilize emerging technologies such as photonics, 3D integration, and non-volatile memory.

Accelerating Architecture Simulation via Communication Optimization
I am currently combining my expertise in accelerating graph processing with my interest in architecture to speed up digital logic simulation. Software simulation is a critical tool for hardware designers, but the slow rate of simulation currently bottlenecks the design process by limiting the number of designs that can be explored or the time simulated for each design. I recast logic simulation as a graph evaluation problem, and by applying my approach, I improved its communication efficiency to achieve substantial speedups. My simulation tool exploits the observation that large parts of hardware designs are often inactive, so it only simulates active portions of the design. Normally, using conditional execution in this context introduces too much overhead, but I developed a novel acyclic graph partitioning technique to make it profitable. A publication detailing my project's contributions is currently under review, and I am preparing the code for an open-source release. This project has unearthed a wealth of potential communication-inspired optimizations to further accelerate logic simulation.

Accelerating Bioinformatics via Communication Optimization
Bioinformatics will benefit substantially from improved communication efficiency, and that efficiency will enable processing larger datasets and tackling more complex problems. Bioinformatics not only has to process massive amounts of data, but in practice, the computation is often bottlenecked by inefficient communication, even within a single system. By leveraging techniques and tools developed for my graph processing workload characterization, I will identify the most impactful sources of communication inefficiencies. With those insights, I will design new blocking techniques or methods of exploiting parallelism to substantially improve communication efficiency for bioinformatics workloads. Those optimizations could be manually applied to library functions, but it will be more useful to embed those optimizations within an optimizing compiler for a DSL. This will allow users of the DSL to express new algorithms but still gain the benefits of the optimizations.

Communication-Efficient Architecture
To cope with Moore's Law slowing down and providing fewer additional transistors, computer architects are increasingly improving performance via hardware specialization to better utilize those transistors. In many cases, the benefit from specialization will not come solely from increasing raw compute throughput, but instead from tailoring the memory hierarchy to the application to improve communication efficiency. Even the renowned compute throughput of Google's Tensor Processing Unit (TPU) is enabled by its efficient communication abilities and how well matched they are to its application [11]. For data-intensive applications, communication should displace compute as the primary concern for computer architects.

In contrast to a conventional hardware design process which first considers the computational needs of the target workload and then designs a memory system to match, I will put my communication-centric perspective into practice by designing for efficient communication first. To design for efficient communication, I will determine the minimum amount of data movement required to solve the problem, how local memories should be used to achieve that data movement minimum, how the system can most efficiently generate memory requests to utilize the memory bandwidth, and finally the compute necessary to keep up with the flow of data. This design process will yield communication-efficient accelerators for specific applications, but to support more workloads, I also want to design a more flexible architecture that achieves nearly the same communication efficiency.

To achieve reusability and communication efficiency, I will design a reconfigurable spatial architecture. Unlike a conventional general-purpose processor which performs computation temporally by executing instructions, this novel architecture will perform computation spatially by using different hardware blocks. To obtain reusability, the architecture will be able to reconfigure both the blocks and the connections between them. By decomposing communication patterns into communication primitives, the user will be able to construct efficient communication streams by mapping the primitives onto the hardware blocks. In contrast to prior research on coarse-grained reconfigurable architectures, my focus is on enabling efficient communication more so than increasing compute density. I will use graph algorithms and bioinformatics workloads to drive its development, but more generally, the architecture will excel at data-intensive workloads.

References

[1] S. Beamer. Designing multisocket systems with silicon photonics. Master's thesis, University of California, Berkeley, 2009.

[2] S. Beamer. Understanding and Improving Graph Algorithm Performance. PhD thesis, University of California, Berkeley, 2016.

[3] S. Beamer, K. Asanović, and D. Patterson. Searching for a parent instead of fighting over children: A fast breadth-first search implementation for Graph500. Technical Report UCB/EECS-2011-117, EECS Department, University of California, Berkeley, 2011.

[4] S. Beamer, K. Asanović, and D. A. Patterson. The GAP benchmark suite. arXiv:1508.03619, 2015.

[5] S. Beamer, K. Asanović, and D. A. Patterson. Direction-optimizing breadth-first search. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[6] S. Beamer, K. Asanović, and D. A. Patterson. Locality exists in graph processing: Workload characterization on an Ivy Bridge server. *International Symposium on Workload Characterization (IISWC)*, 2015.

[7] S. Beamer, K. Asanović, and D. A. Patterson. Reducing pagerank communication via propagation blocking. *International Parallel & Distributed Processing Symposium (IPDPS)*, 2017.

[8] S. Beamer, A. Buluç, K. Asanović, and D. A. Patterson. Distributed memory breadth-first search revisited: Enabling bottom-up search. *Workshop on Multithreaded Architectures and Applications (MTAAP), at the International Parallel & Distributed Processing Symposium (IPDPS)*, 2012.

[9] S. Beamer, C. Sun, Y.-J. Kwon, A. Joshi, C. Batten, V. Stojanović, and K. Asanović. Re-architecting dram memory systems with monolithically integrated silicon photonics. *International Symposium on Computer Architecture (ISCA)*, 2010.

[10] A. Joshi, C. Batten, Y.-J. Kwon, S. Beamer, I. Shamim, K. Asanović, , and V. Stojanović. Silicon-photonic clos networks for global on-chip communication. *International Symposium on Networks-on-Chip*, 2009.

[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture (ISCA)*. ACM, 2017.

[12] S. Kamil, D. Coetzee, S. Beamer, H. Cook, E. Gonina, J. Harper, J. Morlan, and A. Fox. Portable parallel performance from sequential, productive, embedded domain-specific languages. *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2012.

[13] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. M. Shainline, R. R. Avizienis, S. Lin, et al. Single-chip microprocessor that communicates directly using light. *Nature*, 528(7583):534–538, 2015.