Research Statement

Rohit Zambre

The architectures of supercomputers have critically evolved over the last decade, jeopardizing traditional models of programming them. With the increase in number of cores per processor and, more recently, the acceleration with multiple GPUs per node, heterogeneous architectures are becoming the norm. The supercomputers Sierra and Summit, for exampe, feature 4 and 6 GPUs per node, respectively. To fully utilize the computational capability of a modern supercomputer, domain scientists need to program both processors and GPUs. In the processor space, MPI+threads (e.g. MPI+OpenMP) is gaining popularity over the traditional memory-hungry MPI everywhere approach since it better maps to modern processors—where MPI everywhere runs out of memory, MPI+threads is able to run [5, 2]. MPI+threads, however, is slower due to its dismal communication performance. The story is the same for GPU-to-GPU communication even if each GPU is driven by a distinct thread, and even when MPI libraries utilize GPU technologies like GPUDirect RDMA. A more fundamental problem exists: **the supercomputing community views the network as a single device**.

A trend that that has been overlooked by the supercomputing community is the increase in network parallelism available on a node of a supercomputer. The network interface cards (NICs) of modern interconnects, such as Intel Omni-Path, feature multiple network hardware contexts [1]. Moreover, as we approach the throughput limits of a single network link, the only way to increase the aggregate bandwidth from a node is through the use of multiple NICs, implying even more parallelism. Summit, for example, already features dual-rail Mellanox EDR InfiniBand. Domain scientists, however, typically do not express logical parallelism in their MPI+threads communication because state of the art MPI libraries use conservative approaches, such as a global critical section, to maintain MPI's ordering constraints. The community has not capitalized on existing hardware capabilities. This outdated view does not hurt MPI everywhere since each MPI process transparently maps to a distinct network context, but it hurts MPI+threads drastically.

The goal of my research is to analyze and dissolve the communication bottleneck in MPI+threads through practical solutions. Existing solutions haven't been adopted by the community since they either sacrifice correctness for performance or jump to MPI standard extensions without fairly comparing the capabilities of the existing standard. To tackle this challenging yet critical problem, I have taken a bottom-up approach. I first investigate the limits of multithreaded communication on modern network hardware, and then devise a new MPI implementation that leverages the lessons learned at the lower level for fast MPI+threads communication that matches the performance of MPI everywhere without extensions to the standard, unlike prior works. **The results of this research will unlock the true potential of MPI+threads**, which when combined with a GPU programming model, will best fit modern heterogeneous supercomputing architectures.

1 Ph.D. dissertation research

1.1 Capabilities of modern hardware

The first fundamental question I sought to answer was: **what are the differences when threads drive network resources in parallel instead of processes?** The answer to this question is not straightforward since the design space of creating multiple software communication portals in a multithreaded environment (MPI+threads) is much larger than that in a multi-process environment (MPI everywhere). In the former, one can share certain software resources between threads while keeping others independent. In the latter, on the other hand, every process gets its own copy of each software resource. The question then evolves to: what level of resource sharing is ideal in a multithreaded environment? My research on Mellanox InfiniBand, shows that the answer depends on the performance requirements and availability of resources at a particular instance of a runtime system, such as an MPI library. There exists a tradeoff space between performance and resource efficiency. We have proposed a resource sharing model, scalable communication endpoints, which categorizes the design space into six categories going from maximum resource usage (highest performance) to maximum resource efficiency (lowest performance). The resource sharing model also provides an answer to the fundamental question: **multiple threads can achieve the same communication performance as that of multiple processes using just a third of the resources** [6]. This research is the first of its kind and has been published at IEEE ICPADS 2018. The poster presentation for this paper received the Best Poster Presentation award.

The recent addition of non-traditional players, such as Arm processors and AMD GPUs, in supercomputing suggested another question: **how do the communication performances of non-traditional architectures fare?** To answer this question, I collaborated with Arm Research and analyzed, using high-precision PCIe analyzers, the performance breakdown of Arm-based ThunderX2 servers connected with Mellanox InfiniBand. The investigation resulted in analytical models that explain the injection overhead and end-to-end latency of a system within a 5% margin of error [8]. The communication performance when driven by Arm-based servers is on par with that when driven by traditional Intel servers, but it is not faster. **The models pinpoint bottlenecks in the critical path of communication** where the Arm-based server is slower than Intel servers, guiding the optimization efforts of the engineers at Arm. Such work is the first of its kind on an Arm architecture and has been published at ACM ICPP 2019. The methodology detailed in the paper will help system architects measure and analyze the breakdown of any system of their interest.

1.2 Fast MPI+threads

Having learned that an MPI+threads environment can achieve the same communication performance as an MPI everywhere environment, I sought to answer the capstone question of my dissertation: how can domain scientists effectively utilize the underlying network parallelism through MPI? If the communication of different threads is logically parallel, the threads can map to distinct hardware contexts on the NIC. To facilitate the expression of this logical parallelism, user-visible MPI Endpoints has been proposed as an extension to the MPI-3.1 standard [3]. Users would express communication parallelism by creating endpoints within an MPI process and the MPI library would map the endpoints to distinct network contexts. If users map each thread to a distinct endpoint, they would establish a dedicated channel to the network for each thread. The onus of managing the network resources, however, is on domain scientists, hurting their productivity.

A key insight, though, is that the existing MPI standard already provides ways to express logical parallelism. Although MPI specifies certain sequential ordering between messages, it also provides ways to overcome them. The user can inform the MPI library that two or more messages have no relative ordering between them by using, for example, different communicators, thus expressing parallelism between the messages.

So, the research question evolved to: **are extensions to the standard necessary to improve MPI+threads communication?** My research has sought to answer the question from a performance perspective through a new implementation of the MPI-3.1 standard (based on MPICH/CH4) that internally uses multiple virtual communication interfaces (VCIs). A VCI represents an ordered communication stream and maps to a distinct network hardware context. When users express communication parallelism through existing MPI mechanisms (such as communicators, ranks, tags, and windows), the MPI library maps that parallelism to distinct hardware contexts by funneling messages over distinct VCIs [7]. The VCIs are completely hidden within the MPI library, thus neither requiring extensions to the MPI standard nor burdening domain scientists with thread-to-network-resource mapping. More important, unlike the demonstration of MPI Endpoints, our new MPI-3.1 implementation does not sacrifice correctness for performance. Multinode analysis on microbenchmarks and communication kernels of applications, such as stencils, show that **VCIs perform as well as user-visible endpoints** in the majority of cases. In fact, **VCIs perform nearly as well as MPI everywhere**. In a few cases where the semantics of MPI-3.1 prevent the user from expressing logical parallelism, however, user-visible endpoints perform better than VCIs. Our paper at ACM ICS 2020 details the results of this research and provides users with recommendations on how to express parallelism in their communication with MPI-3.1.

1.3 A usability perspective on VCIs

During my performance evaluation of VCIs, I observed that expressing parallelism with MPI-3.1 can be clumsy compared to that with user-visible endpoints. For example, expressing communication parallelism for a 3D 27-point stencil, a common case, with communicators is much more complex than expressing parallelism with the MPI Endpoints API. The matching requirements—both the sender and receiver thread must use the same communicator—is the primary reason for this complexity. Expressing parallelism is straightforward with endpoints since each endpoint is directly addressable. So, although MPI-3.1 does not introduce a new API to domain scientists, it may still hurt their productivity because of the complexity in expressing parallelism. On the other hand, expressing parallelism with user-visible endpoints is easier but they put the burden of mapping to network resources on the user. This "burden", however, could just be a preconceived notion, which I myself was a victim of in my study as devil's advocate to user-visible endpoints.

Our ongoing collaborations and conversations with MPI+threads application developers confirm the notion that the term "Endpoints" guides domain scientists to believe that it is their job to map to network resources. However, similar to our guiding principle with MPI-3.1, it is possible to decouple the roles of user-visible endpoints in expressing logical parallelism and mapping to network resources. User-visible endpoints could be another way for users to express logical points of parallelism, and the role of mapping these parallel points to parallel network resources would be the job of the MPI library. In other words, endpoints could be a virtual layer over the underlying network resources. To combat with preconceived notions, there is some merit to rebranding MPI Endpoints to, say, MPI Parapoints.

VCIs are critical for MPI+threads performance and are here to stay, but **what is easiest for domain scientists to express logical parallelism?** Domain scientists, of course, are best suited to answer this question and hence we are conducting a survey to collect the perspectives of MPI+threads application developers.

2 Future research

I am broadly interested in network performance and, in particular, I am interested in thorough investigations of modern network hardware capabilities. My experience as a doctoral student has shown me that there tends to be a disconnect between the capabilities of the network hardware and the software that uses it. This is true not just at high-level runtime systems such as MPI libraries but also at low-level network drivers. For example, the mlx5 driver of modern Mellanox network cards wastes up to 95% of its network hardware registers by default [6].

In the near future, I plan to continue my research in supercomputing networking to combat outdated views of the network as a single device. The slow communication performance of MPI+threads has been a decade-long conundrum and still has some ways to go before fast MPI+threads communication becomes the new normal. Below are research directions I would like to pursue to accelerate the attainment of the new normal.

Multi-VCI design exploration. MPI libraries must map to multiple network resources in a process. The design space of creating multiple VCIs, however, is a large one. Moreover, a design that is best for one interconnect may not be the best one for another. Modern MPI libraries use abstract communication libraries such as OFI and UCX for their network communication and hence, the design that provides the best performance for an interconnect is dependent on the support that the given interconnect provides for OFI/UCX. I would like to produce a suite of microbenchmarks that can be used as a tool to accelerate this design exploration process for MPI library developers.

VCIs for deep learning. Distributed deep learning has single-handedly influenced the heterogeneous architectures of supercomputers. Distributed deep learning frameworks, such as PyTorch, are still evolving and they currently use an MPI everywhere approach to drive the GPUs (a process per GPU). Even with a process per GPU, though, some communication frameworks, such as Horovod, maintain asynchronous communication threads and drive MPI communication in parallel. Since the communication from multiple GPUs is independent in the common data parallelism approach, the idea is that **threads driving different GPUs can use distinct VCIs** and hence utilize the underlying network parallelism in an MPI+threads approach.

GPU initiated MPI. State of the art GPU networking relies on the CPU to drive the communication. This exhibits obvious synchronization overheads since the communication can occur only after the GPU kernels finish executing. One way to eliminate this overhead is by initiating network communication from the GPU-kernel threads themselves thereby eliminating the CPU's involvement in the critical path of communication. Researchers have recently tried out this radical idea in the context of OpenSHMEM and demonstrate significant improvements in latencies for communication initiated at a work-group (AMD ROCm terminology) level [4]. The benefits of completely asynchronous (GPU thread-level) GPU-to-GPU communication in the context of the MPI standard remains to be investigated. Such a study would be important since traditional scientific applications—not just deep learning— are also adopting their code bases to utilize GPUs.

References

[1] Intel Omni-Path Fabric Host Software. `https://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/Intel_OP_Fabric_Host_Software_UG_H76470_v9_0.pdf`.

[2] A. Buluç, S. Beamer, K. Madduri, K. Asanovic, and D. Patterson. Distributed-memory breadth-first search on massive graphs. *arXiv preprint arXiv:1705.04590*, 2017.

[3] J. Dinan, R. E. Grant, P. Balaji, D. Goodell, D. Miller, M. Snir, and R. Thakur. Enabling communication concurrency through flexible MPI endpoints. *The International Journal of HPC Applications*, 28(4):390–405, 2014.

[4] K. Hamidouche and M. LeBeane. GPU initiated OpenSHMEM: correct and efficient intra-kernel networking for dGPUs. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 336–347, 2020.

[5] R. Thakur, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, and J. L. Träff. MPI at Exascale. *Proc. of SciDAC*, 2:14–35, 2010.

[6] R. Zambre, A. Chandramowlishwaran, and P. Balaji. Scalable Communication Endpoints for MPI+Threads Applications. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 803–812. IEEE, 2018.

[7] R. Zambre, A. Chandramowlishwaran, and P. Balaji. How I Learned to Stop Worrying About User-Visible Endpoints and Love MPI. *arXiv preprint arXiv:2005.00263*, 2020.

[8] R. Zambre, M. Grodowitz, A. Chandramowlishwaran, and P. Shamis. Breaking Band: A Breakdown of High-performance Communication. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–10, 2019.