Data Analytics, Accelerators, and Supercomputing
The Challenges and Future of MPI

The Message Passing Interface faces new challenges as modern concepts and technologies like data analytics and accelerators penetrate high-performance computing. Here, we discuss the future of MPI with HPC expert Prof. Torsten Hoefler of ETH Zürich, Switzerland.

By Vasileios Kalantzis

DOI: 10.1145/3055600

Some years ago, as a graduate student in Greece, I showed my slides to a colleague for a talk I was to give at the 2012 SIAM Parallel Processing Conference, prompting him to ask, "All good, but why do you guys doing numerical linear algebra and parallel computing always use the Message Passing Interface to communicate between processors?" Having read Parlett [1], I had the wit to imitate Marvin Minsky and say, "Is there any other way?" The same question today would be even more interesting, and my answer would certainly be longer, perhaps too long. Execution of programs in distributed computing environments requires communication among perhaps thousands of processors. It is thus natural to consider by what protocols and guidelines the processors should communicate with one another. This is the question for which the Message Passing Interface, or MPI has been the answer for more than 25 years.

MPI is a standardized portable message-passing system designed for the communication of processors in cluster-computing environments or even today's desktops and laptops with, say, two to four compute cores each. MPI is designed to function on a variety of parallel computing architectures and provide vendors a base set of routines they can implement efficiently. Each vendor implements its own portable version, with the only requirement being adherence to MPI protocols. Since its first version in 1994, MPI has been widely adopted by researchers from academia and industry.

But today's computer science and high-performance computing (HPC) are not the same as they were 25 years ago. From an application's viewpoint, until a few years ago, the HPC community has sought to solve large-scale linear systems and eigenvalue problems required by the discretization of partial differential equations used to model real-world phenomena (simulations). There is huge interest focused on exploiting the massive amounts of data constantly being

generated in order to infer patterns and trends. Given the enormous volume of data that must be analyzed, their analysis requires computers with excessive compute capabilities that are bundled by adding together thousands of processors, each with multiple cores, into machines we tend to call "supercomputers."

MPI has long been the de facto language by which the different processors in supercomputers communicate, and it is no overstatement to say current progress on supercomputers is owed largely to MPI. But is MPI the right communication protocol for new disciplines like data analytics? Should MPI undergo major change, especially now that scientists with a limited background in computer science are also interested in solving large-scale scientific, business, statistical, and network problems requiring supercomputers?

We were thus prompted to contact Torsten Hoefler to help shed light on these issues. He is today an assistant professor of computer science at ETH Zürich. Before joining ETH, he led the performance-modeling and simulation efforts of parallel petascale applications for the National Science Foundation-funded Blue Waters project at the National Center for Supercomputer Applications at the University of Illinois at Urbana-Champaign. He is also a key member of the MPI Forum, chairing the Collective Operations and Topologies working group. He won best paper awards at the ACM/IEEE Supercomputing Conference 2010, EuroMPI 2013, ACM/IEEE Supercomputing Conference 2013, and other prestigious conferences. He has published numerous peer-reviewed scientific conference and journal articles and authored chapters of the MPI-2.2 and MPI-3.0 standards. For this work, he received the SIAM SIAG/Supercomputing Junior Scientist Prize in 2012 and the IEEE TCSC Young Achievers in Scalable Computing Award in 2013. Following his Ph.D., he received the Young Alumni Award 2014 from Indiana University. He was elected onto the first steering committee of ACM's SIGHPC in 2013. His research interests revolve around "performance-centric software development," including scalable networks, parallel programming techniques, and performance modeling. Additional information is available at Hoefler's homepage htor.inf.ethz.ch and http://ppopp17.sigplan.org/profile/torstenhoefler

"In general, integrating accelerators and communication functions is an interesting research topic."

The following interview has been condensed and edited for clarity.

XRDS: MPI began about 25 years ago and has since become the "king" of HPC. What characteristics of MPI make it the de facto language of HPC?

TORSTEN HOEFLER (TH): I would say MPI's major strength is due to its clear organization around a relatively small number of orthogonal concepts, including communication contexts (communicators), blocking/nonblocking, datatypes, collective communications, and remote memory access. They work like Lego blocks and can be combined into powerful functions. For example, the transpose of a parallel fast fourier transformation can be implemented with a single call to a nonblocking MPI_Alltoall when using datatypes [2]. Big parts of MPI's success also stem from its simplicity to implement libraries and applications. MPI's library interface requires no compiler support to be added to most languages and is thus easily implemented.

XRDS: What new features should we expect in the future? Will the standard have to address any specific challenges on the way to exascale computing?

TH: I would conjecture that the first exascale application will use MPI, even if it's not extended until then. MPI's latest,

version 3, is scalable and has very basic support for fault tolerance, making it a great candidate for early exascale systems. However, among the things that could benefit very large-scale execution are improved fault-tolerance support, better interaction with accelerators, and task-based executions.

XRDS: We hear much lately about the era of exascale computing and the extreme computing power it will deliver. Given the increasing gap between the speed of processors compared to bringing data from memory, how should we proceed?

TH: Minimizing and scheduling communication efficiently will be one of the major challenges for exascale. Very large-scale machines already have relatively low communication bandwidth. While minimizing communications is a problem for algorithm developers, communication scheduling is managed by the MPI implementation. MPI-3 added several interfaces to enable more powerful communication scheduling, in, say, nonblocking collective operations and neighborhood collective operations. Implementation of these interfaces has spawn interesting research questions that are being tackled by the community right now.

XRDS: With the rise of data analytics, new parallel computing paradigms have emerged, including Spark and Hadoop. How do you see MPI competing against these paradigms in big data applications, where other considerations (such as length of the code, fault tolerance, and non-CS scientists) are enabled?

TH: MPI originated from within the HPC community, which runs large simulation codes that have been developed over decades on very large systems. Much of the big data community moved from single nodes to parallel and distributed computing to process larger amounts of data using relatively short-lived programs and scripts. Programmer productivity thus played only a minor role in MPI/HPC, while it was one of the major requirements for big data analytics. While MPI codes are often orders-of-magnitude faster than many big data codes, they also take much longer to develop—which is most often a good trade-off. I would not want to spend weeks to write a data-analytics application to peek at some large dataset. Yet when I am using tens of thousands of cores running the same code, I would probably like to invest in improving the execution speed. So I don't think the models are competing; they're simply designed for different uses and can certainly learn much from each other.

"I would say that MPI's major strength is due to its clear organization around a relatively small number of orthogonal concepts."

XRDS: MPI supports I/O operations, but many applications of interest today are in the big data regime where large chunks of data have to be read/written from/to the disk. What is the status of MPI-I/O? Will anything new be part of the next release?

TH: MPI I/O was introduced nearly two decades ago to improve the handling of large datasets in parallel settings. It is used successfully in many large applications and I/O libraries, including HDF-5. It could also be used to improve large block I/O in MapReduce applications, though it needs to be shown in practice. I do not know of any major new innovations in I/O planned for MPI-4.

XRDS: Compute accelerators like GPUs play an important role in modern data analytics and machine learning. How has MPI responded to this change?

TH: MPI predates the time when the use of accelerators became commonplace. However, when accelerators are used in distributed-memory settings (such as computer clusters), MPI is the common way to program them. The current model, often called "MPI+X" (such as MPI+CUDA), combines traditional MPI with accelerator programming models (such as CUDA, OpenACC, and OpenMP) in a simple way. In this model, MPI communication is performed by the CPU. Yet this can be inefficient and inconvenient. We recently proposed a programming model called distributed CUDA, or dCUDA, to perform communication from within a CUDA compute kernel [3]. This allows use of the powerful GPU warp scheduler for hiding communication latency. In general, integrating accelerators and communication functions is an interesting research topic. Even a model called MPI+MPI combining MPI at different hierarchy levels seems very useful in practice, depending, of course, on the application [4].

ACKNOWLEDGMENTS

We thank Torsten Hoefler for sharing his knowledge, as well as Geoffrey Dillon and Pedro Lopes for their comments on an earlier version of this interview. I also thank Efstratios Gallopoulos of the University of Patras for bringing to my attention the book review by Parlett [1] of the University of California, Berkeley. Any errors are solely the responsibility of the author.

References

1. Parlett, B.N. Review of *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods* by D. Watkins and *Numerical Methods for General and Structured Eigenvalue Problems* by D. Kressner. *SIAM Review 52*, 4 (2010), 771–791.
2. Hoefler, T. and Gottlieb, S. Parallel zero-copy algorithms for fast fourier transform and conjugate gradient using MPI datatypes. In *Proceedings of the 17th European MPI Users Group Meeting Conference on Recent Advances in the Message Passing Interface* (Stuttgart, Germany, Sept. 12–15, 2010), 131–141.
3. Gysi, T., Baer, J., and Hoefler, T. dCUDA: Hardware supported overlap of computation and communication. In *Proceedings of the ACM/IEEE Supercomputing Conference* (Salt Lake City, UT, Nov. 13–18). ACM Press, New York, 2016.
4. Hoefler, T., Dinan, J., Buntinas, D., Balaji, P., Barrett, B., Brightwell, R. et al. MPI+MPI: A new hybrid approach to parallel programming with MPI plus shared memory. *Journal of Computing 95*, 12 (2013), 1121–1136.

Biography

Vasileios Kalantzis is a Ph.D. candidate in the Computer Science and Engineering Department of the University of Minnesota, Minneapolis, MN. His research interests are in scientific computing, more specifically, numerical linear algebra and parallel computing. He is a member of SIAM, ILAS, and ACM.