SAND2022-13929C

Remora: A MPI runtime for Composed Applications at Extreme Scale

Noah Evans
Sandia National Laboratories
nevans@sandia.gov

Kevin Pedretti
Sandia National Laboratories
ktpedre@sandia.gov

Shyamali Mukherjee
Sandia National Laboratories
smukher@sandia.gov

Ron Brightwell
Sandia National Laboratories
rbbrigh@sandia.gov

Brian Kocoloski
The University of Pittsburgh
briankoco@cs.pitt.edu

John R Lange
The University of Pittsburgh
jacklange@cs.pitt.edu

Patrick Bridges
The University of New Mexico
bridges@cs.unm.edu

ABSTRACT

As on-node computational power continues to outstrip I/O bandwidth HPC workflows are increasingly composed into communicating processes on the same node.

While MPI provides methods of composition, such as MPMD jobs and intercommunicators, they require significant implementation overhead in the composed applications themselves, either via knowledge of the structure of the composition or via an intimate understanding of the mappings between applications.

This paper describes a new MPI runtime, Remora, currently in progress, which uses the composition primitives of the Hobbes project to make it possible to compose MPI applications in such a way that minimizes implementation overhead. Remora requires minimal changes to the applications themselves, preserving traditional MPI_COMM_WORLD semantics while using intracommunicators instead intercommunicators in order to limit platform specific dependencies.

1. INTRODUCTION

The Message Passing Interface (MPI) has supported composition since it's inception via intercommunicators. However, the lack of intercommunicator wire up and bring down until version 2.0 of the specification means intercommunicators remain in limited use.

Modern MPI runtimes such as [4], [3] also support composition via MPMD, however the traditional implementation of MPI relies on a common MPI_COMM_WORLD which means that composed applications must be aware of each other in the common rank space in order to communicate effectively, an assumption that requires refactoring compared in many MPI applications.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

These two compositional approaches force composed applications to be intimately aware of the nature of the composition being performed, which in turn would require nontrivial effort to implement in MPI simulations that can be millions of lines of code.

This has lead to situation where, dispite having composition mechanisms, there are few composed MPI simulations in practice.

This paper describes a alternative model of MPI composition in progress based on a novel MPI runtime called *Remora* which makes composition of MPI possible with a minimum of invasive changes to prexisting code. Every application maintains its view of MPI COMM WORLD while at the same time providing a new global intracommunicator. The infrastructure provided by this runtime can potentially be a foundation for future MPI runtime composition mechanisms like MPI Sessions [5].

2. COMPOSITION IN MPI

To effectively compose applications using MPI, composition mechanisms must minimize the modifications the individual composed applications as MPI Simulation codes are often millions of lines of code. This means that traditional methods of composing MPI applications are ill suited to the needs of modern composition: MPMD needs ranks to be divided among the composed applications, which entails a certain awareness of the nature of the underlying composition. Intercommunicators are difficult to compose programmatically for more than two applications.

To make composition easier, the ability to compose applications without changing standard interfaces, especially MPI_COMM_WORLD, would allow applications to run while preserving the illusion that they are running in an isolated evironment. Only the actual composition logic would be necessary to allow individual applications to interact in a shared environment.

To enable this sort of composition using MPI we are currently developing *Remora*, a novel MPI runtime built to enable composition.

Remora is built on top of the Hobbes project [2] which makes composition possible by provisioning multiple *enclaves* on a node, each of which consists of a subset of the node's hardware and an internalized OS and runtime stack.

Heretofore, Hobbes has primarily focused on provisioning isolated environments that prevent cross-enclave interference from harming application performance. However, as a result support for cross-enclave communication must be explicitly supported by the underlying enclave environments, and heretofore has been limited to shared memory mappings via the XEMEM system [6].

This paper details novel components of the Hobbes runtime that provide additional support for application composition.

2.1 MPI_COMM_UNIVERSE

One of the foundations of Remora is a new communicator called MPI_COMM_UNIVERSE, a new communicator that replaces MPI_COMM_WORLD as the global communicator of created by the MPI runtime.

Upon initialization, each individual composed application receives its own portion of the MPI_COMM_UNIVERSE rank space which is then renamed to MPI_COMM_WORLD, which in turn allows each application to operate as if it was running in isolation.

This approach allows incremental composition, since unmodified applications operate normally and it is only necessary to add composition logic aware of MPI_COMM_UNIVERSE. This extra logic is free to implement custom composition logic to wire-up for individual applications, including name services and client/server communications.

2.2 Cross OS Composition via MPI

The Hobbes project leverages this incremental implementation to make it possible to run heterogeneous MPI applications, not only can heterogeneous MPI applications be launched, composed and run transparently in enclaves, but different OS and Runtime (OSR) stacks can be run on the same node via pisces co-kernels [**?**].

This makes it possible to run MPI simulation jobs on custom light weight kernels such as Kitten [7] for performance, concurrently with heavier weight kernels such as Linux which provide the full POSIX support for analytics and visualization applications.

2.3 The Implementation Remora Cross OS MPI Runtime

However traditional MPI runtimes like OpenMPI's ORTE are fundamentally bound to POSIX, they assume a full featured POSIX environment during process setup and bring down. This assumption makes it difficult to port MPI to novel architectures.

One of the central contributions of Remora is a custom OpenMPI runtime which minimizes POSIX dependencies while at the same time supporting the Hobbes infrastructure on both Kitten and Linux. This allows for different operating systems to use the MPI library across operating systems. Remora currently runs on both Kitten and Linux.

Most of the work of the runtime, including process startup, resource allocation and teardown are done by per-enclave control processes that execute in each enclave environment and query the Leviathan information service and communicate via XEMEM [6], which removes the kernel from the control plane. Remora utilizes the Vader [8] BTL, which is built on the XPMEM API and thus is supported via XEMEM without modification.

2.4 The Remora Process Management Infrastructure(RPMI) and Process Launcher

The Remora Process Laucher launches composed applications using an XML file which specifies a multilevel topology mapping applications to communicators and enclaves to applications. The MPI_COMM_UNIVERSE is partitioned within the process laucher which sets the relevant PMI values not only to the local MPI_COMM_WORLD rank and size, but the MPI_COMM_UNIVERSE rank and size as well.

This bring-up and tear down is coordinated by a custom PMI implementation on Remora, RPMI. RPMI is implemented using a shared memory database client/server infrastructure, built using facilities provided by Leviathan. RPMI currently supports the PMI1 API [1]. Individual MPI processes query the Leviathan information service in order to discover the global state of MPI applications, both the state of their local MPI_COMM_WORLD and the global MPI_COMM_UNIVERSE.

3. RELATED WORK

Remora explores a very similar problem space as MPI Sessions [5]. MPI Sessions increase MPI scalability by providing individually configured and initialized MPI "sessions" eliminating the need for libraries and composed applications to share an MPI_COMM_WORLD entirely. Remora differs from MPI Sessions by still maintaining a global communicator, but Remora's implementation does not require a global communicator, and could be used as a substrate for implementing session functionality.

4. CONCLUSIONS AND FUTURE WORK

this paper has introduced a novel MPI runtime, Remora, that provides mechanisms enabling the composition of MPI applications on multiple operating systems on the same node. By preserving traditional MPI_COMM_WORLD semantics on a per application basis it minimizes the changes to an MPI application needed to enable composition. Likewise by minimizing the reliance of the runtime on POSIX it can work on many different operating systems unmodified with minimal porting effort. This makes it possible to run MPI on novel operating systems and explore mechanisms such as the Hobbes Node Virtualization Layer.

References

[1] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, J. Krishna, E. Lusk, and R. Thakur. Pmi: A scalable parallel process-management interface for extreme-scale systems. In *European MPI Users' Group Meeting*, pages 31–41. Springer, 2010.

[2] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. E. Bernholdt. Hobbes: Composition and virtualization as the foundations of an extreme-scale os/r. In *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, page 2. ACM, 2013.

[3] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, and A. Lumsdaine. Open mpi: A high-performance, heterogeneous mpi. In *2006 IEEE International Conference on Cluster Computing*, pages 1–9. IEEE, 2006.

[4] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[5] D. Holmes, R. E. Grant, K. Mohror, M. Skjellum, Anthony Schulz, W. Bland, and J. M. Squyres. Mpi sessions: Leveraging runtime infrastructure to increase scalability of applications at exascale. In *Proceedings of the 23rd European MPI Users' Group Meeting*, page 9. ACM, 2016.

[6] B. Kocoloski and J. Lange. Xemem: Efficient shared memory for composed applications on multi-os/r exascale systems. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 89–100. ACM, 2015.

[7] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, et al. Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[8] M. G. Venkata, R. L. Graham, N. T. Hjelm, and S. K. Gutierrez. Open mpi for cray xe/xk systems. *Proceedings of the 2012 Cray User Group, Greengineering the Future. Stuttgart, Germany*, 2012.