Compiled MPI: Cost-Effective Exascale Application Development

Daniel Quinlan, Greg Bronevetsky: Lawrence Livermore National Laboratory,
Andrew Lumsdaine, Indiana University and
Torsten Hoefler, University of Illinois at Urbana-Champaign.

The complexity of Petascale and Exascale machines makes it increasingly difficult to develop applications that can take advantage of them. Future systems are expected to feature billion-way parallelism, complex heterogeneous compute nodes and poor availability of memory. This new challenge for application development is motivating a significant amount of research and development on new programming models and runtime systems designed to simplify large-scale application development. Unfortunately, DoE has significant multi-decadal investment in a large family of mission-critical scientific applications. **Scaling these applications to Exascale machines will require a significant investment that will dwarf the costs of hardware procurement.** A key reason for the difficulty in transitioning today's applications to Exascale hardware is their reliance on explicit programming techniques, such as the Message Passing Interface (MPI) programming model to enable parallelism.

MPI provides a portable and high performance message-passing system that enables scalable performance on a wide variety of platforms. However, it also forces developers to lock the details of parallelization together with application logic, making it very difficult to adapt the application to significant changes in the underlying system.

Further, MPI's explicit interface makes it difficult to describe the application's synchronization and communication structure, reducing the amount of support that can be provided by the compiler and run-time tools. This is in contrast to recent research on more implicit parallel programming models such as X10, Chapel, OpenMP and OpenCL, which promise to provide significantly more flexibility at the cost of reimplementing significant portions of the application.

Compiled MPI (CoMPI) is a novel compiler-driven approach to enable existing MPI applications to scale to Exascale systems with minimal modifications that can be made incrementally over the application's lifetime. It includes:

- A new set of source code annotations, inserted either manually or automatically, that will clarify the application's use of MPI to the compiler infrastructure, enabling greater accuracy where needed,
- A compiler transformation framework that leverages these annotations to transform the original MPI source code to improve its performance and scalability,
- Novel MPI runtime implementation techniques that will provide a rich set of functionality extensions that will be used by applications that have been transformed by our compiler,
- A novel compiler analysis that leverages simple user annotations to automatically extract the application's communication structure and synthesize most complex code annotations.

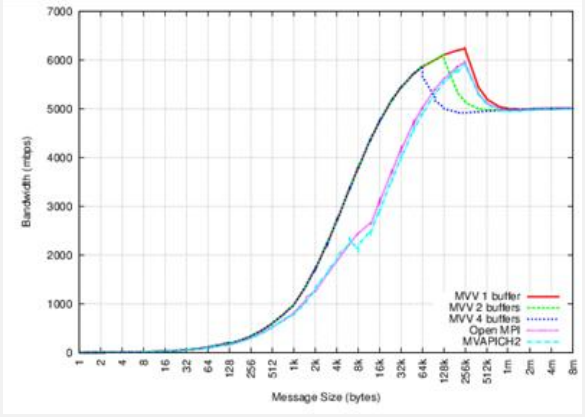
CoMPI is based on the R&D 100 award-winning ROSE compiler and production Open MPI implementation of MPI, enabling it to target complex DoE Applications and become a practical tool.

The final deliverable of this project will be a system that will help legacy MPI applications reach Exascale performance while making only incremental changes to their source code, thus gaining most of the benefits of implicit programming models without most of the costs. By providing developers with unprecedented levels of compiler and runtime support, this system will enable them to focus on new science and to concentrate their reimplementation efforts on the few portions of their applications that require truly new numerical methods rather than new ways of expressing them.

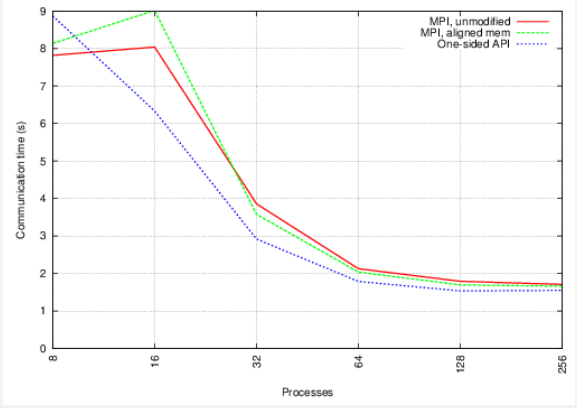
Selected Research Thrusts

Communication Protocols

Our first strategy is to transform two-sided communication routines into faster and potentially more scalable one-sided communications. The following graphs show our initial experiments with multi-versioned variables (MVV) and one-sided communication operations. MVVs can avoid local memory copies at sender or receiver. Avoiding such copies can significantly lower the runtime and power consumption of a large-scale system. The performance benefits of MVVs are shown in the left figure below. The right figure shows how our preliminary optimizations, which transform two-sided communications into one-sided communications over InfiniBand (OFED), improve the communication time of an application kernel (miniMD from the Mantevo benchmarks).



NetPIPE Bandwidth (higher is better)

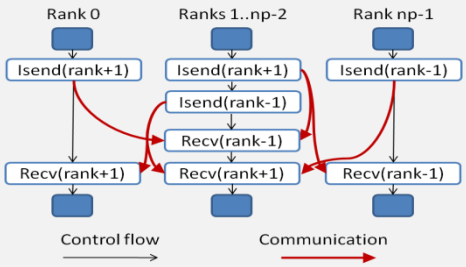


Communication Time in miniMD

Compiler Analyses and Transformations

Communication Structure Analysis:

We are developing a compiler analysis infrastructure to statically match MPI send and receive operations. The analysis abstracts the behaviors of an unbounded number of processes into a few equivalence classes and uses symbolic inference to connect matching send and receive expressions. This analysis will be used to reduce the cost of receive matching and to detect collective communication patterns and replace them with implementations optimized for the target platform.



Send-Receive Fusion

If a send and receive operation are executed in the same shared memory domain, their data packing and unpacking code can be fused so that data is copied directly from the sender's to the receiver's data structure. We are developing a compiler analysis to perform this fusion.

Original Code		Transformed Code

for(i=0; i<numAtoms; i++)
  if(borderAtom(i)) {
  buf[len] = atoms[i];
  len++;
  totalAtoms--;
  }
MPI_Send(buf, len);
	MPI_Recv(buf, status);
	MPI_Get_count(status, &len)
	for(j=0; j<len; j++) {
	  atoms[totalAtoms] = buf[j];
	  totalAtoms++;
	}
		for(i=0; i<numAtoms; i++)
		  if(borderAtom(i)) {
		  remote->atoms[remote->totalAtoms] = atoms[i];
		  remote->totalAtoms++;
		  len++;
		  totalAtoms--;
		  }