CoMPI – Configurable Collective Operations in LAM/MPI

Espen Skjelnes Johnsen, John Markus Bjorndalen, and Otto J. Anshus

Department of Computer Science, University of Tromsø

1 Introduction

This paper describes an extension to LAM/MPI[3] which enables the user to configure a subset of the collective operations by using Scheme[5], which is a high level general purpose programming language in the Lisp family.

Currently the operations that may be configured are broadcast, reduce, allreduce and barrier, but the system is general enough to be extended with other operations if that is required. Sophisticated reconfigurations can be done without modifying or recompiling the application that is to be run.

We will show that high level languages and Scheme in particular, could be used in high performance computing despited the general opinion that such languages are slow, and that this would be an improvement compared to the more conventional way of writing such extensions in low level languages like C, due to the fact that a lot of details could be hidden or abstracted away.

We will also use CoMPI to show that reconfiguration of collective operations in MPI could result in better scaling when running in multi-cluster[1] environments.

The goal of CoMPI is to provide a system which is simple to use and to provide a framework for general experimenting with collective operations in LAM/MPI. It was largely inspired by the PATHS[2] system which uses Python to set up an runtime environment for parallel applications.

Compiled Communication[4] (CC-MPI) describes a method to optimize cluster communication by using information available at compile time by using a modified C compiler. CC-MPI and CoMPI share some of the same basic ideas about doing as much optimization as possible before the communication takes place. In CC-MPI this happens when the application is compiled, while in CoMPI when an opertion is invoked for the the first time.

2 Architecture of CoMPI

The idea behind CoMPI is that when a collective operation is invoked, a hash table lookup is done to check if a suitable coll-op closure[2] exists to perform the

[1] A multi-cluster is in our definition a collection of clusters connected by high latency/low bandwidth WAN links.

[2] A *closure* in Lisp terminology a closure is an object containing a function and the lexical environment in which the function was created.

given operation. If no such closure exists, a coll-op creator is invoked to create one. The newly created coll-op closure would then be invoked and also cached for reuse later.

Each supported collective operation needs it's own set of unique coll-op creators. By default, creators are supplied to mimic the semantic behavior of native LAM. The most primitive way to configure operations, would be for the user to install new coll-op creators which will return specialized coll-op closures. Coll-op creators are invoke in reverse order of installation (latest invoked first) until one of them returns a closure. They may also be chained by having one creator calling the next in the list. Chaining may be useful to create wrappers for profiling or debugging.

The interface between the MPI API layer and CoMPI consists of functions called *coll-op trampolines* which are invoked by MPI when the application does a collective operation. The trampolines are responsible for looking up and invoking coll-op closures.

3 CoMPI Advantage

One of the advantages with CoMPI is the possibility to configure and experiment with MPI without having to recompile applications. In some cases the user may not even have the source code to the application he is running. LAM/MPI already has some options which may be used at runtime to configure and customize the environment, but the strength of CoMPI lies in the use of a high level programming language. By not having think about low level details such as memory management, the task of trying out different configuration may become much simpler than if he were to use a language like C. Also by not having to stick with a limited customization language, but having access to a general purpose language give the system great flexibility. For simpler configurations CoMPI comes with a set of coll-op creators which let the user specify the network topology based on node ranks or IP numbers.

4 Experiments

A simple experiment were conducted to show that the performance of CoMPI is comparable to that of native LAM when using the same algorithms, and that the choice of implementation language did not have a negative impact on performance. In addition we're running the same benchmark with a communication pattern configured to match the underlying network layout.

The hardware platform used for these experiments consists of three interconnected clusters, each having the following configuration:

- 28 Dell 370 P4 Prescott, 3.2 GHz, 2 GB RAM
- 38 Dell 370 Prescott 64 EMT, 3.2 GHz, 2 GB RAM
- 44 Dell 360 Prescott 3.2 GHz, 2 GB RAM

The nodes of the clusters and the front-ends are interconnected using TCP/IP over Ethernet through 1Gbit switches.

Each cluster are using IP addresses from the private 10.0.0.0 range and are not reachable from outside of the front-ends. To allow routing between nodes from different clusters, IP tunnels are set up between the front-ends. This add to the inter-cluster latency, increasing it from 100 to 300-500 microseconds, which are taken advantage of to emulate WAN connections.

Figure 4 show the code run in the experiment. The code measure the average execution time of 1000 `MPI_Allreduce` operations, with buffer sizes for 1 to 50000 elements. To make sure that the correct sum is computed, the code also checks the result of each iteration. This is the same benchmark as used in [1].

```
t1 = get_usecs ();
for (i = 0; i < ITERS; i++) {
  MPI_Allreduce (&i, &ghit, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
  if (ghit != (i* size))
    printf ("oops at %d. %d != %d\n", i, ghit, (i * size));
}
t2 = get_usecs ();
```

Figure 1. Global reduction benchmark

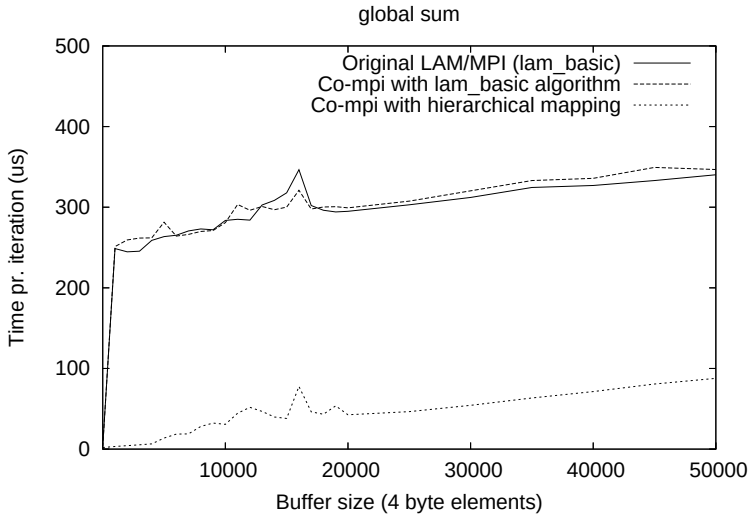

Figure 2. Results of the global sum experiments

Figure 2 shows the result of running the global sum benchmark. The following experiments were done:

Native LAM/MPI (`lam_basic`) The experiment was run using LAM/MPI's native logarithmic broadcast and reduce trees.

Co-mpi using the `lam_basic` algorithm and tree used above The experiment was run using CoMPI with the same algorithm as `lam_basic`.

Co-mpi with hierarchical mapping In this experiment we used a hierarchical mapping which minimize the inter-cluster communication. This mimics asymmetric algorithm in used by MagPIe [6].

As the figure shows, the overhead of CoMPI is insignificant compared to native LAM when both systems use the same communication pattern. With a hierarchical mapping, the performance of CoMPI is significant better than of LAM.

5 Conclusion

We have shown that high level scripting languages are well suited for use in the infrastructure of high performance clusters by using such a language to implement a collective operations module for LAM/MPI.

We have created a flexible framework, CoMPI, which may be used for experimentation and optimization of collective operation in LAM/MPI without modifying existing applications.

When running experiments the overhead observed by using CoMPI is negligible compared to `lam_basic`. By reconfiguring the communication pattern to match the actual network layout substantial increase in performance were achieved.

References

1. Bjørndalen, J. M., Anshus, O., Vinter, B., and Larsen, T. The Performance of Configurable Collective Communication for lam/mpi in clusters and multiclusters. *NIK 2002, Norsk Informatikk Konferanse, Kongsberg, Norway* (November 2002).
2. Bjørndalen, J. M., Anshus, O., Larsen, T., and Vinter, B. Paths – integrating the principles of method-combination and remote procedure calls for run-time configuration and tuning of high-performance distributed application. *Norsk Informatikk Konferanse* (November 2001), 164–175.
3. Burns, G., Daoud, R., and Vaigl, J. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium* (1994), pp. 379–386.
4. Karwande, A., Yuan, X., and Lowenthal, D. K. Cc–mpi: a compiled communication capable mpi prototype for ethernet switched clusters. In *PPoPP '03: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming* (2003), ACM Press, pp. 95–106.
5. Kelsey, R., Clinger, W., and (Editors), J. R. Revised^5 report on the algorithmic language Scheme. *ACM SIGPLAN Notices 33*, 9 (1998), 26–76.
6. Kielmann, T., Hofman, R. F. H., Bal, H. E., Plaat, A., and Bhoedjang, R. A. F. MagPIe: MPI's collective communication operations for clustered wide area systems. *ACM SIGPLAN Notices 34*, 8 (1999), 131–140.