Scalable Runtime for MPI: Efficiently Building the Communication Infrastructure

George Bosilca[1], Thomas Herault[1], Pierre Lemarinier[2], Ala Rezmerita[3], and Jack J. Dongarra[1]

[1] University of Tennessee, Knoxville
[2] Université de Rennes 1, IRISA
[3] Grand-Large, INRIA Saclay – Université Paris-Sud

1 Introduction and Motivation

The runtime environment of MPI implementations plays a key role to launch the application, to provide out-of-band communications, enabling I/O forwarding and bootstrapping of the connections of high-speed networks, and to control the correct termination of the parallel application. In order to enable all these roles on a exascale parallel machine, which features hundreds of thousands of computing nodes (each of them featuring thousands of cores), scalability of the runtime environment must be a primary goal.

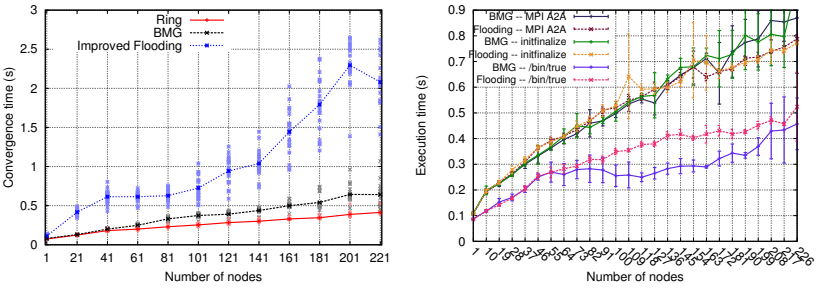
In this work, we focus on an intermediate level of the deployment / communication infrastructure bootstrapping process.We present two algorithms: the first to share the contact information of all runtime processes, enabling an arbitrary set of connections, and the second to distribute only the information needed to build a binomial graph. We implemented these two algorithms in ORTE, the runtime environment of Open MPI, and we compare their efficiency, one with the other, and with the runtime systems of other popular MPI implementations.

2 Evaluation

We use the underlying launching tree to exchange contact information at the runtime level, and let the runtime system build for itself a communication infrastructure mapping a binomial graph [1] topology. This topology has several interesting properties such as redundant links to support fault tolerance and binomial tree shape connectivity rooted in each peer. A precedent work [3] proposes a self-stabilizing algorithm to build such an overlay on top of a tree. Such an algorithm provides two main features: 1) inherent fault-tolerance and 2) self-adaptation to the underlying tree topology, which negates the need for initializing the construction of the binomial topology.

We compare our implementation with three other setups: the implementation of ORTE described in [4] (prsh with improved flooding), MPICH2 [5] version 1.3.2p1 using Hydra [6,2] with rsh and MVAPICH2 version 1.7a using the ScELA [7] launcher. All versions are compiled in optimized mode and the experiments based on rsh were using ssh as a remote shell system.

Y. Cotronis et al. (Eds.): EuroMPI 2011, LNCS 6960, pp. 342–344, 2011.



(a) Contact Information Exchange Time

(b) Scalability

Fig. 1. Comparison of ORTE prsh with BMG and ORTE prsh with Flooding

First, we compare the two implementations in ORTE together, in the figures presented in Fig. 1. The first micro benchmark, presented in Fig. 1a measures the time taken to solely exchange the Contact Information, following the Improved Flooding Strategy, or the BMG Building strategy, as functions of the number of nodes. The latter consists in two phases: first the building of the ring, then of the BMG, and the two phases are represented in the figure. Individual measurements are represented with light points, and mean values are connected with a line.

The BMG algorithm presents a better convergence time, in practice, than the Improved Flooding algorithm. This is expected, since it exchanges much less information (each node receives only the contact information of $O(\log_2(n))$ nodes) than the Flooding algorithm ($O(n)$). The ring construction time occupies a major part of this time, but the system appears to scale faster than linearly.

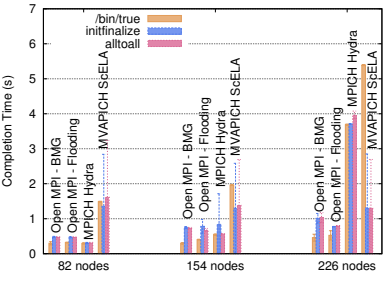


Fig. 2. Comparison with other MPI runtime systems

This is also demonstrated in Fig. 1b, which presents how both implementations perform when increasing the number of nodes. On the `/bin/true` benchmark, the BMG construction algorithm demonstrates a better scalability than the Flooding Algorithm, with noticeable steps that characterize the logarithmic behavior of the algorithm. This logarithmic behavior disappears, when launching a communicating MPI application, like `a2a`, or even a simple empty MPI application, like `initfinalize`. This is due to the third phase of the launching in ORTE, the modex, that introduces a linear component to the performance.

Fig. 2 compares the two ORTE implementations with Hydra (MPICH2), and ScELA (MVAPICH) for the three benchmarks, and various number of nodes. Although Hydra performs slightly better than both ORTE implementations at a small scale, ORTE reaches the same performance for 154 nodes and above.

After about 166 nodes, both Hydra and ScELA for the `/bin/true` benchmark suffer from connections storms, that impact the performance by introducing a delay of 3s, due to TCP SYN packets retransmission.

3 Conclusion

In this paper, we presented two strategies for the construction of a runtime communication infrastructure running in parallel with the deployment of the runtime processes and the deployment of the parallel application. The first strategy uses an improved flooding algorithm, that enables any runtime process to communicate with any other directly, thus providing an arbitrary routing topology for the runtime. The second strategy uses an ad-hoc self-adapting algorithm, that transforms the initial spawning tree into a binomial graph, not only sharing the needed contact information (and only this information), but also establishing at the same time the corresponding links. We implemented both algorithms in ORTE, the runtime system of Open MPI, and compared the implementations with the state of the art runtime environments for MPI. Experiments demonstrated an improved scalability, highlighting the importance of tight integration between launching and communication infrastructure construction, and the advantages of a flexible routing topology at the runtime level.

References

1. Angskun, T., Bosilca, G., Vander Zanden, B., Dongarra, J.: Optimal routing in binomial graph networks. In: Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2007, pp. 363–370 (December 2007)
2. Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Krishna, J., Lusk, E., Thakur, R.: PMI: A scalable parallel process-management interface for extreme-scale systems. In: Keller, R., Gabriel, E., Resch, M., Dongarra, J. (eds.) EuroMPI 2010. LNCS, vol. 6305, pp. 31–41. Springer, Heidelberg (2010)
3. Bosilca, G., Coti, C., Herault, T., Lemarinier, P., Dongarra, J.: Constructing resiliant communication infrastructure for runtime environments. Advances in Parallel Computing 19, 441–451 (2010), doi:10.3233/978-1-60750-530-3-441
4. Bosilca, G., Herault, T., Rezmerita, A., Dongarra, J.: On scalability for mpi runtime systems. In: IEEE International Conference on Cluster Computing (to appear, 2011)
5. Mathematics, and Computer Science Division, A. N. L. MPICH-2, implementation of MPI 2 standard (2006), `http://www-unix.mcs.anl.gov/mpi/mpich2/`
6. Mathematics, and Computer Science Division, A. N. L. Hydra process management framework (2009), `http://wiki.mcs.anl.gov/mpich2/index.php/HydraProcessManagementFramework`
7. Sridhar, J., Koop, M., Perkins, J., Panda, D.: ScELA: Scalable and extensible launching architecture for clusters. In: Sridhar, J., Koop, M., Perkins, J., Panda, D. (eds.) HiPC 2008. LNCS, vol. 5374, pp. 323–335. Springer, Heidelberg (2008)