Tracing Data Movements within MPI Collectives

Kevin A. Brown, Jens Domke and Satoshi Matsuoka
Tokyo Institute of Technology
2-12-1-W8-33, Ookayama,Meguro-ku,
Tokyo 152-8552, Japan
{brown.k.aa, domke.j.aa}@m.titech.ac.jp, matsu@is.titech.ac.jp

ABSTRACT

We propose extending common performance measurement and visualization tools to identify network bottlenecks within MPI collectives. By creating additional trace points in the Peruse utility of Open MPI, we track low-level InfiniBand communication events and then visualize the communication profile in Boxfish for a more comprehensive analysis. The proposed tool-chain is non-intrusive and incurs less than 0.1% runtime overhead with the NPB FT benchmark.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Performance measures*

Keywords

Performance analysis, MPI, Peruse, profiling

1. INTRODUCTION

Supercomputers are rapidly growing in size and their networks are simultaneously growing in complexity. With this trend, inter-process communication becomes a significant performance factor, especially with communication bound programs [1]. The de facto standard for inter-process communication on these systems, the Message Passing Interface (MPI) [7], abstracts the hardware layer from the application; this makes it difficult to explicitly link application performance to network activity.

Unlike process-centric tools like Scalasca and Vampir, Boxfish [3] uniquely circumvents this abstraction by including network utilization metrics in application performance analysis. Nonetheless, monitoring network activity within collectives and other complex routines remains challenging since tracing and profiling is done in the application layer.

We present a method for conducting performance analysis of collective communications running on complex networks. This involves adding new trace points into Peruse [4], a performance revealing extension in Open MPI. The new events, which track point-to-point network events within MPI, are recorded using a non-intrusive profiling library that we developed. Finally, we create a new visualization module for Boxfish to reveal possible network bottlenecks in applications running on InfiniBand-based fat-tree networks.

2. RELATED WORK

Landge et. al [6] described how to capture the flow of network packets on IBM Blue Gene/P for tracing application communication over network links. However, the tracking of network packets was done by capturing port counters at preset points during the application run. Unlike our approach, this method is impractical for systems running multiple applications simultaneously. Additionally, their method is limited to torus networks while ours can be used on any InfiniBand-based network.

3. DESIGN

3.1 Extending Open MPI

Peruse facilitates the tracking of internal events within MPI communication operations by linking a user-supplied callback function to those events. We added a new Peruse event, `PERUSE_OPENIB_SEND`, to Open MPI's byte transfer layer (`btl`) for tracking data sent via InfiniBand network ports. The event is placed immediately after each InfiniBand `ibv_post_send` command is issued, thus allowing us to overlap the communication with our callback function in order to reduce profiling overhead.

3.2 Profiling an Application

We built a shared library that registers a callback function with our Peruse event, captures the amount of data sent to each remote network adapter within the callback function, and writes the aggregated information to the OTF files [5]. Our library supports the of use environment variables to define which collective(s) should be tracked. If no collectives are specified, the library measures point-to-point network traffic generated by the entire application. By preloading our library with the MPI application using `LD_PRELOAD`, we are able to generate detailed communication profiles without modifying the application's source code.

3.3 Visualization

A python script was written to: (a) parse the files containing the list of port connections (`ibdiagnet.lst`) and the unicast forwarding tables (`ibdiagnet.fdbs`), which are both

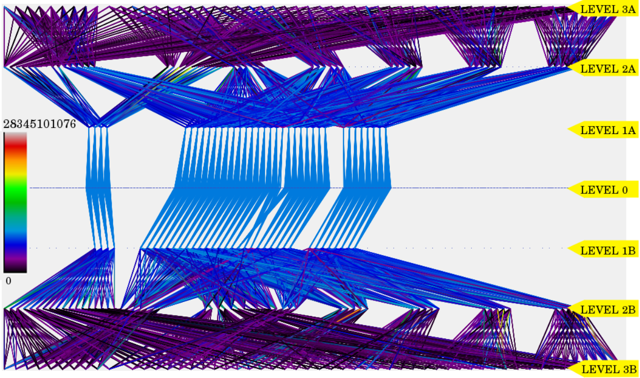

Figure 1: Boxfish visualization of TSUBAME2.5's network traffic caused by the NPB FT benchmark.

generated by the InfiniBand `ibdiagnet` command; (b) parse the OTF profiles written by the application; and (c) build a connected network using port connection data. Weights are added to the network links based on information in the profiles and tracing the traffic paths using the network's forwarding tables. After this point, the Boxfish visualization data is generated.

Fig. 1 shows the profile visualization of the FT benchmark from the NAS parallel benchmark suite (NPB) with problem size E running on 512 nodes of TSUBAME2.5. Compute nodes are positioned horizontally at level 0 while switches are shown at the other levels. TSUBAME2.5 uses two subnets and each node has a connection to both subnets. Each line drawn between the different levels represent a link connecting a switch to another switch or to a compute node. Link colors, which indicate network traffic, are based on the color-value map shown on the left of the figure. Red links indicate traffic hotspots, i.e., points of possible bottlenecks. For readability, links with no traffic are not shown.

4. OVERHEAD MEASUREMENT

We used the TSUBAME-KFC system at the Tokyo Institute of Technology for our overhead measurement experiments. Each of the 40 compute nodes is connected to one of two InfiniBand FDR switches, which share a 15 link interconnect. Tests were carried out on 32 nodes with no other user process running on the network. The results presented in this section do not include the time for writing output files, which was approximately 13 ms for each test case.

An `MPI_Alltoall` microbenchmark was used with message sizes in the range of 0 - 32 kB. 30 profiled trials and 30 unprofiled trials were ran for each message size, with each trial comprising of 20,000 iterations (2 initialization runs + 19,998 timed runs). The minimum of the average runs for each trial was used for the resulting value since this would be the most reproducible result [2]. All timing values were measured on process 0 using a combination of `MPI_Barrier()` and `gettimeofday()`. These results are shown in Fig. 2.

The maximum overhead for the `MPI_Alltoall` call is 4.08% when the message size is 256 bytes. We attribute the variations in overhead across the different message sizes to the

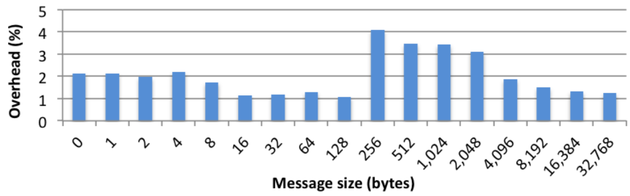

Figure 2: Chart showing the overhead when profiling MPI_Alltoall with various message sizes

changes in the protocol being used when different size thresholds are reached. Most notable, messages larger than 12 bytes and less than 256 bytes use send/receive semantics, while smaller messages use the RDMA eager protocol. Messages larger than 256 bytes use the RDMA pipeline protocol.

The communication bound FT benchmark from the NAS Parallel Benchmark (NPB) suite was also used. We also ran 30 profiled runs and 30 unprofiled runs for this benchmark with the class C problem size. Again, we took the minimum of the average runtime for each trial. The communication overhead is a negligible 0.0205% of the application's overall runtime, i.e., an increase from 12.1849 secs to 12.1874 secs.

5. SUMMARY AND FUTURE WORK

By profiling network communication using a modified Peruse interface in Open MPI, we enabled the monitoring of low-level network-wide events during MPI collective communication. A non-intrusive profiling library and parsing tool were developed to record and process these events, respectively. We showed how the visualization of our profile information in Boxfish allows for the identification of potential network bottlenecks within applications. Our profiling library does not require instrumentation or recompilation of the user application and incurs only 0.0205% overhead when tested with the NPB FT benchmark.

Our next step is to implement this functionality using the MPI Tools interface in Open MPI and other MPI libraries.

6. REFERENCES

[1] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap. In *Proc. of IPDPS*, 2006.

[2] W. Gropp and E. L. Lusk. Reproducible Measurements of MPI Performance Characteristics. In *Proc. of Euro PVM/MPI*, 1999.

[3] K. E. Isaacs, A. G. Landge, T. Gamblin, P.-T. Bremer, V. Pascucci, and B. Hamann. Exploring Performance Data with Boxfish. In *Proc. of SC Comp.*, 2012.

[4] R. Keller, G. Bosilca, G. Fagg, M. Resch, and J. J. Dongarra. Implementation and Usage of the PERUSE-Interface in Open MPI. In *Proc. of Euro PVM/MPI*, 2006.

[5] A. Knüpfer, R. Brendel, H. Brunst, H. Mix, and W. E. Nagel. Introducing the Open Trace Format (OTF). In *Proceedings of the 6th International Conference on Computational Science - Volume Part II*, ICCS'06, pages 526–533, 2006.

[6] A. Landge, J. Levine, A. Bhatele, K. Isaacs, T. Gamblin, M. Schulz, S. Langer, P.-T. Bremer, and V. Pascucci. Visualizing Network Traffic to Understand the Performance of Massively Parallel Simulations. *IEEE Trans. on Vis. and Computer Graphics*, Dec 2012.

[7] MPI Forum. `MPI`:A Message-Passing Interface Standard. http://www.mpi-forum.org/, Mar 2014.