SAND2007-4804C

Seshat Collects MPI Traces: Extended Abstract

Rolf Riesen*

Sandia National Laboratories
rolf@sandia.gov

Traces collected at the MPI level can help understand the behavior of applications by using these traces to visualize the communication patterns of an application. The traces can also be used for debugging and as input to system simulators. These trace driven simulators can help with learning how an application makes use of the communication fabric and how an application will perform on a next-generation machine.

Tools that collect MPI traces have one these two drawbacks: They do not produce accurate enough, fine grained traces, or they distort the application behavior and run time. Tools that generate detailed traces influence the run time, and sometimes the behavior, of the application under measurement, because the amount of data collected is large and requires time to send to storage [1]. The timestamps in the trace data are influenced as well. Tools that are less intrusive, collect less, or less accurate, information [2,3,4]. We propose a method to solve both of these problems.

Some users of trace data are interested in the message data itself. For example, a trace driven simulator that simulates one node of a parallel application needs to feed the process on that node valid data. Otherwise the process might not behave in the same way as it would outside the simulator, when it is running as part of a parallel application. Collecting the application data of every MPI message during an application run generates enormous trace files and greatly influences the timing of an application.

This extended abstract describes a tool named Seshat[1] [5] which we have extended to allow tracing of MPI applications. Seshat is an execution-driven network simulator with a feedback channel into the application that it uses to update the virtual time the application is running in. It is written as a library that is linked with an MPI application. No instrumentation of the application code is necessary; relinking it with Seshat is enough. Seshat makes use of the profiling interface that is part of the MPI standard (PMPI). With hooks into most of the MPI calls, Seshat is able to initialize itself, collect information about the running application, and adjust the application's virtual timeframe.

The application runs as before, but on an additional node runs the Seshat network simulator. All MPI messages are sent and received as before, but they also generate events that are sent to the simulator. The simulator calculates the time this message was supposed to spend in the simulated network and informs the receiver how much virtual time has elapsed since the message was sent. The receiver uses that information to update its local virtual clock. In effect, we can simulate a different network than the one we are running on, and use Lamport's time synchronization mechanism [6] to keep the application unaware of the wall-clock time.

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

[1] Seshat was the Egyptian goddess of measurement and recording.

This independence of the time system the application runs in, plus the knowledge about every single message of the application under test, allows the network simulator to generate MPI traces. For proof of concept we built a prototype that writes a 90-byte ASCII text line to a file for every message event that arrives at the network simulator. Writing this data slows down the simulator when measured in wall-clock time. However, the virtual time-frame the application is running in, has not changed. We have conducted several experiments to test this hypothesis.

We ran experiments on Sandia's Cray XT3^{TM} Red Storm machine. It was running version 1.5.39 of the Cray system software. The performance characteristics of that software release are also the ones Seshat simulates. We used the NAS parallel benchmarks version 3.2.1 to verify our claims. These benchmarks are simple compared to real applications. However, we are only interested in a proof of concept. Any code that sends and receives a large number of MPI messages will do. In some tests we write so much information that we slow down the benchmark so it takes several hours of wall-clock time to complete. Yet, it reports the same few seconds or minutes of (virtual) run time as it would when run natively.

When tracing is enabled, the LU, class A benchmark reports (virtual) times that are within 6% of what it reports when run in native mode. (In most cases within 2%.) The wall-clock time, however, is 2,600 times higher than the 64-node native run. This could be lowered by making the Seshat network simulator parallel and have it write to a high-performance file system. The fact that this simple experiment, writing to an NFS-mount file system from a single simulator node, does not change the time the LU benchmark reports, shows that our approach works. It will let us collect huge traces that take a long time to write to stable storage, without changing the time-related behavior of the application under test.

Unfortunately, there is is currently a bug in Seshat that prevents it from performing as well as LU for some of the other NAS parallel benchmarks. The CG benchmark, for example, reports widely different times when attached to Seshat, then when it runs natively. We know this is a Seshat virtual time bug, and is not due to tracing. Although the virtual time reported when CG runs under Seshat is wrong, it does not change when we enable tracing. That means our mechanism for collecting large traces without influencing the application works. We are investigating the problem in Seshat's virtual time routines and will fix it.

1. Chung, I.H., Walkup, R.E., Wen, H.F., You, H.: MPI performance analysis tool on Blue Gene/L. In: Proc. IEEE/ACM SuperComputing, Tampa, FL (November 2006)
2. Vetter, J.S., Yoo, A.: An empirical performance evaluation of scalable scientific applications. In: Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, Los Alamitos, CA, USA, IEEE Computer Society Press (2002)
3. Noeth, M., Mueller, F., Schulz, M., de Supinski, B.R.: Scalable compression and replay of communication traces in massively parallel environments. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS). (2007)
4. Knüpfer, A., Nagel, W.E.: Compressible memory data structures for event-based trace analysis. Future Generation Computer System **22**(3) (2006) 359–368
5. Riesen, R.: A hybrid MPI simulator. In: IEEE International Conference on Cluster Computing (CLUSTER'06). (2006)
6. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (1978) 558–565