Scalable client-server visualization of callpath traces for large-scale parallel executions

Philip Taffet*
ptaffet@rice.edu
Dept. of Computer Science, Rice University
P.O. Box 1892, Houston, TX 77005-1892

Introduction As supercomputers grow in scale, performance analysis tools must also scale. Prior work on the HPCToolkit performance tools has shown the utility of analyzing traces of call stack samples of parallel program executions [3]. HPCToolkit's `hpctraceviewer` graphical user interface turns gigabytes of such trace data into actionable visual insights about an application's behavior and performance bottlenecks. `Hpctraceviewer` displays information about the execution by using sampling to project data in the process/thread and time dimensions. Since traces can be very large, the scalability of `hpctraceviewer` is an important concern.

Approach I extended HPCToolkit's `hpctraceviewer` graphical user interface to introduce an optional client-server model to improve handling for large traces. In this new model, the `hpctraceviewer` GUI client can renders data provided by `hpctraceserver`, which runs as an MPI application and processes traces in parallel for the GUI. The client and server communicate over a TCP connection, which can be tunneled through SSH for security if necessary.

When the user zooms or pans in `hpctraceviewer`, the client sends a windowing command to the server defining the new viewing region. MPI rank 0 of `hpctraceserver`, which manages communication with the client on the TCP link, receives the windowing command and broadcasts it to all MPI ranks. Each MPI rank determines its assigned portion of a data projection operation based on its rank, computes and optionally compresses its projections, and forwards them to MPI rank 0 for transfer to the client over the TCP link.

The `hpctraceviewer` client, which implemented in Java, receives a stream of trace lines data from the server and then employs Java threads to decompress, and render the projections it receives in parallel. This enables computation on the server to overlap with communication and computation on the client.

Because the server sends call path identifiers (see [3]), which represent an entire call path, rather than a visualization of the data, client-side visualization operations such as changing the depth of the view, selecting a different process for display in the depth view, and changing the method to color mapping can be performed entirely on the client and without further additional information from the server.

There are three primary benefits to client-server approach:

First, using the server enables us to leave trace data on the high-performance parallel file system of a remote supercomputer instead of transferring it all to the client. The network traffic with the server required to display projections depends only on the size of the client's window and is less than 2MB for typical desktop and laptop displays.

Second, `hpctraceserver` computes trace line projections in parallel. Using p MPI ranks for the server improves the time complexity to display a region of the trace that spans time t in a window h pixel high and w pixels wide from $O(hw \log t)$ to $O(\frac{hw}{p} \log t)$. The server partitions the client's request among the p server ranks along the thread dimension. Each server rank maps several contiguous regions of the trace file into memory, one for each process trace line it will handle.

Third, `hpctraceserver` benefits from a supercomputer's large memory. Sampling trace lines to compute their projections uses binary search along a trace line to locate the trace records needed. Using `mmap`, we access data in RAM rather than seeking and reading from a file. Although `hpctraceserver`

*This work was supervised by Dr. Laksono Adhianto and Prof. John Mellor-Crummey at Rice University.

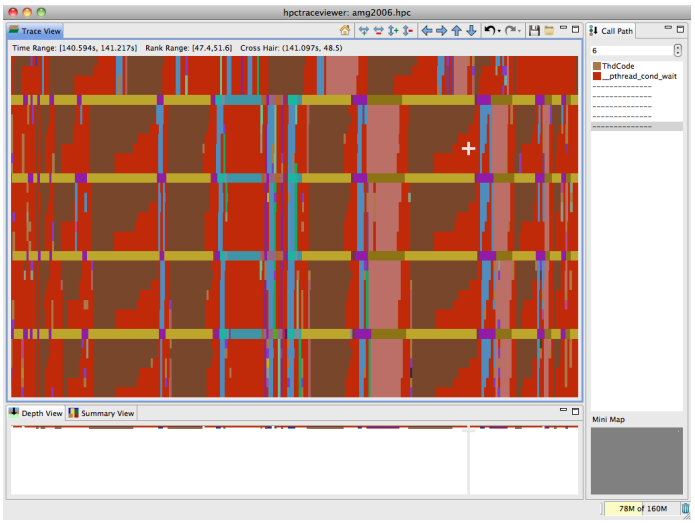

Figure 1: An execution detail from the AMG2006 benchmark running on Argonne's Mira using 1024 processes × 8 OpenMP threads. The trace database is 4.97 GB. Rendering a 2560x1567 home view of this database with `hpctraceviewer` running on a laptop at Rice University and `hpctraceserver` running on 8 cores (0.1% execution size) of Tukey at Argonne took 8.63s. Rendering this view took 1.03s. Rendering these views on a quad-core laptop took 75.71s and 1.96s, respectively. The red stair-step patterns indicate load imbalance. Our new client-server approach enables us to analyze long executions for much larger numbers of processors.

and `hpctraceviewer` (when analyzing local data) use mapped memory, laptops and workstations that run `hpctraceviewer` typically have far less RAM and must perform disk IO more frequently. Additionally, while `hpctraceviewer` requires access to the entire database of collected data, no single MPI rank in `hpctraceserver` needs to `mmap` the entire file. `hpctraceserver` uses a LRU algorithm to map and unmap pages of the file as needed.

Related Work The Vampir performance toolset also supports a client-server decomposition [2]. `Hpctraceserver`'s sampling-based approach for computing trace projections is more lightweight. Vampir presentations recommend running VampirServer on 10% of the number of monitored processes [1].

Summary By parallelizing projections and rendering of performance data, our client-server enables analysis of remote execution traces for large executions.

References

[1] J. Domke. Titan Workshop - Vampir and VampirTrace: Trace based performance analysis at large scale on Titan, Oak Ridge National Laboratory, January 2012. `http://www.olcf.ornl.gov/wp-content/uploads/2012/01/TitanWorkshop2012_Day3_Vampir.pdf`.

[2] A. Knüpfer et al. The Vampir performance analysis tool-set. In R. Keller, V. Himmler, B. Krammer, and A. Schulz, editors, *Proceedings of the 2nd Intl. Workshop on Parallel Tools for High Performance Computing*, Tools for High Performance Computing, pages 139–156. Springer Verlag, Berlin Heidelberg, 2008.

[3] N. R. Tallent, J. Mellor-Crummey, M. Franco, R. Landrum, and L. Adhianto. Scalable fine-grained call path tracing. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 63–74, New York, NY, USA, 2011. ACM.