# HPCToolkit

hpctoolkit.org

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

Philip Taffet, Dept. of Computer Science, Rice University, ptaffet@rice.edu

## Introduction and Motivation

**Producing scalable applications requires the ability to diagnose performance bottlenecks**

- Asynchronous sampling enables tools to profile large-scale applications without severely perturbing execution
- Yet, data size is typically proportional to the number of cores used in the execution
  - Traces of executions of >10k cores often produce measurement databases >10 GB
- Need tools that can:
  - Distill collected data into actionable feedback
  - Support interactive post-mortem analysis of execution traces to analyze time-varying behavior of large-scale parallel executions on supercomputers

## Challenges

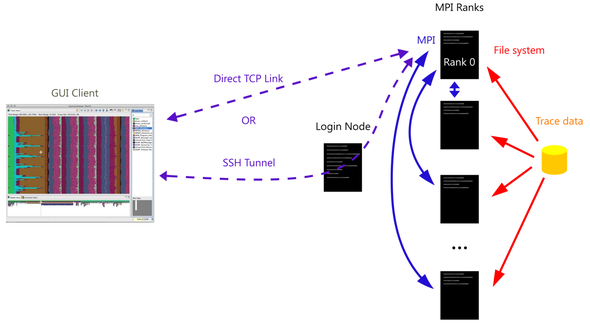
- Sampling large execution traces to compute a projection for visualization is disk and CPU intensive
- Large execution traces may not fit in the memory of a single node
- Using a server-side application and displaying traces on a remote system using X11 is slow due to high network latency

## Goals

- Employ parallelism to support interactive visualization of large execution traces
  - On supercomputer to compute a projection of the data to render
  - On client to render traces in a graphical user interface
  - Lower the time complexity from $O(hw \log t)$ to $O(hw/p \log t)$
- Keep execution traces on the supercomputer where they are collected
  - Avoid transferring multi-GB traces to the client

## Our Approach

- Refactor HPCToolkit's hpctraceviewer visualization tool to support an optional trace server that runs on a remote supercomputer
- Trace server receives windowing commands from GUI client and streams back compressed trace projections over a TCP link
- Trace server computes projections of trace data needed by GUI client
- Trace server can run single-threaded, or as an MPI job to compute projections of large traces in parallel
- Once a trace projection is received from the server, the GUI client can render it at different levels of abstraction (i.e., call stack depth) without further data from the trace server




## Parallelism

- Projections for trace lines to be rendered can be computed and compressed for transfer independently of one another
- Results are sent to a master process, which transfers them to the GUI client over a TCP link
- Client decompresses and renders each projection as it is received, in parallel

## Compression

**Compression reduces the necessary communication bandwidth**

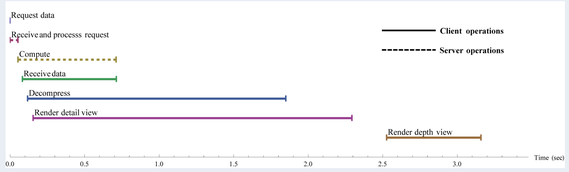
- Must transfer two types of data to client
  - XML representation of calling context tree
  - Binary representation of trace line projections
- Compression Results
  - XML: gzip-compresses to < 10% of its original size
  - Binary trace lines
    - Two components per entry: 4-byte call path identifier, 8-byte timestamp
    - Delta-encode timestamps to 4 bytes (50% of original)
    - Use DEFLATE algorithm for compression
    - Result: ~3 bytes/entry, 25% of original size

## Filtering

- In hybrid programs (MPI+OpenMP) and others, not all ranks/threads are comparable (e.g., MPI communication helper threads)
- Examining a homogenous subset can facilitate understanding of behavior
- Approach:
  - Server provides metadata to client for all traces when a connection is established
    - Only tens of bytes per trace line
  - Server-side filtering selects trace data of interest for transfer
    - filter by range and/or stride for process number and thread number
      e.g., 2:100:4 hides every 4th thread from 2 to 100

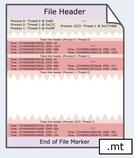
## Communication Overlap

- Each compressed trace line projection can be read from the TCP link and processed independently by the client GUI
- Pipelined processing of trace line projections by the client GUI overlaps client rendering with communication from server

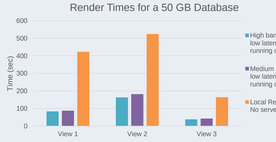



## Managing Massive Traces

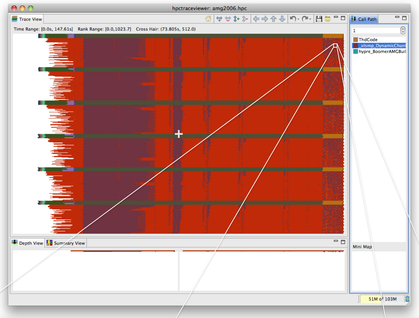
- Representation of trace data on disk is a large file with an index that identifies offsets of individual traces
- For an MPI rank to compute trace projections, it needs the index and the section of the file containing the subset of trace lines it projects
  - Mmaps file disk pages on demand
- To enable parallelism for computing any trace projection, we map sections (pages) of the file to an MPI rank on demand
- Receipt of new windowing commands from the GUI client may shift attention to different regions of the trace
- Server uses LRU mapping of pages and unmaps data as needed

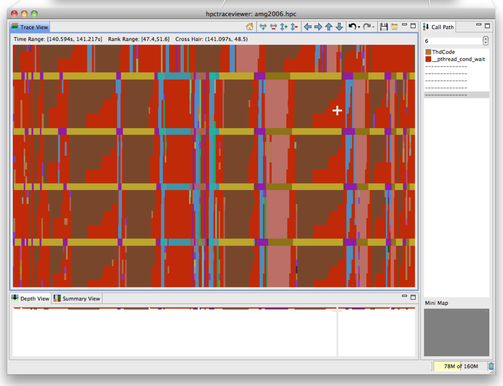



## Results and Next Steps




- The ideal number of server processes per node varies by database and the platform
  - Too many processes per node results in contention for I/O resources while too few results in wasted computation power
- For large databases, using the server significantly reduces render times, increasing interactivity and facilitating the diagnosis of load imbalance issues and other bottlenecks.
- Interactivity is only slightly dependent on the quality of the connection between the server and client
  - Reducing the connection speed by 30% (7 MB/s to 5 MB/s) only increased render times by 5-10%
- More performance tuning, especially tuning of the I/O access patterns to take better advantage of parallel file systems, is still needed




A 4.97 GB database of the execution of the AMG2006 benchmark running with 1024 processes, each with 8 OpenMP threads, on Argonne's Mira. Rendering the home view for this database with hpctraceviewer running on a laptop at Rice University and the server running on 8 cores of Tukey at Argonne took 8.63 seconds. Rendering the inset took 1.03 seconds. Rendering these same views without the server took 75.71 seconds and 1.96 seconds, respectively.

The red stair-step patterns indicate load imbalance. Our new client-server approach will enable us to analyze long executions for much larger numbers of processors.