Caliper: Composite Performance Data Collection in HPC Codes

David Böhme
Lawrence Livermore National Laboratory
Livermore, CA, USA
boehme3@llnl.gov

Todd Gamblin
Lawrence Livermore National Laboratory
Livermore, CA, USA
tgamblin@llnl.gov

Peer-Timo Bremer
Lawrence Livermore National Laboratory
Livermore, CA, USA
ptbremer@llnl.gov

Olga Pearce
Lawrence Livermore National Laboratory
Livermore, CA, USA
olga@llnl.gov

Martin Schulz
Lawrence Livermore National Laboratory
Livermore, CA, USA
schulzm@llnl.gov

ABSTRACT

Correlating performance metrics with program context information is key to understanding HPC application behavior. Given the composite architecture of modern HPC applications, metrics and context information must be correlated from independent places across the software stack. Current data-collection approaches either focus on singular performance aspects, limiting the ability to draw correlations, or are not flexible enough to capture custom, application-specific performance factors. With the Caliper framework, we introduce (1) a flexible data model that can efficiently represent arbitrary performance-related data, and (2) a library that transparently combines performance metrics and program context information provided by source-code annotations and automatic measurement modules. Measurement modules and source-code annotations in different program and system components are independent of each other and can be combined in an arbitrary fashion. This composite approach allows us to easily create powerful measurement solutions that facilitate the correlation of performance data across the software stack.

1. INTRODUCTION

Understanding the performance of large-scale simulations generally relies on two types of data: *measurements* that reflect the performance of the hardware or system software, e.g., FLOP counts, cache misses, or network stalls; and the *context* in which these measurements were taken, e.g., on what processor, in which library or at what call-path.

Traditionally, much of the research focus has been on the measurement component of performance analysis. As a result, the HPC community has developed a number of comprehensive profiling and trace-recording tools [4, 1, 5, 2, 3] to collect a wide range of measurements. These tools are able to manage the immense volume of data coming from large-scale parallel simulations using a variety of programming models such as MPI, OpenMP or CUDA; and can collect data across thread or address-space boundaries. However, they are often geared towards specific types of analyses, e.g., call-stack profiling or message tracing. Consequently, these tools typically only maintain limited and specialized context information. This lack of context information increasingly limits our ability to understand and optimize the performance of large-scale applications. Thus, many users would like to expose custom application semantics in the form context information through a tool-agnostic interface. Examples for such contexts are phases of a coupled solver or levels of an adaptive grid. Moreover, unlike the more monolithic code bases of the past, modern applications consist of a complex network of different physics packages, numerical solvers and support libraries. To understand the performance behavior, it is therefore necessary to relate the resulting information between them.

The Caliper framework provides an extendable, process-wide shared context, capable of combining information across the entire software stack and application structure. To enable this, Caliper provides a simple annotation API using *attribute:value* semantics that lets each software module independently define context attributes and contribute information to a process-wide context store. Additionally, Caliper provides a set of measurement services that automatically tag each performance measurement with the current context information. External tools can use Caliper's API to extract the current context and annotate their own measurements. This produces a rich set of performance data that supports a wide variety of data analysis and visualization techniques.

2. COMPOSITE DATA COLLECTION

To collect performance data, Caliper provides

- An annotation interface for applications to provide arbitrary domain-specific context information, such as iteration number, computational phase, or element locations within a physical domain;

- An interface for plug-in tools to provide measurement data, which allows tool developers to easily create small measurement plug-ins that integrate with context annotations and other plug-ins; and

- The ability to compose context annotations and measurement providers across the software stack.

Composition of different context annotations and measurement providers is Caliper's key capability. Two key features enable this composability. First, Caliper is based on a generic *attribute:value* data model. This allows us to store any kind of data; it is not limited to pre-defined context categories. In contrast, classic performance analysis tools only support few, non-extensible types of automatically derived or user-provided context information, such as the source-code call path or application phases. Second, Caliper provides a per-process in-memory data store for context information. Annotated modules can update this in-memory store independently from any location in the software stack. Thus, developers can independently annotate application modules, libraries, or runtime system components. Caliper automatically combines the information provided by the individual annotations and attached measurement plug-in services at runtime. This capability enables the correlation of measurement data and context information across the software stack.

2.1 Data Model

Conceptually, Caliper stores measurement records in the form of *attribute:value* lists. Attributes have a unique name and other metadata, such as the datatype and scope (process, thread, or task) of the associated values. Caliper automatically keeps per-thread or per-task entries thread or task-local, respectively.

Internally, Caliper creates an efficient tree-based representation of context data, so that a combined context built from multiple, independent attributes can be referenced through a single tree node pointer.

2.2 Workflow

Caliper is designed to be a "glue layer" between context annotations, measurement services, and measurement control/data processing services.

Annotated source-code modules and measurement services provide data by updating the in-memory data store through annotation and service APIs. An independent measurement trigger module creates *context snapshots* upon certain events, e.g. any or specific attributes being updated, or a timer running out. The context snapshot contains the current state of the in-memory buffer together with an up-to-date set of performance measurements from the attached measurement services. Context snapshots are pushed to Caliper's output stack. Alternatively, external tools can query context snapshots.

2.3 Data Processing

Caliper creates a single context stream per instance (i.e., per process), which contain metadata (attribute descriptions and Caliper's internal context tree) as well as context snapshot records. Currently, the per-process context streams are written to disk, where it can be analyzed off-line. We plan to push Caliper output streams into a scalable tree-based reduction network to perform merge and aggregation operations on-line.

Because of its flexible data format, it is straightforward to import Caliper data into external data analytics and visualization tools.

3. STATUS & RESULTS

We used Caliper to instrument several components in an LLNL radiation hydrodynamics code. Although each component is instrumented individually, Caliper allows us to look at how the components impact each other. For example, the ability to annotate the domain sizes produced by the AMR library SAMRAI along with the iteration count in the HYPRE solver facilitates application developers in studying the effects of the domain sizes on the solver convergence. Because our model handles shared context, existing instrumentation does not interfere with new instrumentation, making it easy to add new annotations as application developers see fit. Instrumentation reusability and composability is key for large software. This study shows how Caliper's process-wide data store offers a new path towards effective and insightful performance analysis for complex HPC applications.

Acknowledgement

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DEAC52-07NA27344 and supported by the Office of Science, Office of Advanced Scientific Computing Research as well as the Advanced Simulation and Computing (ASC) program.

LLNL release LLNL-POST-676083.

4. REFERENCES

[1] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Hpctoolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[2] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, Apr. 2010.

[3] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

[4] M. Schulz, J. Galarowicz, D. Maghrak, W. Hachfeld, D. Montoya, and S. Cranford. Open|speedshop: An open source infrastructure for parallel performance analysis. *Scientific Programming*, 16(2-3):105–121, 2008.

[5] S. Shende and A. D. Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.