Mitos: A Simple Interface for Complex Hardware Sampling and Attribution

Alfredo Giménez
Lawrence Livermore National Laboratories
gimenez1@llnl.gov
University of California, Davis
agimenez@ucdavis.edu

Benafsh Husain
Lawrence Livermore National Laboratories
husain1@llnl.gov

David Böhme
Lawrence Livermore National Laboratories
boehme3@llnl.gov

Todd Gamblin
Lawrence Livermore National Laboratories
tgamblin@llnl.gov

Martin Schulz
Lawrence Livermore National Laboratories
schulzm@llnl.gov

ABSTRACT

As high-performance architectures become more intricate and complex, so too do the capabilities to monitor their performance. In particular, hardware sampling mechanisms have extended beyond the traditional scope of code profiling to include performance data relevant to the data address space, hardware topology, and load latency. However, these new mechanisms do not adhere to a common specification and as such, require architecture-specific knowledge and setup to properly function. In addition, the incoming data is often low-level and difficult to attribute to any useful context without a high level of expertise. This has resulted in a destructively steep barrier to entry, both in data acquisition and analysis.

Mitos provides a simple interface for modern hardware sampling mechanisms and the ability to define attributions of low-level samples to intuitive contexts. We present the Mitos API and demonstrate how to use it to create detailed yet understandable visualizations of performance data for analysis.

1. INTRODUCTION

In an effort to facilitate performance optimization on complex codes for highly parallel architectures with deep memory hierarchies, vendors have created a variety of advanced performance-monitoring units (PMUs) that allow users to sample different types of hardware events. Traditionally, sampling is used to create a code profile, which breaks down the execution time of a program in terms of the individual lines of code. As such, this constitutes a *code-centric* performance measurement approach.

While understanding performance in terms of the code is useful, there are a multitude of factors that affect performance, such as data structures and layout, hardware locality, and parallel domain decomposition. To address this, architecture vendors now supply additional PMUs capable of sampling information relevant to the data and hardware topology of a program execution, thus enabling *data-centric* and *hardware-centric* performance measurements. However, these PMUs were not developed in accordance to any specification, and as such are highly architecture-specific. As a result, their setup and configuration are non-trivial. In addition, the raw output is provided at a low level and difficult to intepret. Consequently, though these advanced monitoring capabilities promise a great deal of new insight into performance analysis, these drawbacks currently prevent them from being a viable option to a majority of users.

We present Mitos, a simple interface for configuration of advanced sampling mechanisms as well as attribution of samples to different contexts for visualization and analysis. Mitos also provides an API that allows for flexible configuration and attribution, including adaptive sampling and application-specific attribution.

2. HARDWARE SAMPLING INTERFACES

PMUs for advanced sampling have existed in the hardware for approximately four processor generations but lacked operating system support until recent years. Examples include Intel's Precise Event-Based Sampling (PEBS) [4], AMD's Instruction-Based Sampling (IBS) [1], and IBM's Marked Events [3]. Though their implementations and available data vary, they each provide three entries of particular interest in a recorded sample:

1. Instruction latency
2. Virtual address of a data load and/or store
3. Data source information (L1, L2, TLB)

These PMUs have been exposed primarily through the `perf_event` API in Linux, which provides a powerful abstracted API

for performance data acquisition across different architectures but suffers from poor documentation and difficulty of use [8]. Proprietary solutions, such as Intel's PIN tool and VTune, offer advanced analysis using these PMUs but are vendor-specific and do not make acquired data available for different types of visualization and analysis [5] [7]. PAPI [6] aims to support these PMUs in future releases but currently its design does not easily incorporate their integration [9].

3. MITOS

Mitos leverages the rich capabilities of the `perf_event` API but provides a simple and flexible abstraction layer for the user. It is designed such that the user may set up a predefined sampling setup in a few lines of code or define functions for more complex attribution. At the most basic level, Mitos can configure sampling of memory loads with a specified frequency and, if available, latency threshold. Under the hood, Mitos creates an event attribute with the necessary raw event encodings for load sampling, sets up a memory-mapped buffer into which the processor will dump samples, and sets up a per-thread signal handler that reads from the buffer and parses out individual samples. The user must specify a function that takes a recorded sample as input and pass it to Mitos as the "sample handler" function, as shown below:

```
void
sample_handler(perf_event_sample *sample, void *args)
{
    // Do something with the sample
}
```

This design allows the user to perform any kind of custom attribution or recording, though as a starting point they may wish to record the sample in a predefined way using `Mitos_write_sample(mitos_output *out)` which records all fields of the sample in a comma-separated values file.

4. DATA ATTRIBUTION

Mitos also provides predefined mechanisms for attributing sampled data addresses to data objects in the code. After allocation, the user may annotate a buffer by specifying a name, address, element size, and multidimensional buffer size, as shown below:

```
double *a = new double[N*N];

size_t dims[2] = {N,N}; // NxN matrix
Mitos_add_symbol("a",*a,sizeof(double),dims,2);
```

When Mitos receives a sample with a data address that falls within the bounds of this buffer, it will additionally record the name of the buffer and the (possibly multidimensional) index used to access it (e.g. the values x and y in `a[x][y]`).

This attribution allows tools to leverage the context of the data layout in visualization and analysis; for example, the user may visualize where the most loads occurred within a multidimensional array. This may provide insight into when certain elements of a buffer fall out of cache and point towards possible tiling techniques for cache optimization. The visualization tool MemAxes plots the access indices in histograms in order to identify hotspots within data buffers and relate them back to access patterns in the code [2].

5. HARDWARE ATTRIBUTION

Load samples acquired with Mitos provide an associated data source, socket, and CPU, making hardware-centric analysis also possible. MemAxes provides a visualization of utilization in the hardware topology that maps samples acquired through Mitos to individual resources. Here, it is possible to discover patterns in parallel domain decomposition and data sharing between caches for the purpose of optimizing code for improved locality. We demonstrate this hardware-centric visualization using mapped samples from Mitos.

6. APPLICATION-SPECIFIC ATTRIBUTION

Though the previous areas of attribution are generalizeable to all codes, it is often of interest to provide an attribution for a specific domain. This is possible through defining a custom sample handler that maps a data address or calculated access index to an application domain. We demonstrate this technique for mapping accesses to cells within running iterations of an adaptive mesh refinement simulation. This enables analysis of data-dependent performance issues such as boundary conditions.

Because the handler is user-defined, it is easily integrated into other frameworks for context-aware performance analysis. We couple Mitos with the context annotation tool Caliper to attribute load samples with phases in a program's execution.

7. CONCLUSIONS

Mitos provides an accessible interface for setting up complex performance-monitoring units capable of sampling data addresses with associated hardware resources and instruction latencies. It additionally performs a set of automatic attributions for the data objects, access indices, and hardware topology associated with a sample, and makes it possible for the user to define custom attribution functions for specific domains. As a result, Mitos enables a multitude of new analysis methods beyond the traditional code-centric paradigm and in turn provides novel insight into the performance of complex codes.

8. ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and supported by Office of Science, Office of Advanced Scientific Computing Research as well as the Advanced Simulation and Computing (ASC) program.

9. REFERENCES

[1] P. J. Drongowski, L. Yu, F. Swehosky, S. Suthikulpanit, and R. Richter. Incorporating instruction-based sampling into amd codeanalyst. In *ISPASS*, pages 119–120. IEEE Computer Society, 2010.

[2] A. Giménez, T. Gamblin, B. Rountree, A. Bhatele, I. Jusufi, P.-T. Bremer, and B. Hamann. Dissecting

on-node memory access performance: a semantic approach. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 166–176. IEEE, 2014.

[3] B. Hall, R. Arnold, P. Bergner, W. dos Santos Moschetta, R. Enenkel, P. Haugen, M. Meissner, A. Mericas, P. Oehler, B. Schiefer, et al. *Performance Optimization and Tuning Techniques for IBM Processors, including IBM POWER8*. IBM redbooks. IBM Redbooks, 2014.

[4] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3B*, August 2007.

[5] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, June 2005.

[6] P. J. Mucci, S. Browne, C. Deane, and G. Ho. Papi: A portable interface to hardware performance counters. In *In Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999.

[7] J. Reinders. *VTune Performance Analyzer Essentials: Measurement and Tuning Techniques for Software Developers*. Engineer to Engineer Series. Intel Press, 2005.

[8] V. Weaver. perf_event programming guide. Accessed: 2015-08-7.

[9] V. Weaver. Proposed features for the papi 6.0 release. 2014.