Supporting High Level Types and Reduction Functions for Performance Data Collection

Udayanga Wickramasinghe, David Böhme, Todd Gamblin and Martin Schulz
Lawrence Livermore National Laboratory

Modern day performance data collection and tracing tools mostly deal with ‘Data’ in fine grained form in order to cope with non intrusive, low memory footprint and high bandwidth requirements of host applications. However there is great disparity between data description in high level application code and performance data. Effective use of data types and reduction functions can bridge this gap while allowing to maintain same level of application runtime guarantees. The Caliper framework tries to achieve this by representing complex data objects in type hierarchies and introducing an automatic online aggregation module to apply reductions to measurements on different data types to enable higher order data representation.

Caliper Online Aggregation Module

The Caliper aggregation module is responsible for collecting data from various measurement sources and providing in-situ transformation of this data using predefined and user defined operators. Online aggregation module is exposed as a Caliper service.

Features

- **API to describe Complex Type Annotations**: Easy to use API to describe and set hierarchical and other complex data in Caliper.
- **Data Serialization**: Enable performance data serialization using a self describing conduit JSON schema and compact binary formats.
- **Aggregation Service**: On-the-fly efficient aggregation scheme on measurement data per attribute and/or per data item. Has the ability to describe aggregation groups over filters and projections.
- **Predefined and User-Defined Operators**: Avg, Min, Max, Histogram, Join, Product, etc and custom functions to apply on simple/complex types.

Applicators – Mitos/Hwloc

Mitos is a library that samples hardware events via the Linux perf_event API. Mitos coupled with hwloc can provide hardware topology information crucial for performance data collection tools like Caliper.

```cpp
#include "caliper.h"

void use_Counter(Var::Variant input)
{
    // use a counter variable
    // input is of type Var::Variant"name": ["counter"]
    // output is of type Var::Variant"name": ["counter"]
    // output["name"] = input["name"].copy()
    
    // access the current measured value
    int val = input["name"][0].counter.val()
    
    // create an annotation for "name" with the current value
    
    // register_annotation("name", ["value": val])
    
    // output annotation
    
    // write out the annotation to the user
    
    // output["name"] = input["name"];"name": ["counter"]
    
    // output annotation
    
    // register_annotation("name", ["value": val])
    
    // output annotation
    
    // write out the annotation to the user
}
```

Visualization of the aggregated topology information in terms of NUMA domains, sockets and cpu cores.

```cpp
Caliper dev/compotypes branch is accessible via LC stash
https://lc.llnl.gov/stash/projects/PIPER/repos/caliper/browse
Contact: boehme3@llnl.gov wickramasing1@llnl.gov
```