Advanced Event-Sampling Support for PAPI

Forrest Smith, Vince Weaver and **Pascal Francis-Mezger**

{forrest.smith,vincent.weaver,pascal.francismezger}@maine.edu

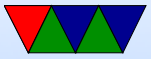
ESPT Workshop 2018 — 16 November 2018

PAPI Background

- PAPI, the Performance API – widely used cross-platform performance library
- Extreme Scale? The bigger the machine, the harder to find where performance is going.
- PAPI provides sampling support, but lacks many features provided by modern hardware

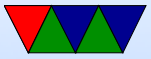
Plain Hardware Performance Counter Measurement

- Counters built into CPU that measure useful performance info:
  - Cycles, Instructions
  - Cache hits/misses
  - Branch predictor
  - etc.
- Typically used for total (aggregate) counts



Sampled Measurements

- Often want more details about program behavior
- Sampling: periodically log performance info while program running
- Can be used to statistically extrapolate detailed program behavior
- Tradeoff of how often interrupt (slow down) vs log size (which can become sizable)



Simple Sampling

- Set a performance counter to overflow, say after 100,000 cycles
- Most hardware allows this to trigger a hardware interrupt
- Have the operating system log key info, such as other counter values, instruction pointer location, register contents
- This can also be done with a timer interrupt if hardware counter overflow interrupts are not available

The traditional PAPI Interface

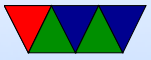
Setting up overflow event:

```
int PAPI_overflow(int EventSet, int EventCode,
                  int threshold, int flags,
                  PAPI_overflow_handler_t handler);
```

Overflow signal handler:

```
typedef void (*PAPI_overflow_handler_t)
                  (int EventSet, void *address,
                  long long overflow_vector, void *context);
```

Mostly just provides the instruction pointer, up to user to gather these values for analysis.



Advanced Sampling Features on Modern CPUs

- Intel Precise Event-Based Sampling (PEBS)
- AMD Instruction Based Sampling (IBS)
- IBM s390 CPU Measurement Facility
- ARM has some support too

Low Latency Sampling

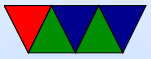
- instead of doing a costly interrupt and entering the operating system, hardware given a dedicated memory buffer it can fill without software intervention
- When the buffer is nearly full, then it can notify the OS to take care of it
- Intel PEBS supports this

Hardware Profiling

- At regular intervals a random instruction is chosen and detailed info about it is logged
- Intel PEBS and AMD IBS support this

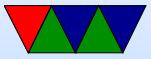
Extended Profile Info

- Extra CPU state
  - Register state
  - Branch predictor outcomes
  - TSC timestamp value
  - Transactional memory info
  - Load and Store info (address, latency, data source)
  - TLB info
- Intel PEBS supports this
- AMD Instruction Based Sampling (IBS) is similar in concept, but only one instruction at a time is recorded (no sampling buffer)



Low-Skid Interrupts

- Modern out-of-order processors take time to stop to handle interrupt
- This can lead to results being offset a few instructions from the instruction that triggered overflow
- Low-skid support will return exact cause of overflow (often at expense of a minor slowdown)
- Intel PEBS supports this
- AMD supports this



Last Branch Sampling

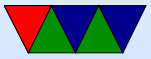
- Intel Last Branch Record (LBR)
  - Record info on last 4 to 32 branches
  - Last location branched to/from
  - Whether it was predicted properly
- Intel Branch Trace Store (BTS)
  - Last N branches written to sample buffer
  - Later machines add additional info

Processor Trace

- Intel Processor Trace (PT)
  - Record program execution traces
  - Generate Basic Block Vectors
  - Trace Power Events
- ARM CoreSight processor tracing
  - Similar idea, same interface
- These traces can be very large

Other Similar Info

- Intel Resource Director Technology (RDT)
- Intel Cache Monitoring Technology
- Intel Memory Bandwidth Monitoring
- Intel Cache Allocation Technology / Code Data Prioritization
- Intel Cache Quality of Service Monitoring
- Linux does not support any of these

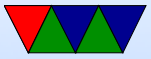


The `perf_event` Interface

- Linux uses the `perf_event` interface
- It has support for most of the advanced features
- Open a mmap page to hold the samples, configure the sample type with the complex `perf_event_attr` struct

The `perf_event` Interface Overview

- Sampled Profiling – you indicate what types of events are desired out of a huge list
- Low latency sampling (PEBS) – simply open a big enough mmap region and set a high enough watermark/threshold
- AMD IBS results are returned differently than PEBS, as "RAW" samples that have to be decoded
- Low-skid interrupts – specify an event with the "precise" value set to the desired



Proposed New PAPI Interface #1

Abstract all settings to one "sample_type" field.

When event threshold is hit the handler is called with a pointer to the `mmap()` sample buffer.

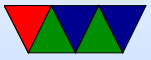
A library of sample code provided to help interpreting.

```
int   PAPI_sample_init(int EventSet, int EventCode,
        long long sample_period, long long sample_type,
        PAPI_overflow_handler_t handler);

typedef void (*PAPI_overflow_handler_t)
        (int EventSet, void *address,
        long long overflow_vector, void *context);
```


Problems with Interface #1

- sample_type allows setting up Intel PEBS type samples, including cache extended info
- does not necessarily allow setting arbitrary other advanced features, or setting buffer size
- If new features are added to perf_event interface, might not fit with this level of abstraction



Proposed New PAPI Interface #2

Just give up and pass in a `perf_event_attr`.

User still has to deal with reading out the sample buffer.

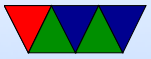
```
int   PAPI_sample_init(int EventSet,
        int EventCode,
        long long sample_period,
        perf_event_attr *attr,
        PAPI_overflow_handler_t handler);
```


Problems with Interface #2

- Allows setting up any supported perf_event interface (except Processor Trace which requires an additional auxiliary mmap buffer to be created)
- Destroys PAPI's ideal of cross-platform interface
- Requires PAPI user to have deep knowledge of perf_event interface

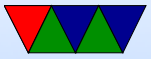
Current Status

- Interface #1 plus example code is in `weaver-sampling` branch of main PAPI git tree
- Raw PAPI data logged, has to be either parsed with (provided) complicated parsing, or written to disk
- Ideally disk format would be the "standard" perf.data file format, but that is poorly documented and not currently working.



Future Work

- Seeking feedback if this is right solution, or should try Interface #2
- Unclear if will be reading in time for proposed PAPI 6.0 major release in late 2018.



Questions?