CS 610: Tracking Performance Monitoring Counters with PAPI

PMC and PAPI

Swarnendu Biswas

Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur

Sem 2024-25-I

Performance Monitoring Counters

- Modern CPUs have hardware counters for tracking many events
  - For example, cycles, instructions, floating-point instructions, loads and stores, i-cache misses, L1 data cache misses, L2 data cache misses, TLB misses, and pipeline stalls
- **Performance counters** are hardware registers attached to processors that measure various events occurring in the processor
- Useful in **analyzing** performance **bottlenecks**
  - Each counter can be programmed with an event type to be monitored (e.g., L1 cache miss or a branch misprediction)

More on Hardware Counters

Why do you need PMCs when we can have software profilers?

- Hardware counters incur lower overhead and require minimal source code modifications
- The information tracked in each run is limited

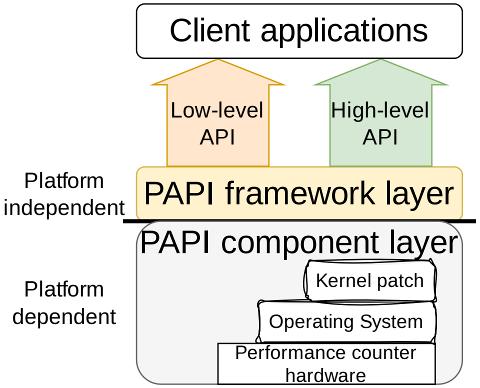
Challenges with using hardware counters

- Manipulating hardware counters directly is complex
  - Different processors have different counters; the code is not portable
  - Many processors have more events that can be tracked than hardware counters, so only a subset of events can be measured in a given run

Performance Application Programming Interface (PAPI)

PAPI

- The PAPI library provides an interface for gathering performance counter information from diverse platforms
  - Supports major CPUs (e.g., Intel, AMD, ARM PowerPC), GPUs, and accelerators
  - Many third-party performance analysis tools use PAPI (e.g., PerfSuite, HPCToolkit, and TAU)
- Provides two interfaces to manipulate hardware counters



https://github.com/icl-utk-edu/papi

Setting up PAPI v7.1

```
# Uninstall older PAPI versions
# sudo apt remove libpapi-dev papi-tools
wget https://github.com/icl-utk-edu/papi/archive/refs/tags/papi-7-1-0-t.tar.gz
tar xf papi-7-1-0-t.tar.gz
cd papi-papi-7-1-0-t/src
# Include optional components like GPU support
./configure
make -j"$(nproc)"
sudo make install
papi_version # PAPI Version: 7.1.0.0
```

Preset Events

- Preset events are platform-independent names for events deemed useful for performance tuning
- Standard set of 108 events for application performance tuning
  - For example, accesses to the memory hierarchy, cache coherence protocol events, cycle and instruction counts, functional unit and pipeline utilization
- Run papi_avail utility to determine preset events available on the platform

PAPI_L1_DCH	Level 1 data cache hits
PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_DCR	Level 1 data cache reads
PAPI_L1_DCW	Level 1 data cache writes
PAPI_L1_DCA	Level 1 data cache accesses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L1_TCH	Level 1 total cache hits
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L3_DCM	Level 3 data cache misses
PAPI_TOT_INS	Total instructions executed
PAPI_TOT_CYC	Total cycles
PAPI_IPS	Instructions executed per second

Native Events

- PAPI also provides access to platform-dependent native events through a low-level interface
    - Any event countable by the CPU (e.g., L3_CACHE_MISS)
- Use papi_native_avail utility to see all available native events
- Preset events can be derived from single or linear combinations of native events
    - PAPI_L1_TCM may be L1 data misses + L1 instruction misses

Useful APIs

PAPI_query_event()

Check whether the CPU can measure the relevant PAPI event

```
if (PAPI_query_event(PAPI_TOT_INS) != PAPI_OK) {
  std::cerr << "PAPI total instruction error!\n";
}
```

PAPI_event_code_to_name()

Get the name of a preset or native event from the event code

```
int EventCode = 0 | PAPI_NATIVE_MASK;
retval = PAPI_event_code_to_name(EventCode, EventCodeStr);
/* Print the native event for this platform */
if (retval == PAPI_OK)
  printf("Name: %s\nCode: %x\n", EventCodeStr, EventCode);
```

High-Level API

- Track performance events in named code regions
  - Meant for application programmers wanting simple but accurate measurements
  - Calls the lower-level API
  - Interface redesigned from v6+
- Events to be recorded are provided via comma-separated environment variable PAPI_EVENTS
  - Only allows preset events
- Values of performance events are the difference between the end region and the begin region calls
  - Only the end values are stored for instantaneous events that track temperature or power
- Print stats to stdout by setting PAPI_REPORT to 1

HL Example

Compile with `g++ -O3 -std=c++17 hl-ex1.cpp -lpapi`

https://github.com/icl-utk-edu/papi/wiki/PAPI-HL

Low-Level API

- Use when you want finer-grained measurements, can track both preset and native events
- Can use both high-level and low-level APIs
  - PAPI library should be initialized before the first low-level PAPI call

LL Example

Compile with `g++ -O3 -std=c++17 ll-ex1.cpp -lpapi`

https://github.com/icl-utk-edu/papi/wiki/PAPI-LL

PAPI with CUDA

- PAPI is also available for CUDA GPUs
  - Uses the CUPTI interface
- Gives useful information about the GPU usage
  - IPC, memory load/stores/throughput, branch divergences, and SM(X) occupancy

References

- A. Kozhokanova et al. PAPI: Performance API Introduction & Overview, Virtual Institute — High Productivity Supercomputing 2021.
- A. Pereira. PAPI - Performance API.
- A. Avila. PAPI: Performance API, Virtual Institute — High Productivity Supercomputing 2011.
- Anthony Castaldo. Introduction Methodology GPU Tuning.