An OS-Oriented Performance Monitoring Tool for Multicore Systems

J.C. Saez, J. Casas, A. Serrano, R. Rodriguez-Rodriguez, F. Castro, D. Chaver, M. Prieto-Matias

ArTeCS Group, Complutense University of Madrid, Spain.
Email: {jcsaezal,jorcasas,abeserra,rrodriguezr,fcastror,dani02,mpmatias}@ucm.es

Contribution

Hardware performance monitoring counters (PMCs) have proven effective in characterizing application performance. A large body of work has demonstrated that several OS components, such as the scheduler, can perform effective runtime optimizations in multicore systems by leveraging performance-counter data. While existing tools greatly simplify the collection of PMC data from userspace, they do not provide an architecture-agnostic mechanism that is capable of exposing high-level PMC metrics to the OS scheduler. Thus, the implementation of PMC-based scheduling schemes is typically tied to specific processor models.

To address this shortcoming we created PMCTrack, a novel tool for the Linux kernel that seamlessly enables the system software to access PMC data and other insightful metrics available in modern processors and which are not directly exposed as PMCs, such as cache occupancy or energy consumption. Despite being an OS-oriented tool, PMCTrack still allows the gathering of monitoring data from userspace, enabling developers to perform offline analysis in various ways.

Userspace oriented monitoring

Available monitoring modes

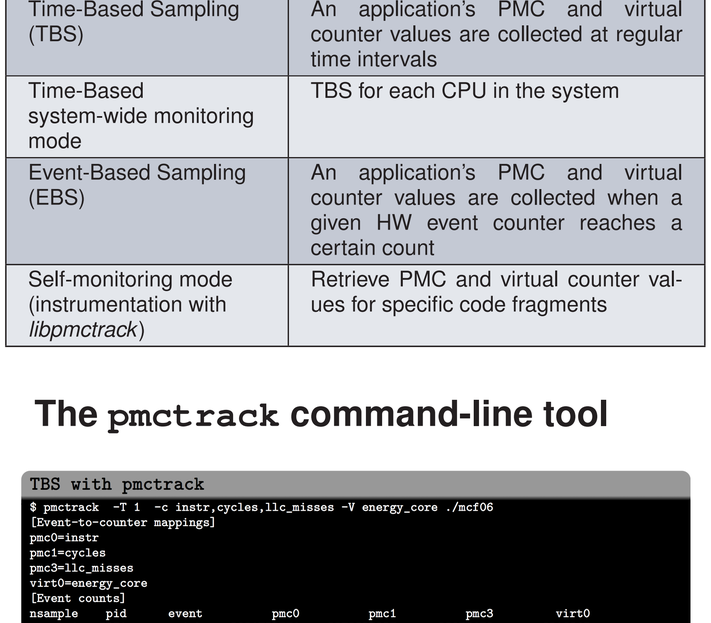
Mode	Description
Time-Based Sampling (TBS)	An application's PMC and virtual counter values are collected at regular time intervals
Time-Based system-wide monitoring mode	TBS for each CPU in the system
Event-Based Sampling (EBS)	An application's PMC and virtual counter values are collected when a given HW event counter reaches a certain count
Self-monitoring mode (instrumentation with *libpmctrack*)	Retrieve PMC and virtual counter values for specific code fragments

The `pmctrack` command-line tool

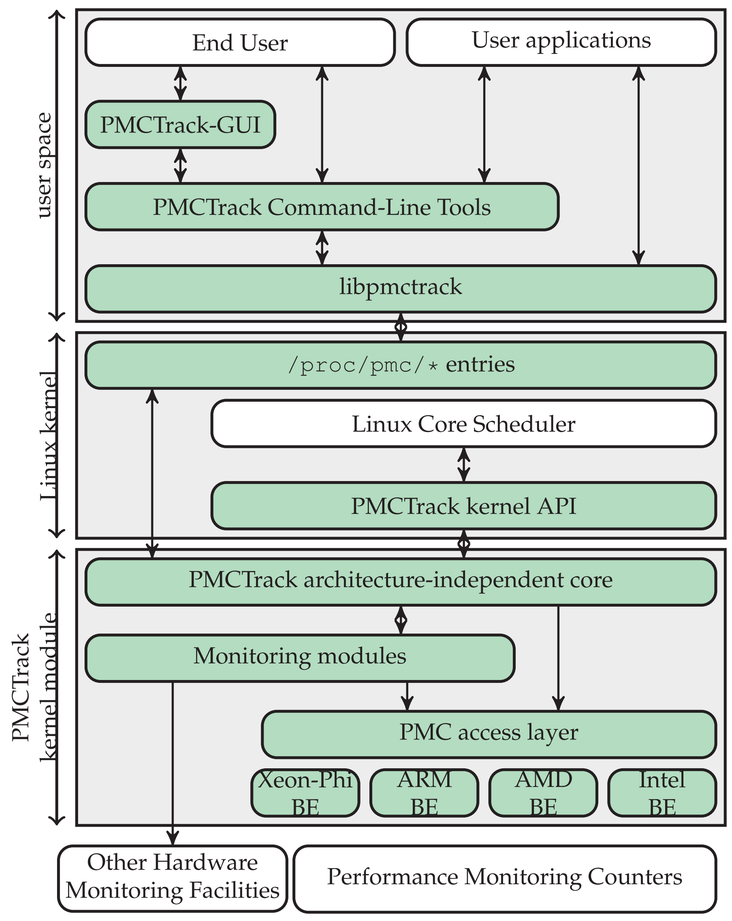
TBS with pmctrack

```
$ pmctrack  -T 1  -c instr,cycles,llc_misses -V energy_core ./mcf06
[Event-to-counter mappings]
pmc0=instr
pmc1=cycles
pmc3=llc_misses
virt0=energy_core
[Event counts]
nsample    pid      event          pmc0          pmc1          pmc3         virt0
      1   5051       tick    2031752774    3278225927      28076472       6816345
      2   5051       tick    1220553549    3581105680      24851799       7674194
      3   5051       tick    1200669666    3579946939      24758056       7533020
      4   5051       tick    1439111649    3377276912      22649829       8372497
      5   5051       tick    1741678429    2910646974       7125557       9042236
      6   5051       tick    2288054908    3591634920      19342428       6588195
      7   5051       tick    2427548635    3593134689      18843632       6766113
      8   5051       tick    1387333303    3592647444      22690759       6272949
      9   5051       tick    1451673704    3593864932      22313698       6244079
     10   5051       tick    1331258605    3593793009      22677829       6211608
     11   5051       tick    1323855919    3593486094      22600530       6065124
     12   5051       tick    1352018668    3592025587      22390828       6119812
     13   5051       tick    1327291415    3552079221      21806709       6572875
     14   5051       tick    1292799158    3584908606      21796200       6203979
...
```

PMCTrack-GUI: a graphical frontend



PMCTrack Architecture



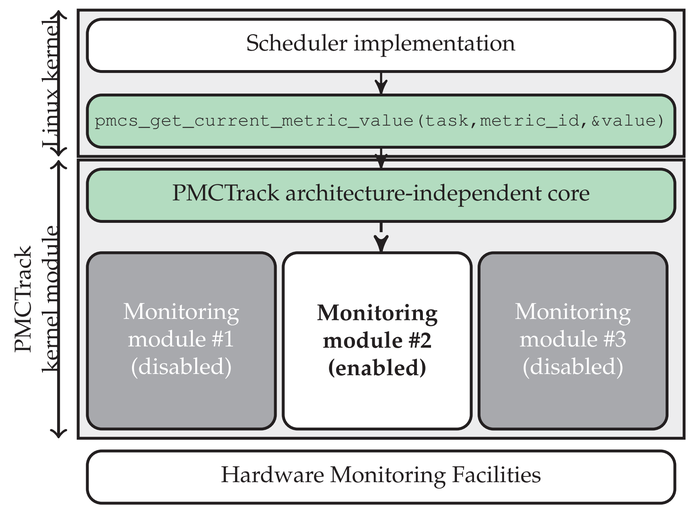
PMCTrack requires a patched Linux kernel to function

- Changes can be easily applied to different kernel versions (2 new source files + ~20 extra lines of code in existing files)

Monitoring modules

A monitoring module (MM) is a *"plug-in"* whose code lives in PMCTrack's loadable kernel module, and ...

- Implements a set of callback functions
- May take full control of PMCs and configure them using an architecture-independent mechanism
- Can expose non-PMC-related information (e.g. energy consumption, cache usage) as *virtual counters*
- Provide the OS scheduler with per-thread performance metrics

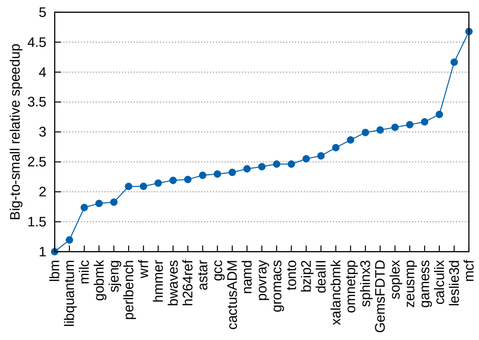


Case study: scheduling on Asymmetric Multicore Processors (AMPs)

An AMP integrates cores with different features (microarchitecture, power consumption, area,...) but with a common instruction set architecture (ISA).

ARM big.LITTLE (e.g., Cortex A57 + Cortex A53)

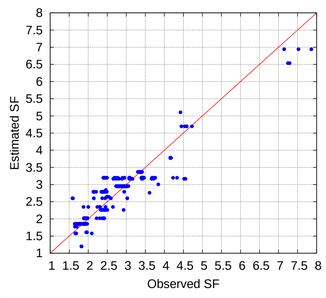
Intel QuickIA prototype: Intel Xeon E5450 (high performance) + Intel Atom N330 (low power)

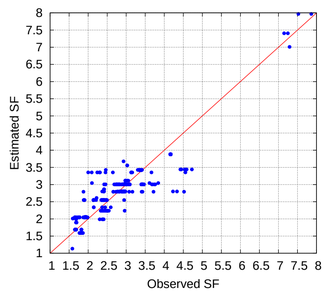


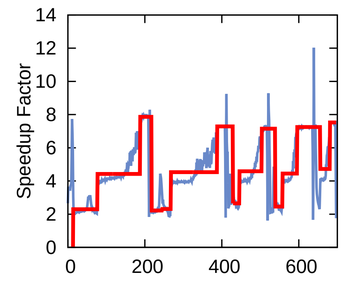
Different big-to-small speedup across applications

To optimize throughput, fairness or energy efficiency the OS scheduler should take thread's big-to-small speedups (SFs) into consideration

Performance-counter based SF estimation with PMCTrack







Scheduler	Publication
CAMP	*A comprehensive scheduler for asymmetric multicore systems*, in Eurosys'10
ACFS	*Towards completely fair scheduling on asymmetric single-ISA multicore processors*, in JPDC'17
EEF-Driven	*On the interplay between throughput, fairness and energy efficiency on asymmetric multicore processors*, in The Computer Journal'17

Case study: cache usage monitoring with PMCTrack

Recent Intel Xeon processors support monitoring LLC usage on a per-application basis (Intel CMT)