An Invitation to Collaborate

Building best-in-class tools for understanding system behavior

jkoshy@ 2023-11-17

What’s this about?

I am looking for people to collaborate with, for building best-in-class tools for understanding ‘whole system’ behavior.

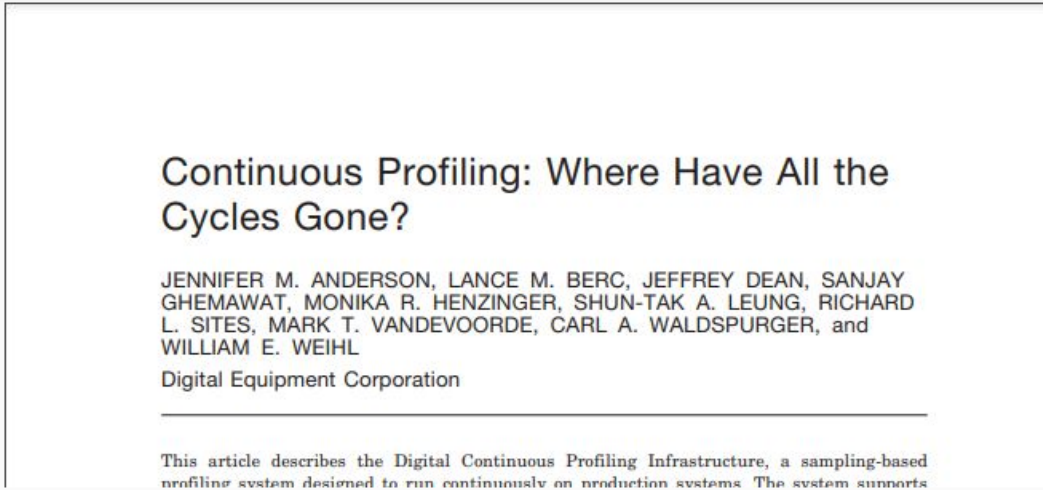
The tools will initially target the BSD family of OSes, but portability is an explicit goal.

The tools are for software developers and for those learning about operating systems, compilation techniques and hardware design. Clear, scaffolded documentation is an explicit goal.

FreeBSD history: hwpmc(4) + tools

Early 2000s:

- Came across the Anderson et al. (1997) paper, and wanted a similar facility on FreeBSD.
- Took off from work to design and implement this.
- Developed in FreeBSD's Perforce repo, moved into FreeBSD CVS in 2005 [link].



Continuous Profiling: Where Have All the Cycles Gone?

JENNIFER M. ANDERSON, LANCE M. BERC, JEFFREY DEAN, SANJAY GHEMAWAT, MONIKA R. HENZINGER, SHUN-TAK A. LEUNG, RICHARD L. SITES, MARK T. VANDEVOORDE, CARL A. WALDSPURGER, and WILLIAM E. WEIHL

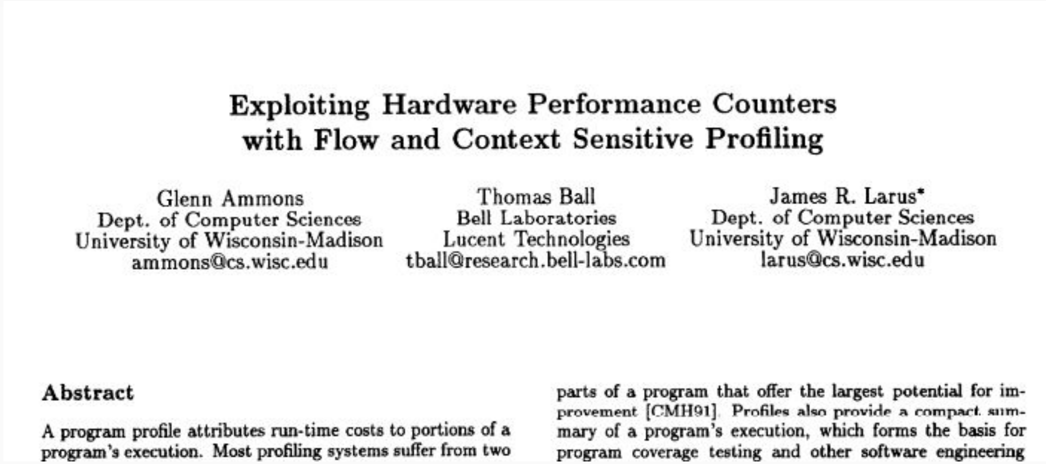
Digital Equipment Corporation

This article describes the Digital Continuous Profiling Infrastructure, a sampling-based profiling system designed to run continuously on production systems. The system supports

FreeBSD history: hwpmc(4) + tools

- Intent: enable self-profiling of apps using cheap `RDPMC` instructions (on x86).
- Added [sampling]() and [callchain capture]() off NMIs later.
- Wrote BSD `libelf` along the way in order to parse ELF.

All very cool, but open-source didn't pay those days, and I needed to get back to earning a living …



Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling

Glenn Ammons
Dept. of Computer Sciences
University of Wisconsin-Madison
ammons@cs.wisc.edu

Thomas Ball
Bell Laboratories
Lucent Technologies
tball@research.bell-labs.com

James R. Larus*
Dept. of Computer Sciences
University of Wisconsin-Madison
larus@cs.wisc.edu

Abstract

A program profile attributes run-time costs to portions of a program's execution. Most profiling systems suffer from two

parts of a program that offer the largest potential for improvement [CMH91]. Profiles also provide a compact summary of a program's execution, which forms the basis for program coverage testing and other software engineering

What wasn't tackled back then

- Portability to other platforms.
  - FreeBSD only (ENOTIME, ENOCLUE).
- Setting up clear processes for stakeholders
  - It was not even clear who the stakeholders were at that time.
- Graded documentation (scaffolding for new users).
  - Only manual pages were written.
- Tools to **visualize** the data being collected; tools to **analyze** data.
- Integration with toolchains (e.g. `cc -pg`, etc).
- Integration with other measurement tools (e.g. **DTrace**).

But it was useful! The community stepped in

Good stuff:

- Support for new CPUs (particularly for x86)!
- New architecture support!
- Some nice features added.
    - Along with some puzzling ones …

But also:

- Complaints now of tools not working / being unmaintained.
- Is self-profiling being used at all?
- Actual utility is yet to be addressed:
    - What does the low-level data collected actually **mean**? How does one interpret it?
    - How do we correlate low-level measurements with other system behavior of interest?
    - How can we **visualize** what the system is doing? What about tools for ‘batch’ analyses?

So, What’s Next?

Community Processes

Portability

New Features

Visualization

Sustainability

Testability

Batch Analysis

Documentation

New Hardware Capabilities

- Needs a change of programming model.
- New **types** of performance monitoring features.
    - E.g. Hardware assisted branch tracing.
- Systems with mixed CPU types!
    - E.g. ARM®.
- Complex constraints around PMC usage.
- Many more architectures now offer PMCs.
    - With their own quirks.
- Performance measurement counters in I/O buses, GPUs, NPUs, peripherals, etc.
    - Need a way to integrate all of these.

Community: ‘RFC’/’PEP’-like Design Process

- **Goal**: coordinate effort, fewer surprises, make it easier to contribute.
- Open to all stakeholders.
- Provide a venue for constructive feedback.
- Provide transparency about upcoming changes.
- **Where?** On a BSD-project agnostic avenue with good tools for collaboration (maybe Github?).

Portability

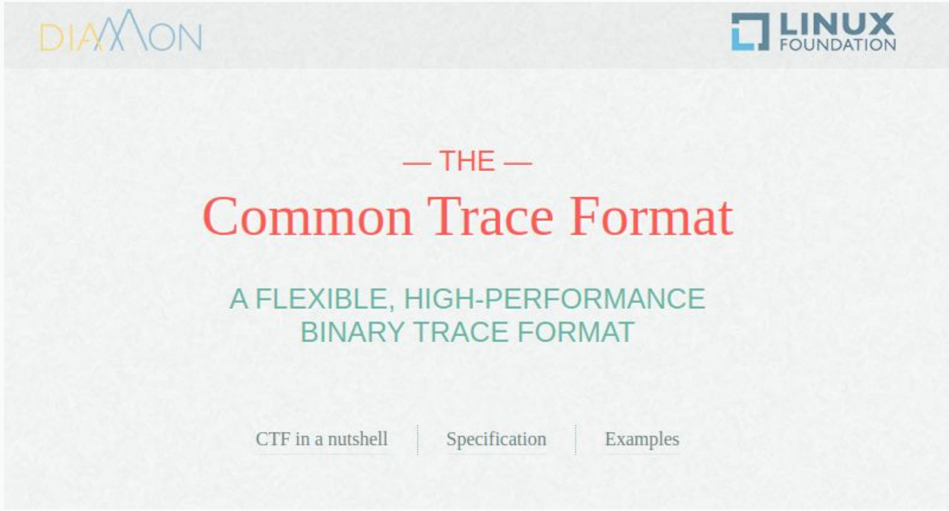
- Portability of code
  - **hwpmc(4)** is very much tied to the FreeBSD kernel.
  - For the next iteration, define interface boundaries between (a) the OS and (b) PMC hardware.
  - Targets: Two *BSDs (FreeBSD & NetBSD)
    - Maybe one microkernel (Minix?, L4?), and maybe one non-*nix open-source OS.
    - Eventually other platforms via the community RFC process.
  - User-space tools intended to be portable to *BSD, GNU/Linux, etc.
- Portability of Skills
  - **Goals**: facilitate sharing of knowledge, processes, tools and techniques for performance analysis across OSes, both production and research.

Testability

- Permit (kernel) modules to be built and tested in user space:
  - Eases development.
  - Eases continuous integration.
  - Eases performance optimization of the kernel modules themselves.
- Inspired by the “Rump kernel” (*Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels*, Antti Kantee, 2012).

Interoperability With Other Tooling

- Align with ongoing standardization efforts.
- Example: the Common Trace Format.
- Could we integrate with Open Telemetry?


— THE —

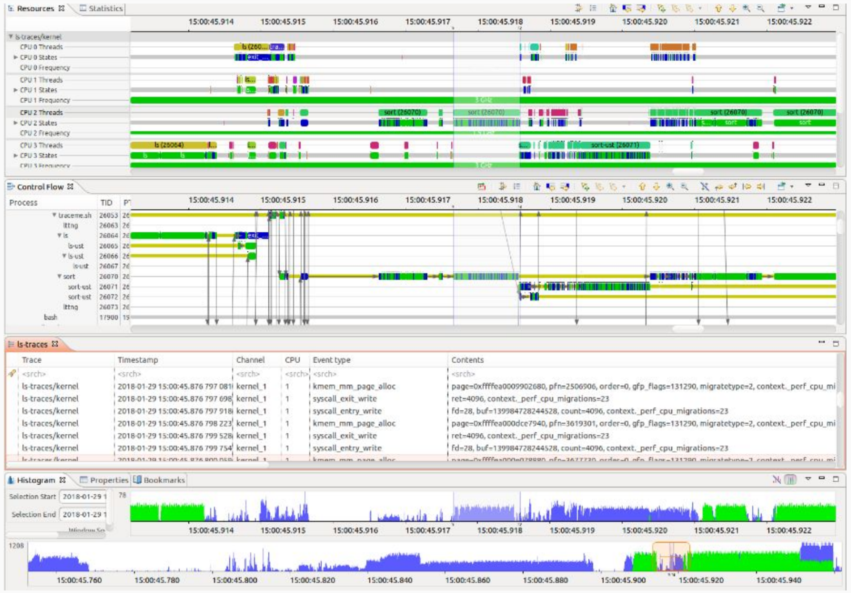
Common Trace Format

A FLEXIBLE, HIGH-PERFORMANCE BINARY TRACE FORMAT

CTF in a nutshell | Specification | Examples

Interoperability: Interactive Visualization

- Integrate with visualization tools (like Trace Compass).
  - Or write our own?
- Aim: visualize hardware, kernel & application behavior **all together**.
  - See hardware behavior, kernel scheduling, CPU voltages & frequencies, database transactions —— at a glance.
  - Trace data ‘flows’ through the system.
  - Share & collaborate on traces (like Perfetto).
- Multi-system tracing.
  - For the distributed/IOT world.
  - **hwpmc(4)** already has partial support.



Trace	Timestamp	Channel	CPU	Event type	Contents
<srch>	<srch>	<srch>	<srch>	<srch>	<srch>
ls-traces/kernel	2018-01-29 15:00:45.876 797 081	kernel_1	1	kmem_mm_page_alloc	page=0xffffea0009902680, pfn=2506906, order=0, gfp_flags=131290, migratetype=2, context._perf_cpu_mi
ls-traces/kernel	2018-01-29 15:00:45.876 797 698	kernel_1	1	syscall_exit_write	ret=4096, context._perf_cpu_migrations=23
ls-traces/kernel	2018-01-29 15:00:45.876 797 918	kernel_1	1	syscall_entry_write	fd=28, buf=139984728244528, count=4096, context._perf_cpu_migrations=23
ls-traces/kernel	2018-01-29 15:00:45.876 798 223	kernel_1	1	kmem_mm_page_alloc	page=0xffffea000dce7940, pfn=3619301, order=0, gfp_flags=131290, migratetype=2, context._perf_cpu_mi
ls-traces/kernel	2018-01-29 15:00:45.876 799 528	kernel_1	1	syscall_exit_write	ret=4096, context._perf_cpu_migrations=23
ls-traces/kernel	2018-01-29 15:00:45.876 799 754	kernel_1	1	syscall_entry_write	fd=28, buf=139984728244528, count=4096, context._perf_cpu_migrations=23

Example UI (Trace Compass)

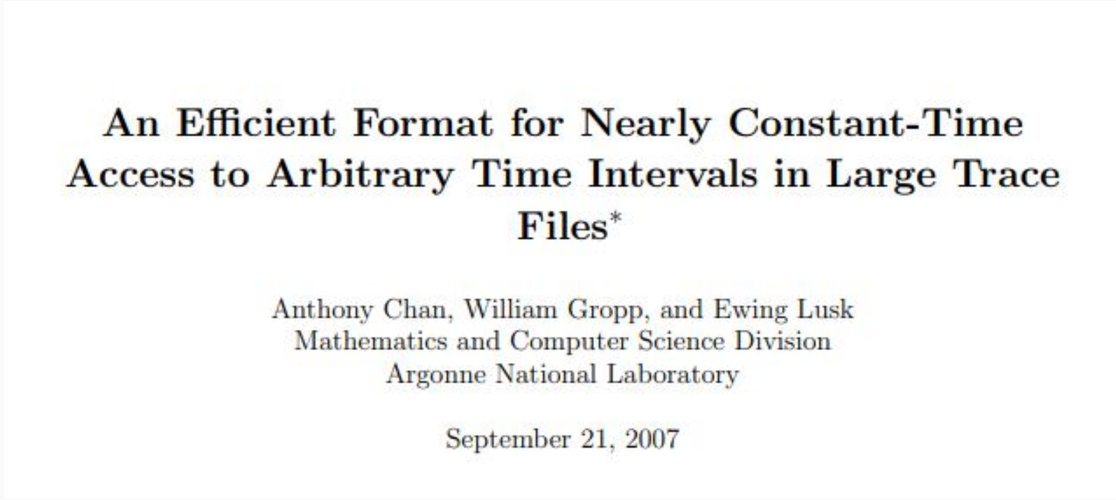
Integrated Operation (Example)

Integrated changes (across kernel & userspace) are one of BSD's strengths:

- E.g. control PMC-1 based on PMC-2 [e.g. turn on/off PMC-1 on PMC-2's overflow].
- Turn on/off PMC measurements from DTrace triggers.
  - E.g. measure L3 misses along a **specific** kernel execution path.
- **Or:** Turn on/off DTrace triggers based on PMC measurements.
- Needs low-overhead extensibility:
  - In-kernel and in userspace, with type safety and efficient execution.
  - BPF-like but easier to program in?
  - Easy scripting of 'performance hypotheses' during performance debugging?

Automated Analysis of Traces

- Support batch-style automated analysis of performance data.
- Using an extensible data format.
- Efficient to query, e.g. SLOG-2.



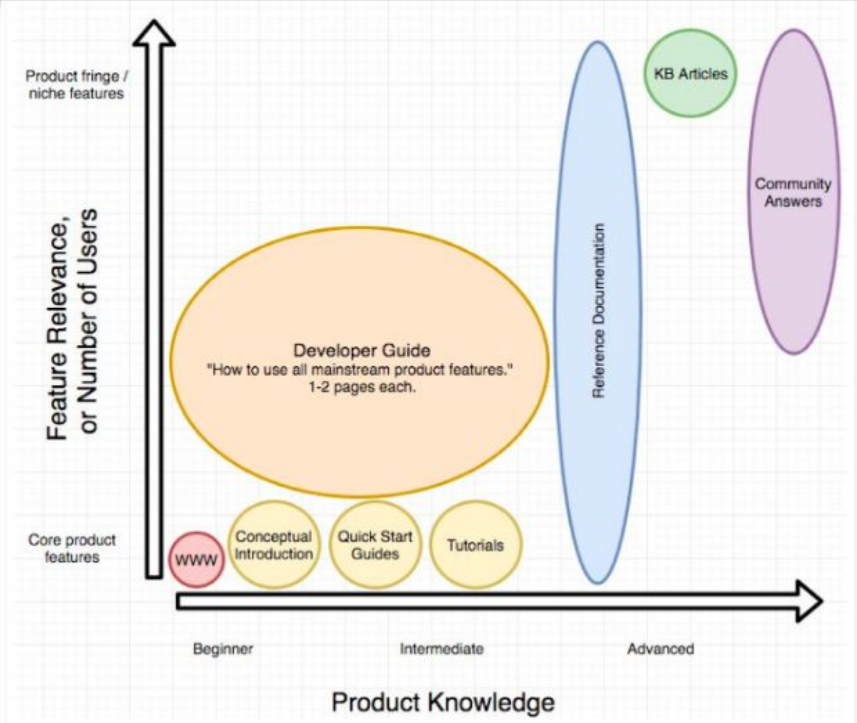
An Efficient Format for Nearly Constant-Time Access to Arbitrary Time Intervals in Large Trace Files*

Anthony Chan, William Gropp, and Ewing Lusk
Mathematics and Computer Science Division
Argonne National Laboratory

September 21, 2007

Documentation

- Effective documentation is crucial.
- Matrix of documentation types
    - From: [Cameron Shorter, 2018]().
- Documentation to be written:
    - CPU/System docs (ideally from system vendors).
    - Generic tutorials on using these tools.
    - System-specific 'How Tos' and overviews.
    - Task-specific 'How Tos'.
- Processes to manage the above.



Sustainability

Bring in new people to *BSD:

- Hopefully these tools would help students gain excellent visibility into *BSD system behavior.

Sustainability for developers themselves:

- Being looked at.
- Hopefully more sustainable than 2004/05.

Interested?

Please get in touch: jkoshy@FreeBSD.org, or jkoshy@NetBSD.org.