

Profiling Tutorial & DEMO

Jigao Luo

2025-06-13

Systems Group @ Technische Universität Darmstadt

Motivation & Background	2
Command-line tool: perf	8
Header: perf-event	14
Library: perf-cpp	19
EXTRA: “AI-HW Profiler” for CPU-code	26
More?	31

Motivation & Background

Modern CPUs: complex 🧠

- Complex behavior, hard to understand:
 - multi-issue: 6 Arithmetic Logic Units @ AMD Zen5
 - out-of-order
 - on-chip caches: L1 & L2
 - cache coherence: MESI ...
 - speculation
 - multiple cores
- **A black box**: need a model to predict performance

Modern CPUs: complex 🧐

- Complex behavior, hard to understand:
 - multi-issue: 6 Arithmetic Logic Units @ AMD Zen5
 - out-of-order
 - on-chip
 - cache
 - speculation
 - multi-ported

Quiz Time:

- What is IPC as a HW-counter?

- In Theory, 6 ALU. But how to reach it in practice?

- **A black box:** need a model to predict performance

Modern CPUs: complex 🧐

- Complex behavior, hard to understand:
 - multi-issue: 6 Arithmetic Logic Units @ AMD Zen5
 - out-of-order
 - on-chip
 - cache
 - speculation
 - multi-processor

Quiz Time:

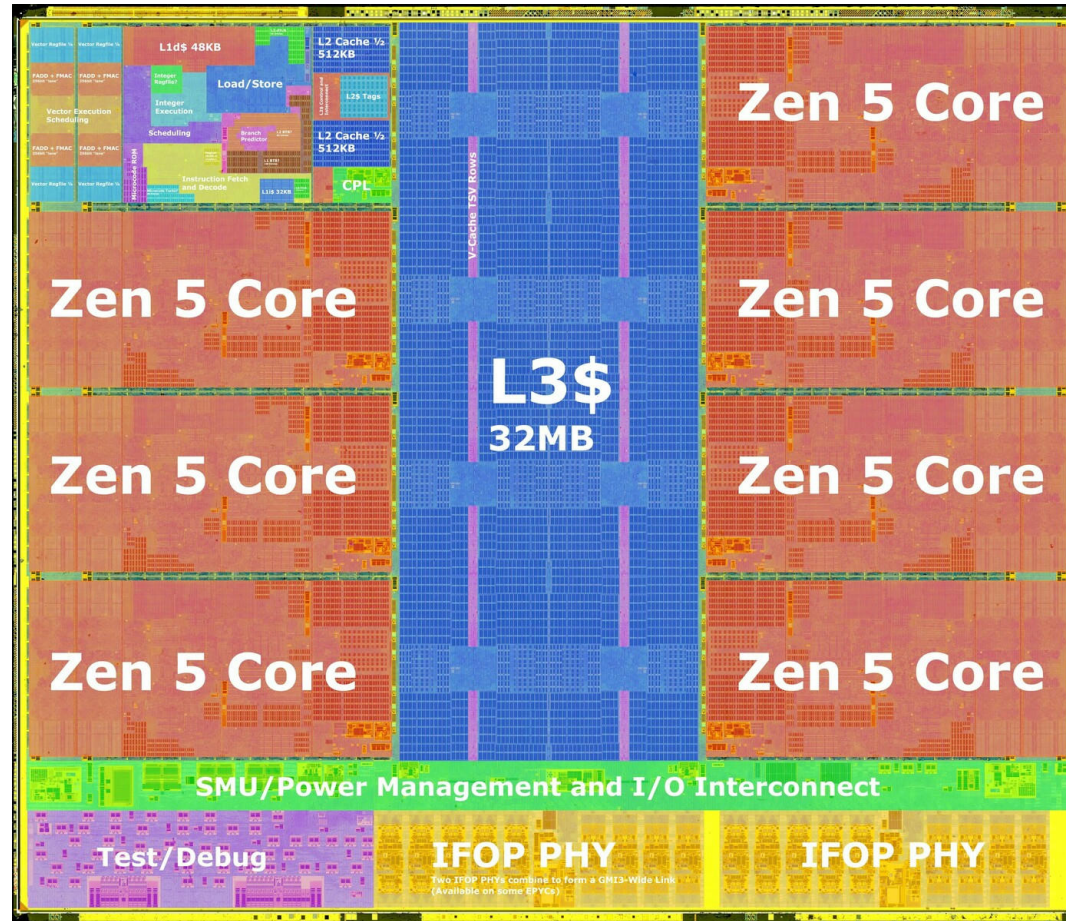
- What is IPC as a HW-counter?

- In Theory, 6 ALU. But how to reach it in practice?

- **A** | “In theory there is no difference between theory and practice.
In practice there is.” – Yogi Berra

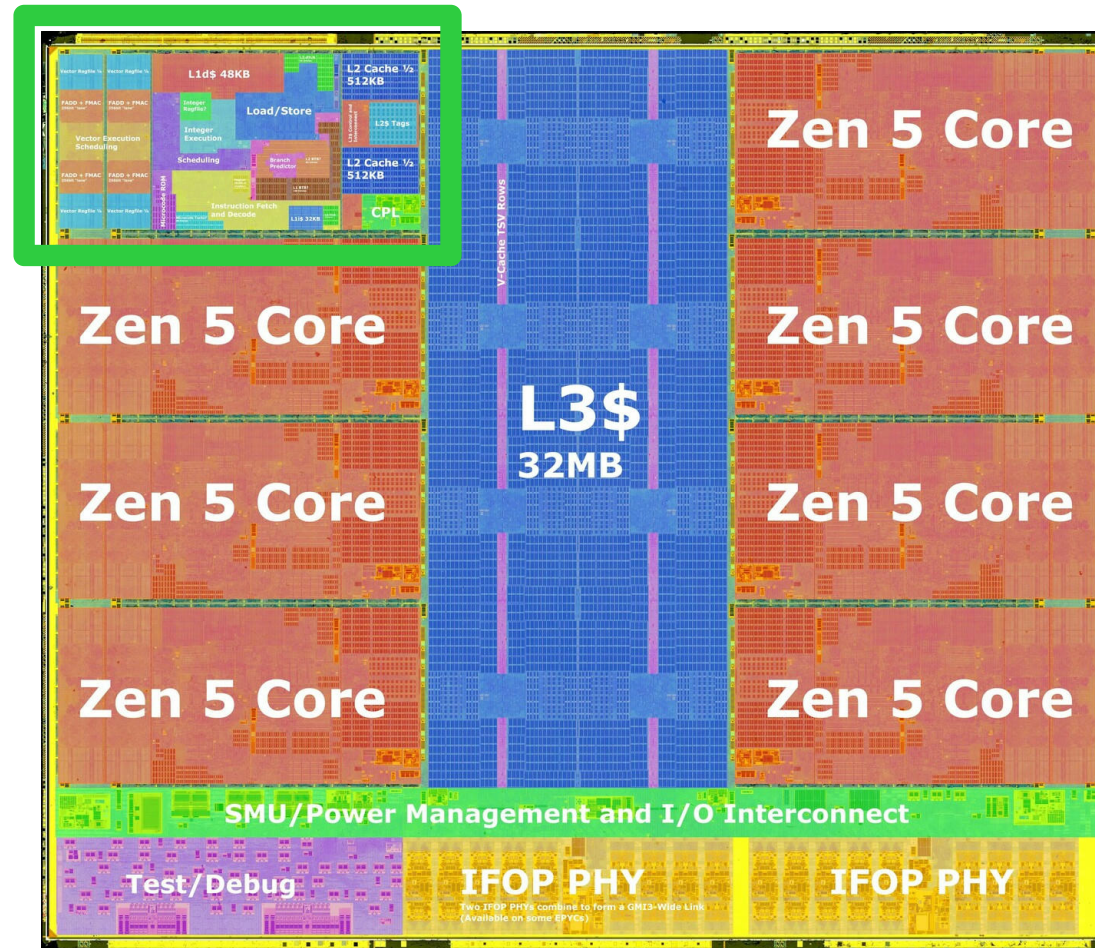
<https://pubmed.ncbi.nlm.nih.gov/24320775/>

AMD Zen5 Die-Shot



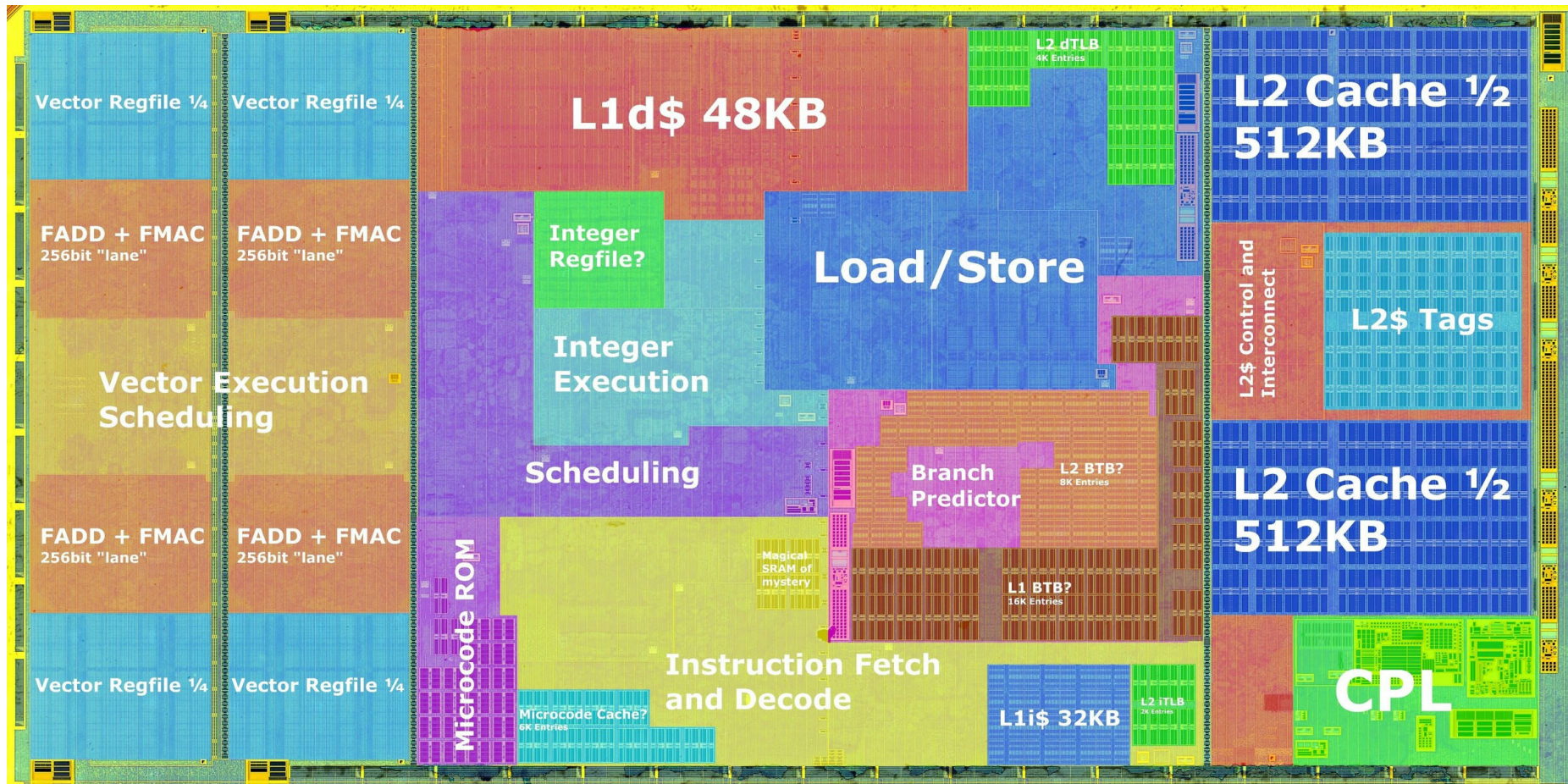
<https://www.techpowerup.com/327388/amd-granite-ridge-zen-5-processor-annotated>

AMD Zen5 Die-Shot



<https://www.techpowerup.com/327388/amd-granite-ridge-zen-5-processor-annotated>

AMD Zen5 Core-Shot: Zoom-in 🔍



<https://www.techpowerup.com/327388/amd-granite-ridge-zen-5-processor-annotated>

Hardware Events: for performance prediction

In CPU PMUs, hardware performance counters are collected:

- instructions: x86 machine instructions executed (“retired”)
 - CPU-cache hits & misses
 - TLB hits & misses
 - branch(-prediction) hits/misses
 - IPC
 - cycles & kernel-cycles (kcycles)
 - ...
-
- up to 4 counters can be measured simultaneously
(more require sampling/multiplexing)

Jaja, but why do I learn profiling?

- Learn system programming
- Improve programming skills
- Understand high-performance data-intensive applications
- Get a great job

Command-line tool: perf

```
# List of pre-defined events
```

```
$ perf list
```

```
# !Simply! gather performance counter statistics
```

```
#  statistisc are not normalized
```

```
$ perf stat <your_program>
```

perf record & report

Step1: record

Recording Performance Data:

```
$ perf record <your_program>
```

for specific events:

```
$ perf record -e cache-misses <your_program>
```

Step2: report

Analyzing Data & view profiling results:

```
$ perf report
```

Compile with debug Symbols

Preferable with symbols to match function name etc.

C++:

```
$ g++ -O3 -g -o my_program my_program.cpp  
$ cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ..
```



<https://makeameme.org/>



:

- Easy as CLI in Linux
- Out-of-box without code-changes



:

- Does profiling really start with reading assembly?
- No (or bad) mapping to code-segment
- What-if:
Dataset-generation burning cycles, but no need to optimize?

Header: perf-event

Single-Header

The screenshot shows the GitHub repository page for **viktorleis / perfevent**. The repository is public and has 5 watchers, 23 forks, and 124 stars. The main branch is **master**. The repository description is "An easy-to-use, header-only C++ wrapper for Linux' perf event API". The file list shows three files: **LICENSE** (added 7 years ago), **PerfEvent.hpp** (fixed indentation, last year), and **README.md** (make config more prominent, 3 months ago). The right sidebar contains links to the **Readme**, **MIT license**, **Activity**, and **124 stars**.

viktorleis / perfevent

Code Issues 1 Pull requests Actions Projects Security Insights

perfevent Public Watch 5 Fork 23 Starred 124

master Go to file + <> Code

About
An easy-to-use, header-only C++ wrapper for Linux' perf event API
Readme
MIT license
Activity
124 stars

viktorleis	make config more prominent	d03e3e5 · 3 months ago
LICENSE	add code and readme	7 years ago
PerfEvent.hpp	Fixed indentation	last year
README.md	make config more prominent	3 months ago

<https://github.com/viktorleis/perfevent>

Instrument CPU counters in source code

```
#include "PerfEvent.hpp"
```

```
BenchmarkParameters params; // Define global params  
params.setParam("name", "MyBenchmark");
```

```
/// Burning Cycles: dataset loading, generation ...
```

```
for (int th = 1; th < maxThreads; ++th) {  
    params.setParam("threads", numThreads); // Change local parameters  
    {  
        PerfEventBlock e(n, params, printHeader);  
        // Counter are started in constructor  
        yourBenchmark(); // !Profiling ONLY HERE!  
        // Benchmark counters are automatically stopped  
        // and printed on destruction of e  
    }  
}
```

DEMO TIME: perf-event



<https://makeameme.org/>



:

- Only needed code-segment benchmarked!
- Easy with header-only
- Simple code-changes



:

- Bad mapping to Code/Application-Knowledge:
 - Let's say nodes in a OLC-Btree?

Library: perf-cpp

A library: static or shared

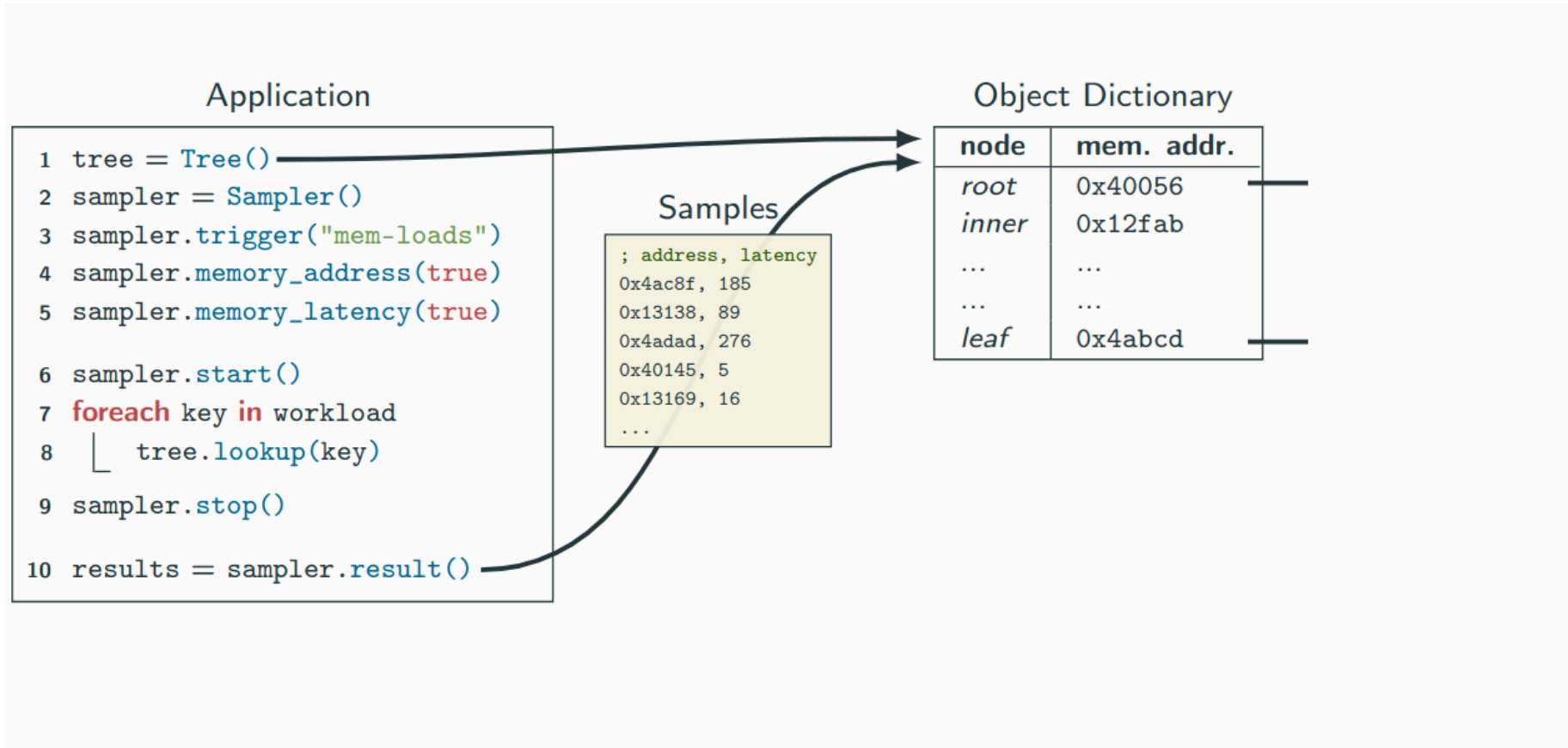
The screenshot shows the GitHub repository page for `jmuehlig/perf-cpp`. The repository is public and has 2 watchers, 8 forks, and 66 stars. The main navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. The repository name `perf-cpp` is displayed with a 'Public' badge. Below the repository name, there are buttons for 'dev' (selected), 'Go to file', and 'Code'. The 'About' section describes the project as 'Lightweight recording and sampling of performance counters for specific code segments directly from your C++ application.' and lists tags: `linux`, `processor-architecture`, `library`, `performance`, `cpp`, `perf`, `performance-metrics`, `cpp17`, `performance-analysis`, and `sampling`. The file list shows the following structure:

File/Folder	Description	Time
<code>docs</code>	Added changelog and set version t...	2 weeks ago
<code>examples</code>	Fixed inconsistencies between tim...	2 weeks ago
<code>include/perfcpp</code>	Fixed inconsistencies between tim...	2 weeks ago
<code>script</code>	Added script to read all hardware-...	last year

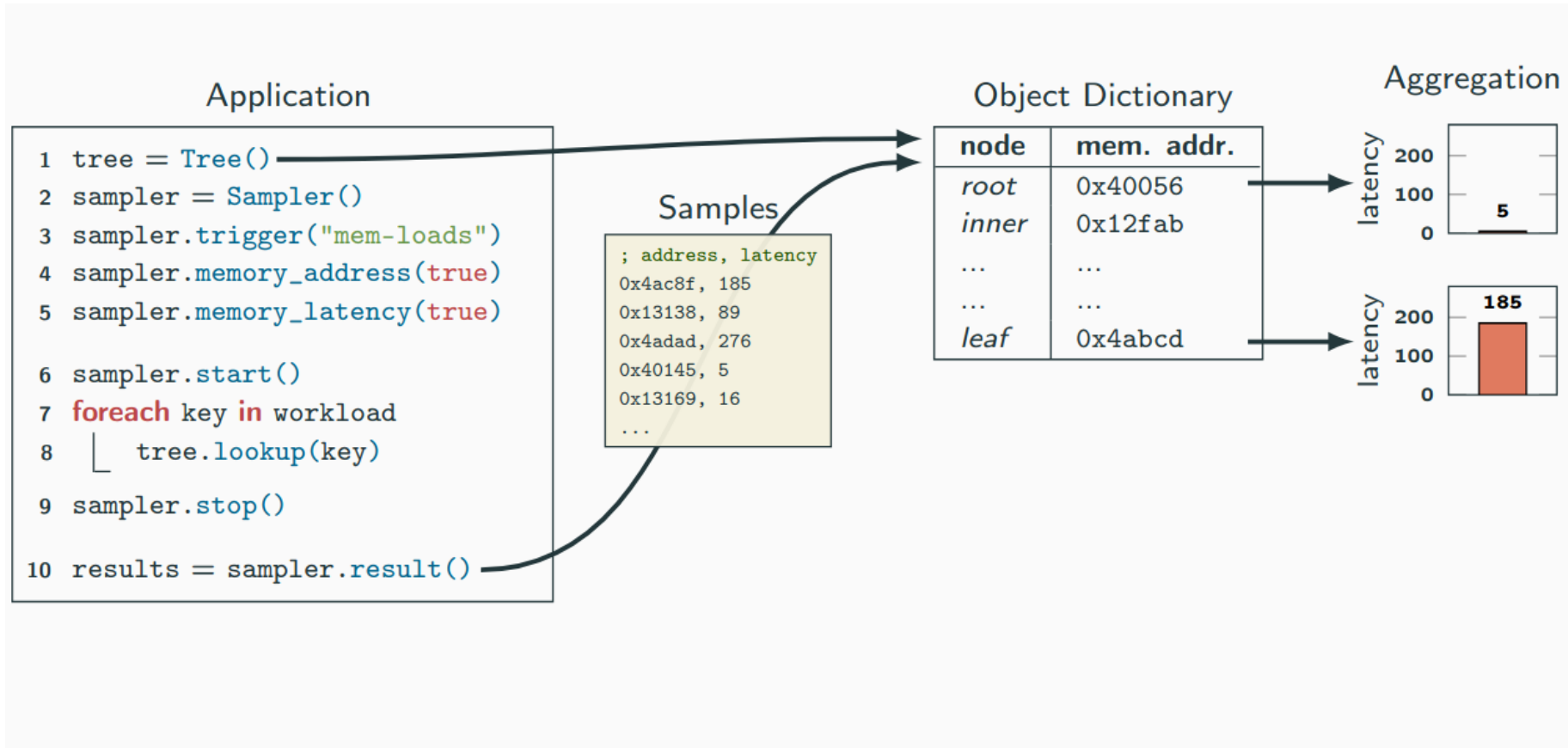
<https://github.com/jmuehlig/perf-cpp>

<https://github.com/jmuehlig/nodmc25-profiling-tutorial>

Profiling with Knowledge of OLC B+tree: Node-mapping



Profiling with Knowledge of OLC B+tree: Node-mapping



Profiling with Knowledge of OLC B+tree: Results

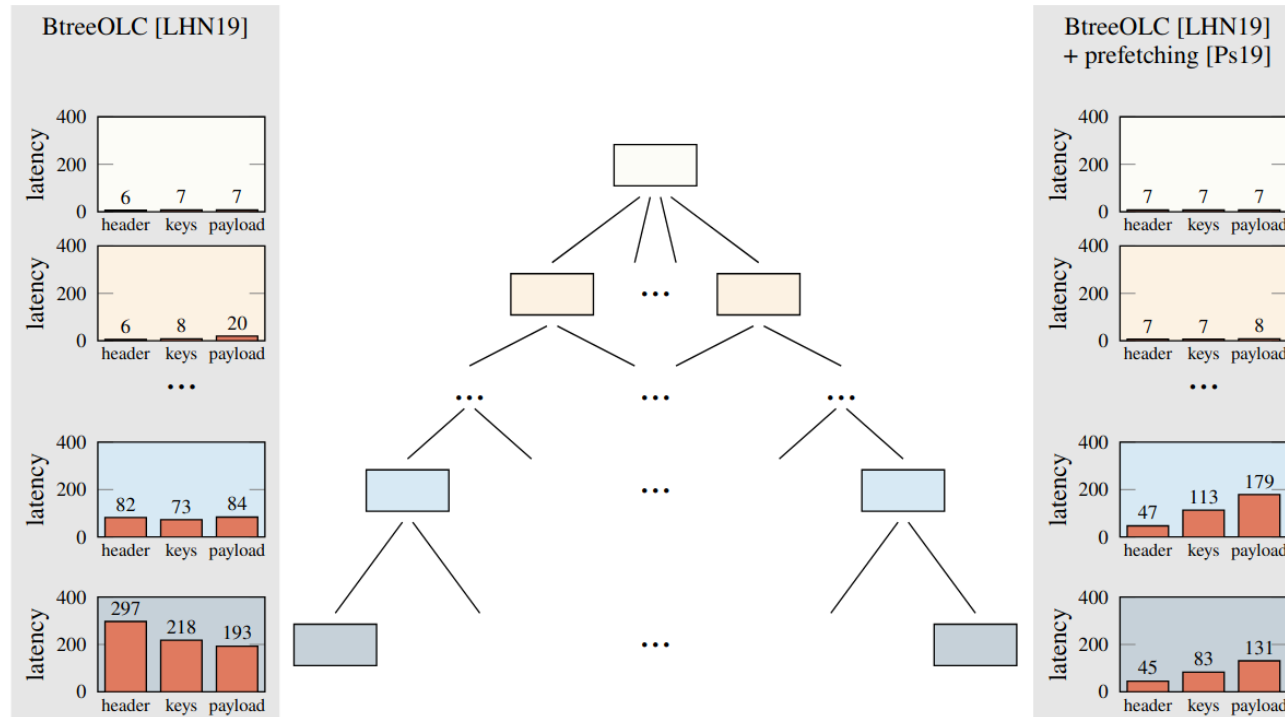


Fig. 4: Memory access latency for different node-segments (header, keys, payloads) in various tree levels derived from sampling—*without* prefetching (left) and *with* software-based prefetching (right).



<https://makeameme.org/>



:

- Yes. Application-Knowledge!
- Nice functionality



:

- (In my experience) Hard to compile&run it on:
 - diff. CPU: Intel and AMD
 - diff. Kernel versions

EXTRA: “AI-HW Profiler” for CPU-code

EXTRA: “AI-HW Profiler” for CPU-code

With “AI-HW”, I mean GPUs.



NVIDIA Nsight Systems

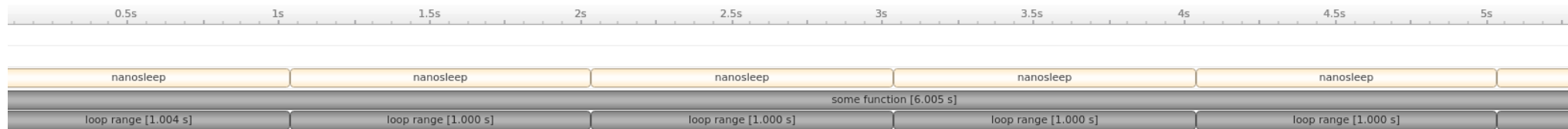
NVIDIA Nsight™ Systems is a system-wide performance analysis tool designed to visualize an application's algorithms, identify the largest opportunities to optimize, and **tune to scale** **efficiently across any quantity or size of CPUs** and GPUs, from large servers to our smallest systems-on-a-chip (SoCs).

<https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cpu-profiling-on-linux>

Runtime Breakdown with NVTX

```
#include <nvtx3/nvtx3.hpp>
```

```
void some_function() {  
    NVTX3_FUNC_RANGE(); // Range around the whole function  
    for (int i = 0; i < 6; ++i) {  
        nvtx3::scoped_range loop{"loop range"}; // Range for iteration  
        std::this_thread::sleep_for(std::chrono::seconds{1}); // Sleep 1s  
    }  
}
```



<https://github.com/NVIDIA/NVTX>



<https://makeameme.org/>



- GUI showing overview and runtime-breakdown
 - C++ (Header-only) & Python (import-only)
 - Bus Monitoring: PCIe (SSD, Network), NVLink...
-
- (I use it everyday. But what I do everyday?)



- Limited CPU metrics.
- Need to install it via NVIDIA.



- GUI showing overview and runtime-breakdown
 - C++ (Header-only) & Python (import-only)
 - Bus Monitoring: PCIe (SSD, Network), NVLink...
-
- (I use it everyday. But what I do everyday?)



- Limited CPU metrics.
- Need to install it via NVIDIA.

Lab2 Hint: Runtime breakdown. Everything is memory-bound there. 

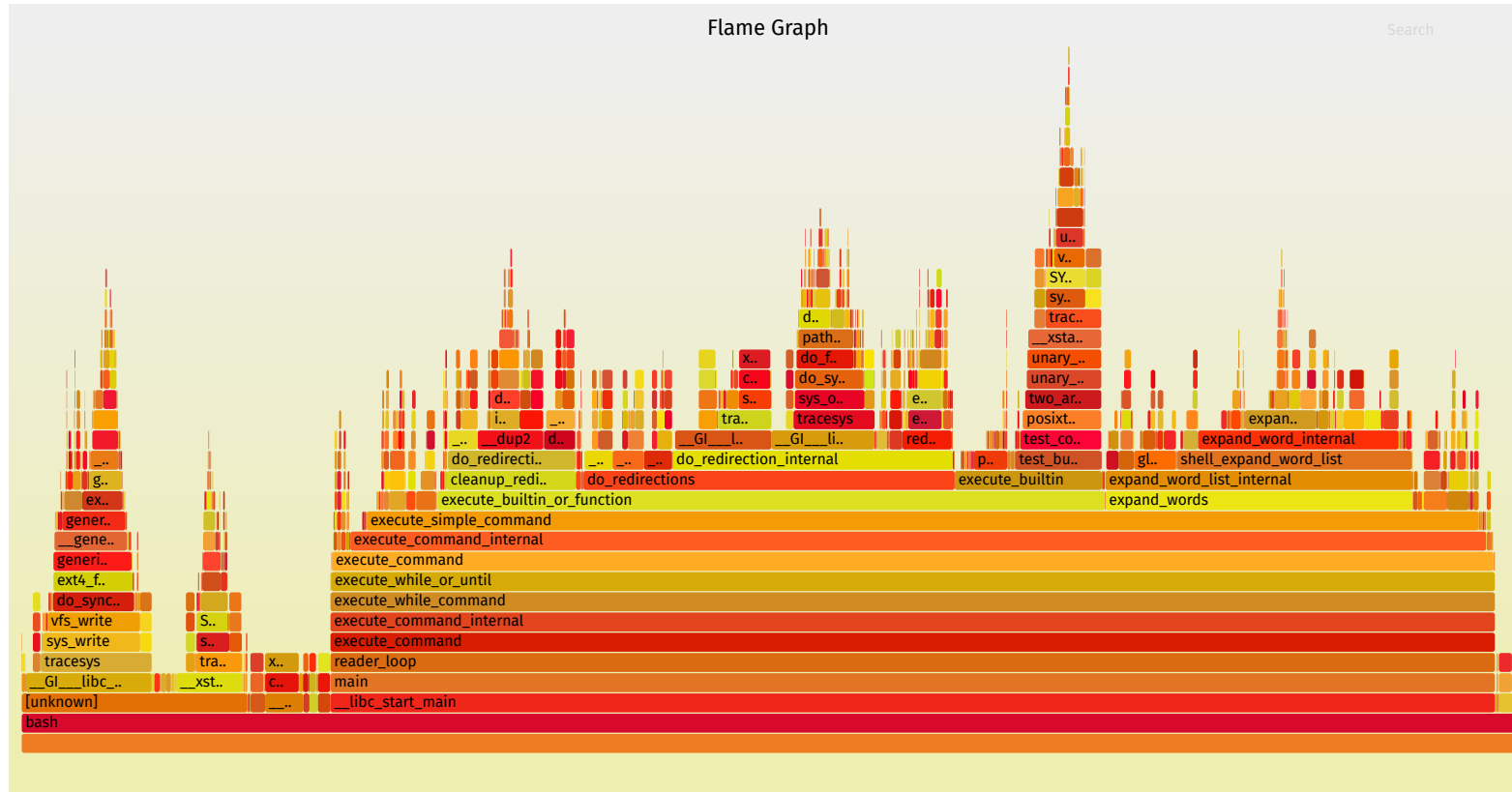
More?



Not covered, but also important

- Details about profiling: PMUs, Sampling, Abstractions... Read the NoDMC paper & perf-cpp documentations
- Debugging & Profiling:
 - Compiler Explorer: <https://godbolt.org/>
 - valgrind & gdb
- CPU:
 - `lscpu`, `lstopo`
 - `/proc/{meminfo, cpuinfo}, dmidecode -t {memory, cache}`
 - perf Flamegraph
 - NUMA with `numactl`
 - perfetto for GUI: <https://perfetto.dev/docs/quickstart/linux-tracing>
- Discussion? 😊

Backup: Flamegraph



<https://github.com/brendangregg/FlameGraph>

His website and books are also very nice!

Resources

- Understanding Application Performance on Modern Hardware: Profiling Foundations and Advanced Techniques - Mühlig, Jan; Kühn, Roland; Teubner, Jens (TU Dortmund, NoDMC@BTW25)
- <https://github.com/jmuehlig/perf-cpp?tab=readme-ov-file#resources-about-perf--profiling>
- Ulrich Drepper's What Every Programmer Should Know About Memory
- Agner Fog's Software optimization resources
- Intel's Top-Down Microarchitectural Analysis Method

Many thanks to Jan Mühlig for perf-cpp and sharing the slide, Matthias Jasny and Tobias Ziegler for sharing ADMS slides, Viktor Leis for perf-event, and all mentioned resources!