Linaro Connect
MADRID 2024 | MAY 14-17 2024

Enhancements in WindowsPerf

Everton Constantino (Linaro)
Przemyslaw Wirkus (arm)

Introduction

- WindowsPerf is Windows on Arm project.
- Linux perf inspired lightweight WOA performance profiling tool. Profiling is based on ARM64 PMU (Performance Monitor Unit) and its hardware counters.
- Technical objective:
  - Provide Arm PMU based performance monitoring tool inspired by Linux perf.
  - Support core PMU, DSU and DMC, and more.
- Architecture:
  - “wperf-driver” - signed Windows Kernel Driver.
  - “wperf”: Command line tool.

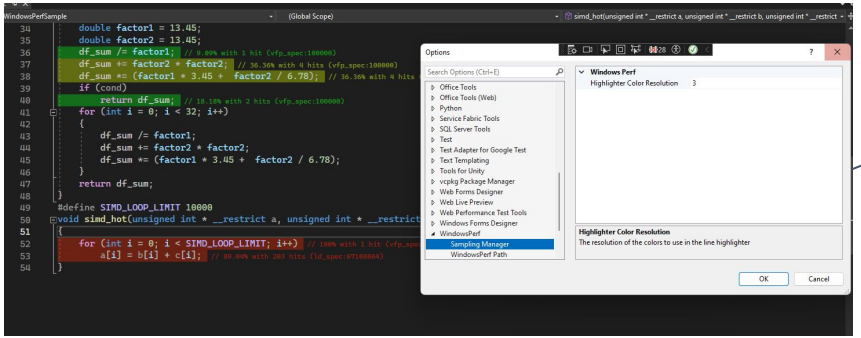
WindowsPerf + Telemetry Solution

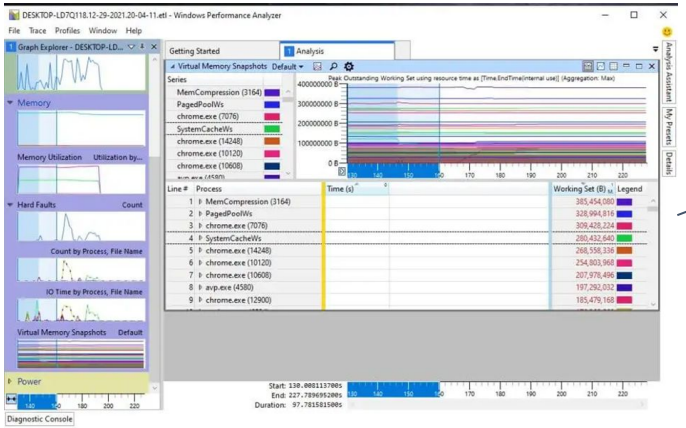
- **Topdown-tool**.
  - Example top-down analysis for both WindowsPerf and Linux perf (as a backend).
- Telemetry Solution JSON meta-data integration with WindowsPerf.
  - Telemetry Solution meta-data integration with WindowsPerf CLI.
  - Events, metrics and groups of metrics.
  - Auto-detection of supported CPUs (neoverse n1/v1/n2).
- **ustress** integration with WindowsPerf test suite.
  - WOA Native ustress built from sources (clang).
  - Micro-benchmarks are used as workloads and TS metrics are used to monitor those.
  - Indirect proof of WindowsPerf correctness.

WindowsPerf Design Decisions

- WindowsPerf:
  - BSD permissive license.
  - Full integration with Telemetry Solution: events, metrics, group of metrics, platform detection.
  - Integration with existing Windows ecosystem: WPR, WPA, ETW.
- Kernel driver:
  - Windows Kernel driver wrapps access to Arm hardware.
  - Mutually exclusive access to the driver (lock/unlock feature).
  - Smart PMU resources acquire / release.
- Command line tool:
  - Command line interface and console output inspired by Linux perf.
  - `wperf` outputs results in JSON format.

WindowsPerf Ecosystem





VS-extension

WindowsPerf GUI orchestrates performance measurement with WindowsPerf back-end.

WPA-plugin

Consumes WindowsPerf JSON input and generates plotting.

WindowsPerf

Performance measurements. Output JSON.

JSON

The road from London to Madrid

April 2023 - May 2024

Where were we at Linaro Connect 2023?

WindowsPerf 2.4.0 (April 2023)

Release highlights:

- **Sampling**.
- DLL symbol resolution.
- New tool **wperf-devgen** to the project. It simplifies how users install/uninstall our Kernel driver.
- First proper binary release of the wperf-driver.
- Improved Kernel driver stability.
- Improvements to timeline feature.
- JSON output.

Where are we and what were the major changes?

WindowsPerf 3.3.3 (March 2024) - Stable release

We had more than 9 releases with bug fixes and new features

- Major app and bug fixes, memory leaks removed and stability highly improved.
- We no longer require all GPCs to be available to install the driver.
- New `record` command added. Processes can now be spawned directly from the command line and pinned to a single core. This works with both sampling and counting.
- Basic export to `perf.data` file format.
- Introduction of the C/C++ `lib` allowing users to add software tracing to their software without having to spawn the CLI.
- Basic driver configuration through command line with `--config`
- Major upgrades to our testing infrastructure.
- Disassembly support for sampling.

WindowsPerf 3.5.0 (April 2024) - Beta release

- Locking and unlocking driver access
- Acquire and release PMU resources
- Early seeds for Device Tree support
- Prototype for ETW output

`Record` - spawning processes from CLI

- Users are no longer required to spawn the process manually to sample it. Neither is the user required to set the process's affinity mask manually.
- When using hardware with more than 64 cores Windows break the cores into processor groups making it quite hard to using the `start /affinity` command as you cannot specify the processor group.
- It is not possible to directly set the spawned process's core affinity with the required flexibility so the solution was to actually changed the individual thread's affinity. This enabled WindowsPerf to pin workloads to any core.
- This feature is available in sampling through the `record` command and also with `stat`. All you need to do is set the CPU with `-c` and by the end of the command just append `--` to start defining the command line of the process that is going to be spawned, e.g. `wperf record -e vfp_spec -c 0 -- WindowsPerfSample.exe`

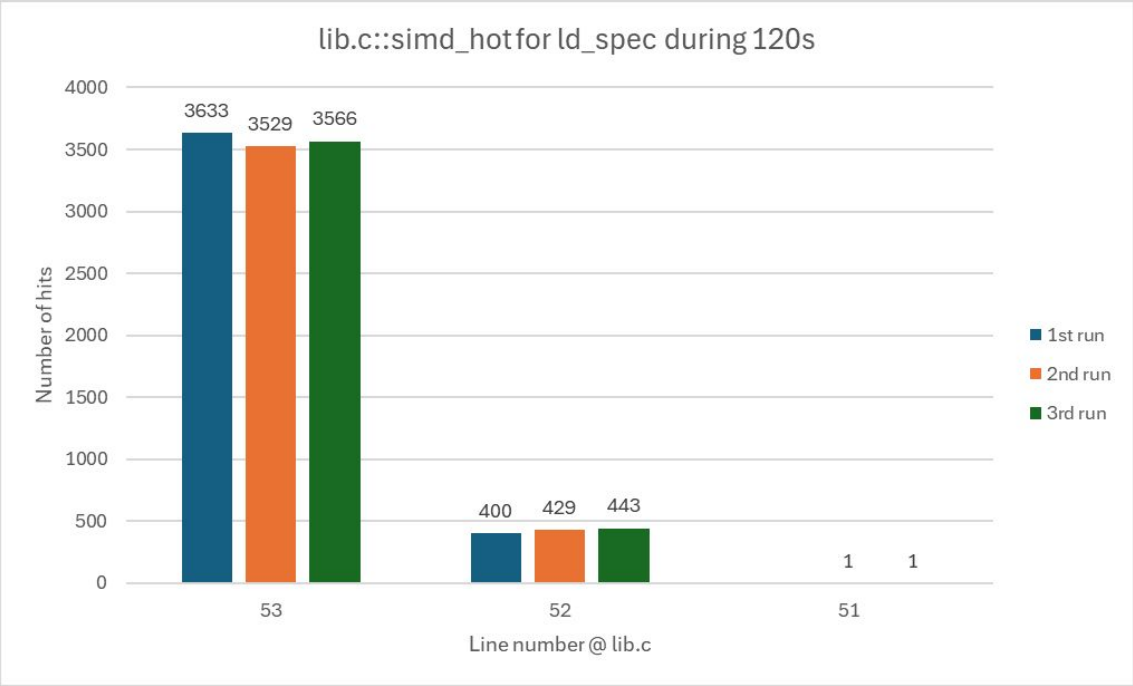
WindowsPerfSample - simd_hot

https://gitlab.com/everton.constantino/windowsperfsample

```
#define SIMD_LOOP_LIMIT 10000
50: void simd_hot(unsigned int * __restrict a, unsigned int * __restrict b, unsigned int *
__restrict c)
51: {
52:     for (int i = 0; i < SIMD_LOOP_LIMIT; i++)
53:         a[i] = b[i] + c[i];
54: }
```

Sampling WindowsPerfSample

```
wperf record -e ld_spec:FREQUENCY -c 0 --timeout 120 --annotate -- WindowsPerfSample.exe
```



Disassembly output support for sampling

- Using the Debug Interface Access SDK along with LLVM's objdump we read the PDB files available and try to show the disassembly around the code hotspots for each event.
- The results are separated per event per function. Due to address skid sometimes the address is not 100% accurate.

Disassembling WindowsPerfSample's `simd_hot`

```
base address of '.\WindowsPerfSample1.exe': 0x7ff6063415c4, runtime delta: 0x7ff4c6330000
sampling ......Ctrl-C received, quit counting... done!
====================== sample source: ld_spec, top 50 hot functions ======================
simd_hot
       line_number  hits  filename                                          instruction_address  disassembled_line
       ===========  ====  ========                                          ===================  ================
       53           83    C:\Users\evert\source\repos\WindowsPerfSample\lib.c  13a34                  address  instruction
                                                                                                   =======  ===========
                                                                                                   13a30    ldr      q18, [x19], #0x40
                                                                                                   13a34    add      v16.4s, v16.4s, v18.4s
                                                                                                   13a38    stur     q16, [x9, #-0x20]
                                                                                                   13a3c    ldr      q16, [x13, x8]
                                                                                                   13a40    add      v16.4s, v16.4s, v17.4s
                                                                                                   13a44    ldr      q17, [x11, x9]
                                                                                                   13a48    str      q16, [x12, x8]
                                                                                                   13a4c    ldp      q16, q18, [x8, #0x10]
                                                                                                   13a50    add      x8, x8, #0x40
                                                                                                   13a54    add      v17.4s, v17.4s, v16.4s
                                                                                                   13a58    ldur     q16, [x19, #-0x10]
                                                                                                   13a5c    add      v16.4s, v18.4s, v16.4s
                                                                                                   13a60    stp      q17, q16, [x9], #0x40
                                                                                                   13a64    cbnz     w10, 0x140013a28 <.text+0x2a28>
                                                                                                   13a68    ldp      x29, x30, [sp], #0x10
                                                                                                   13a6c    ldr      x21, [sp, #0x10]
                                                                                                   13a70    ldp      x19, x20, [sp], #0x20
                                                                                                   13a74    ret
       52           8     C:\Users\evert\source\repos\WindowsPerfSample\lib.c  13a28                  address  instruction
                                                                                                   =======  ===========
                                                                                                   13a28    ldp      q16, q17, [x8, #-0x10]
                                                                                                   13a2c    sub      w10, w10, #0x1

       overhead  count  symbol
       ========  =====  ======
         100.00     91  simd_hot
100.00%        91  top 1 in total
```

WindowsPerf Library

- **What it is not** - A way to directly access the WindowsPerf driver interface bypassing the WindowsPerf application commands.
- **What it actually is** - A way to replicate the CLI functionality directly with functions like `wperf_sample`, `wperf_stat`, `wperf_list`, `wperf_version`
- Majorly contributed by Qualcomm team

Locking and unlocking driver access

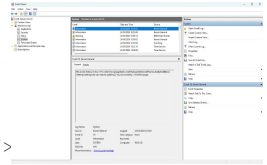
- This enables easier control over other people stopping workflows that run for a considerable amount of time.
- Up until version 3.3.3 the WindowsPerf Driver is set to have exclusive access. This means that no calls to `DeviceIoControl` can be made if one is already in execution.
- The driver has no way to know that another application is requiring access, this makes it impossible to even open the application as communication with the driver is required for startup.
- Version 3.5.0 onwards includes a new lock/unlocking mechanism. When a call is made to the driver it checks it's internal state to verify if it is in use and if the current owner matches with file handle stored at the acquisition of the lock.
- In case an error occurs there is the possibility of running a command with `--force-lock` to clear all current locks in the driver.

Acquire and releasing PMU resources

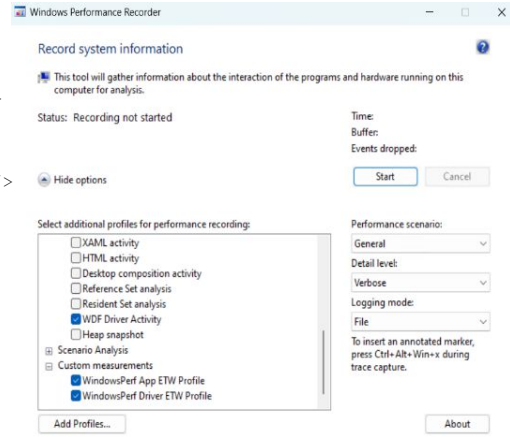
- Up until version 3.3.3 when the WindowsPerf driver was installed it queried the OS for available GPCs and allocated all of them. This persisted for as long as the driver was installed and aimed at test devices that had dedicated GPCs.
- WindowsPerf grew and we now understand that it needs to co-exist with other Windows ecosystem performance tools like XPerf.
- XPerf is part of a wider range of tools that live inside the Windows Performance Toolkit and can, as one of its functionalities, measure PMU events.
- To make sure XPerf, and other tools, could be enabled it was required that WindowsPerf only allocated GPCs during the time the commands were actually being executed. We decided, for now, that when the lock lifetime was also going to be the lifetime for the PMU resource allocation.
- Now users can start XPerf and then run WindowsPerf which will make sure to just use remaining resources.

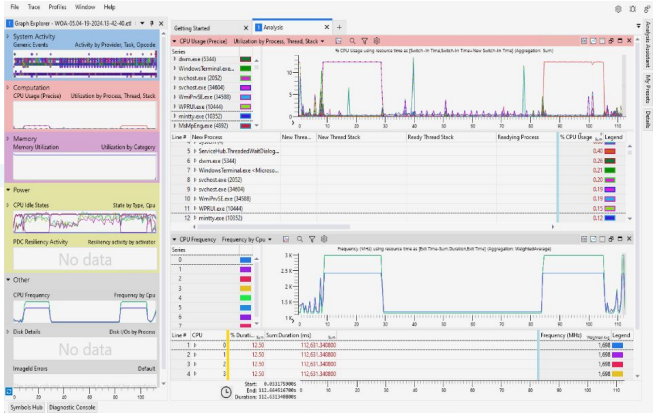
Intermission - Windows Performance Toolkit

- Microsoft's ETW or Event Tracing for Windows is a mechanism to trace and log events. There are several tools and APIs to read and write to ETW, the most readily available is EventViewer.



```
<Event
xmlns="http://schemas.microsoft.com/win/2004/08/events/event ">
        <System>
                <Provider Name="X" Guid="X" EventSourceName="X" />
                <EventID Qualifiers="X">X</EventID>
                <Version>0</Version>
                <Level>4</Level>
                <Task>0</Task>
                <Opcode>0</Opcode>
                <Keywords>0x8080000000000000</Keywords>
                <TimeCreated SystemTime="X" />
                <EventRecordID>95081</EventRecordID>
                <Correlation />
                <Execution ProcessID="X" ThreadID="X" />
                <Channel>X</Channel>
                <Computer>X</Computer>
                <Security UserID="X" />
        </System>
        <EventData>
                <Data Name="param1">X</Data>
                <Data Name="param2">X</Data>
                <Data Name="param3">X</Data>
                <Data Name="param4">X</Data>
        </EventData>
</Event>
```



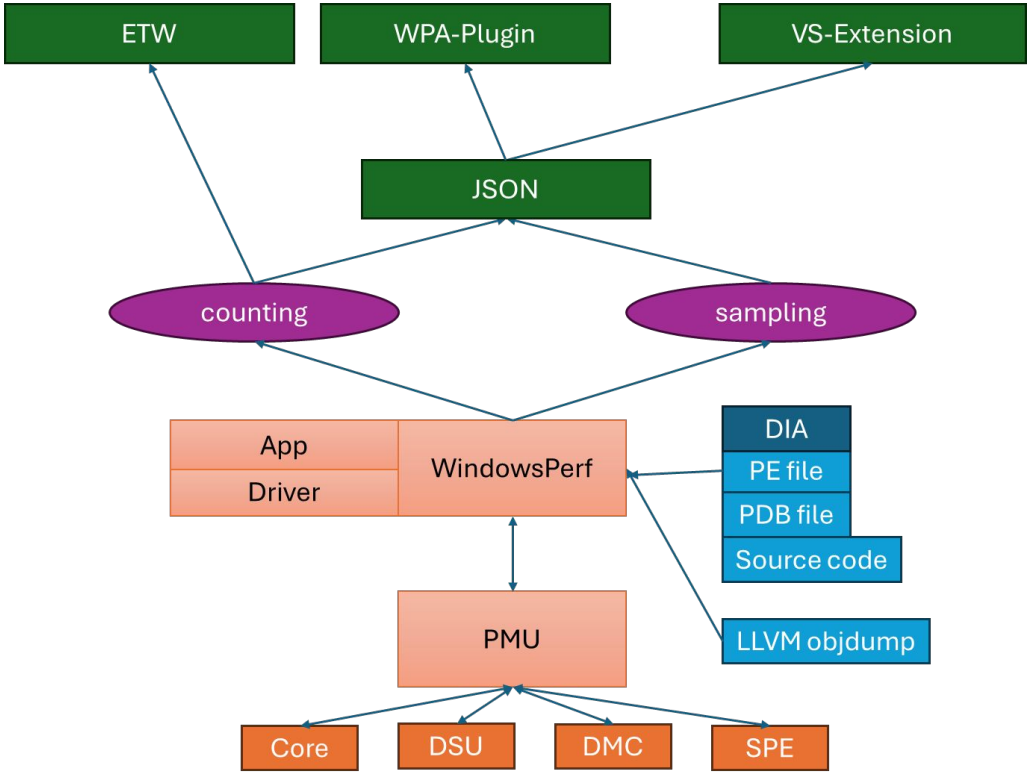


WindowsPerf PMU tracing with ETW output - Beta feature

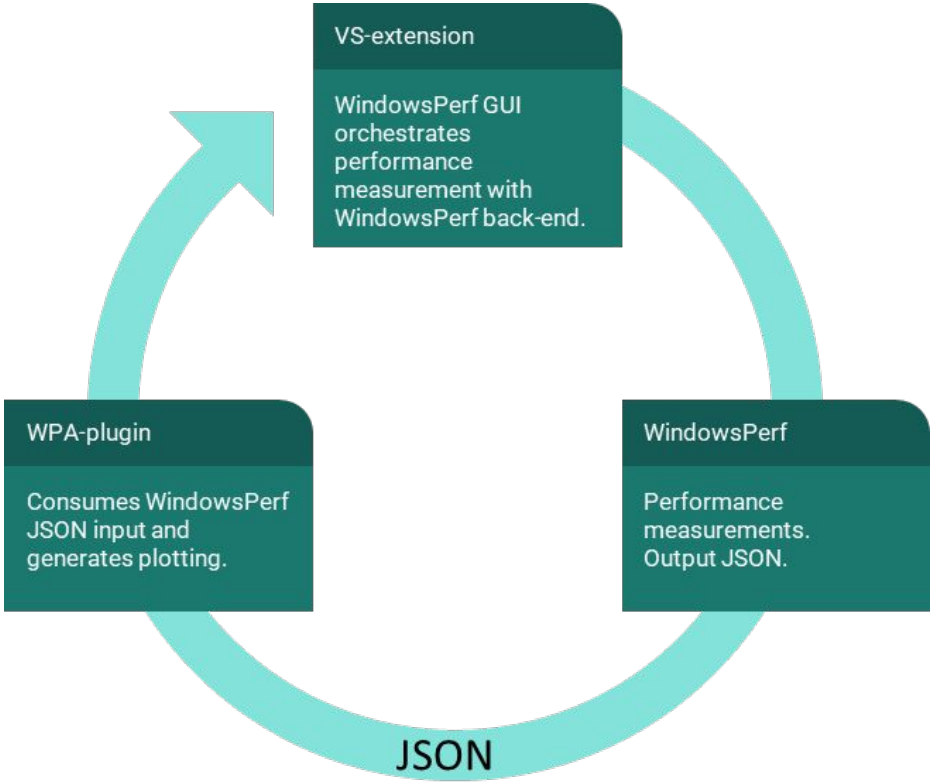
- Now the WindowsPerf App and Driver each can be controlled to output to ETW via a build configuration macro called `ENABLE_ETW_TRACING` and `ENABLE_ETW_TRACING_APP`
- We plan to release driver versions without ETW enabled to reduce overhead.
- The data template for the driver and application is different because they have distinct information about the events being traced. The application trace can provide more details but the driver is more precise.
- In order to make the Windows Performance Analazyer properly interpret the events we developed the WPA-plugin-ETL that is aware of the data template, parses them and produces the required tables and graphs.

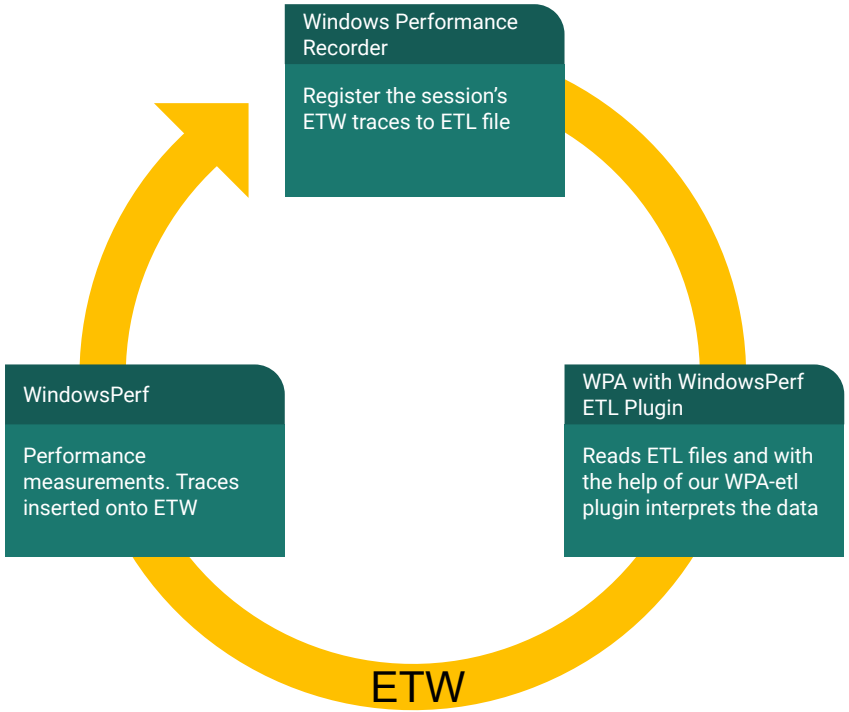
App and Driver ETW output in WPA with our custom ETL plugin

Overview


ETW
WPA-Plugin
VS-Extension
JSON
counting
sampling
App
Driver
WindowsPerf
DIA
PE file
PDB file
Source code
LLVM objdump
PMU
Core
DSU
DMC
SPE

WindowsPerf Ecosystem

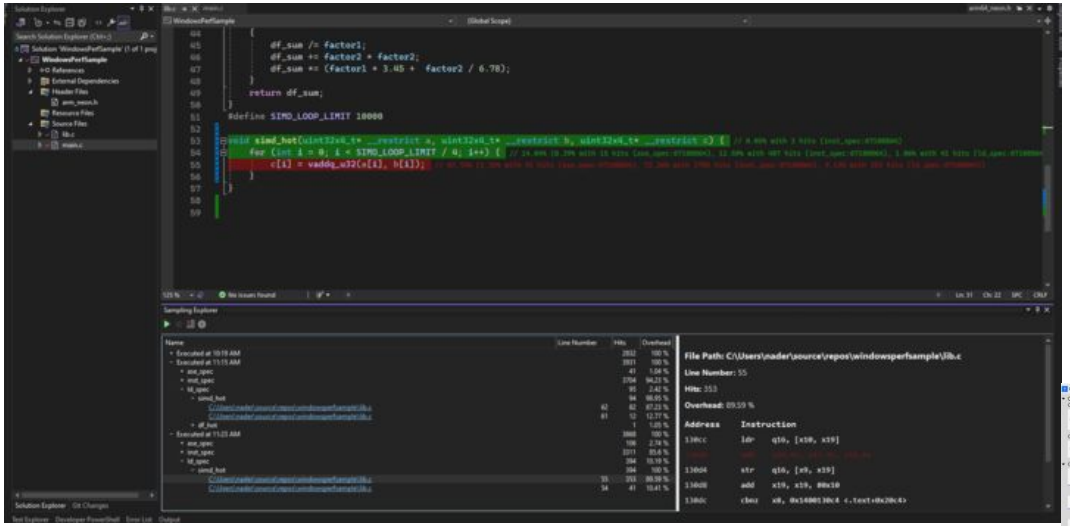


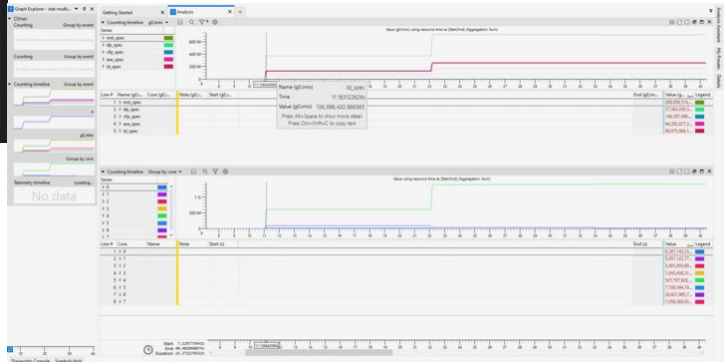


Device Tree - The road towards custom IP blocks

- Writing any changes to the current driver is complicated and very error prone.
- Customers want to write drivers for their own IP blocks but keep all the user level features.
- Version 3.5.0-beta adds the `detect` command. Current architecture required all drivers to use the same GUID as the WindowsPerf Driver.
- Drivers should provide their capabilities as `core`, `dsu`, `dmc`, or `spe` along with operations they provide like `stat` or `sampling`.

Visual Studio Extension and WPA-Plugin





WindowsPerf Roadmap to 4.x

- Driver resource lock / Unlock **[DONE]**.
  - WindowsPerf user-space process is now exclusive with lock in the driver.
  - Concurrent `wperf` process will gracefully exit if other `wperf` process is „counting”.
- Arm Statistical Profiling Extension (SPE) investigation and support **[IN PROGRESS]**.
  - The Statistical Profiling Extension is an optional feature in ARMv8.2.
  - Requires both Kernel Driver and user-space application improvements.
  - PMU(s) suffers from a problems of event skid and blind spots.
- Output to Event Tracing for Windows (ETW) trace stream **[IN PROGRESS]**.
  - ETW provides a mechanism to trace and log events that are raised by user-mode applications and kernel-mode drivers.
- Disassembly on annotate **[DONE]**.
  - Increase sampling resolution to instruction level.
- Beyond 4.x:
  - Packaging to a installer: bundle of WindowsPerf + VS-extension + WPA-plugin.
  - “Device tree” feature - support many WindowsPerf device drivers.
  - Other IP blocks.

Reference

Blog Posts

- Introducing 1.0.0-beta release of WindowsPerf Visual Studio extension
- Introducing the WindowsPerf GUI: the Visual Studio 2022 extension
- Announcing WindowsPerf: Open-source performance analysis tool for Windows on Arm
- WindowsPerf Release 2.4.0
- WindowsPerf Release 2.5.1
- WindowsPerf Release 3.0.0
- WindowsPerf Release 3.3.3

External Documentation

- Perf for Windows on Arm (WindowsPerf) @ Arm Learning path
- Get started with WindowsPerf @ Arm Learning Path
- Sampling CPython with WindowsPerf @ Arm Learning Path

Additional Resources

- Arm Telemetry Solution @ Arm Developer
- Topdown-tool Install Guide @ Arm Learning Path
- Arm CPU Telemetry Solution Topdown Methodology Specification @ Arm Developer

Linaro Connect
MADRID 2024 | MAY 12-17 2024

Thank you