PERFORMANCE ENGINEERING

Lecture 7 Performance counters

May 16, 2022

Ana Lucia Varbanescu

a.l.varbanescu@uva.nl

To do today

- Hardware performance counters
- Performance bottlenecks
- (?) Simulators as models

- Many more interesting PE topics
  - Distributed systems (lecture on Thursday!)
  - Next week: Queuing theory, the Polyhedral model

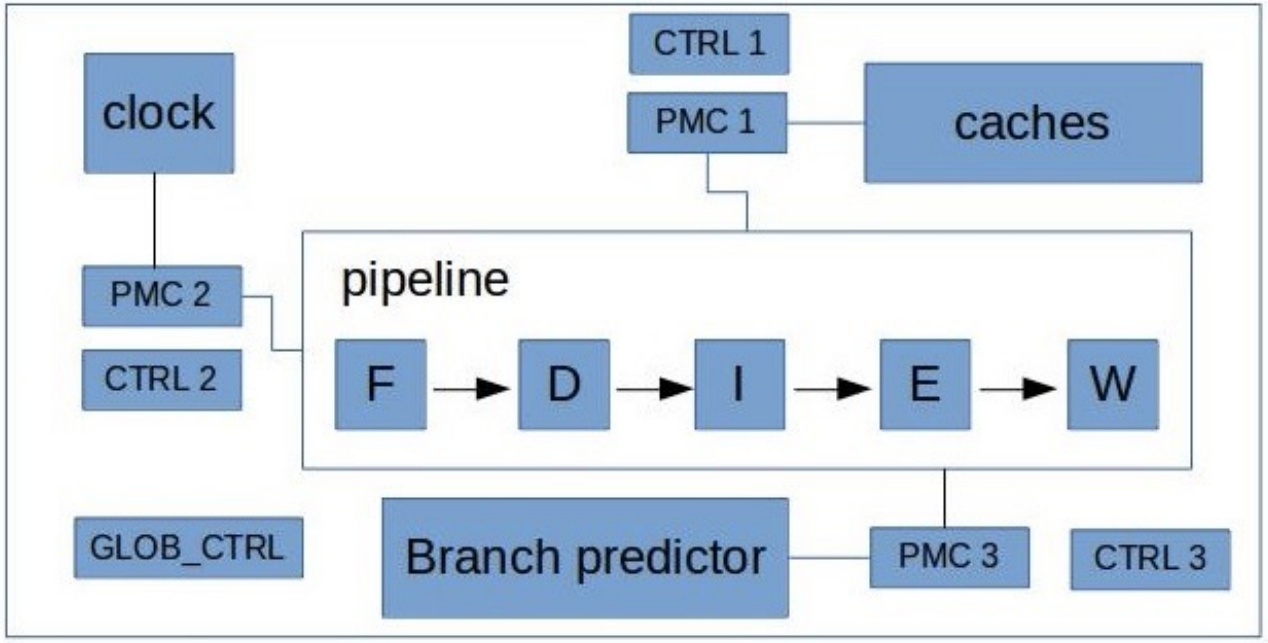
Hardware performance counters

Hardware performance counters

- A set of special-purpose registers built into modern microprocessors
- Store the counts of hardware-related activities/events
- Counters = the actual registers
- Events = actual hardware events
  - Events / counters >> 1 => reprogramming the counters !
- Performance Monitoring Units: hardware units to monitor performance
  - Core: what happens at core level
  - Uncore: outside the core

A “visualization”*

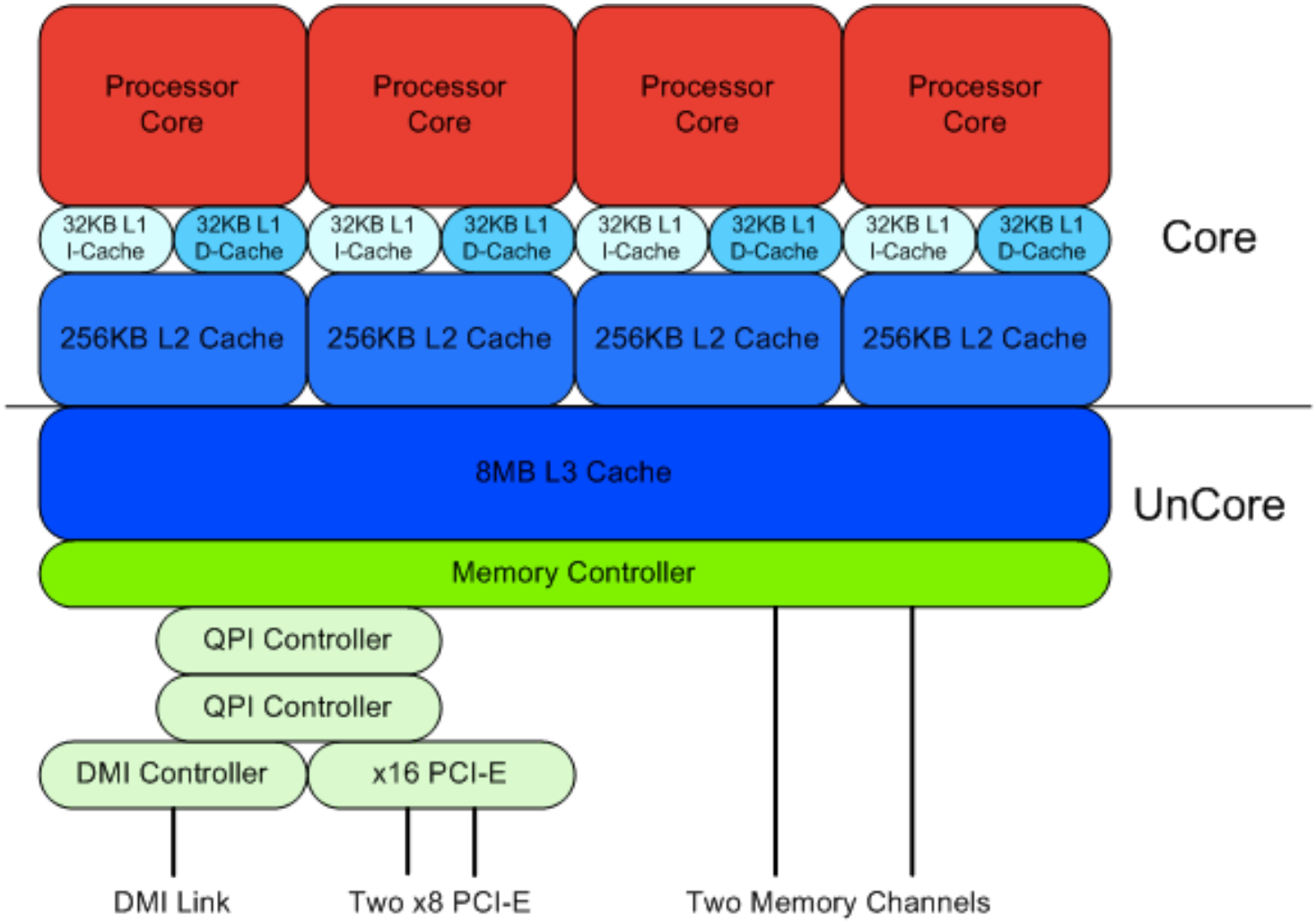
- Performance counters in the system (not all wires included)



- Fixed PMCs = dedicated counters
- Programmable PMCs = configured for different events

https://easyperf.net/blog/2018/06/01/PMU-counters-and-profiling-basics

Core vs. Uncore



Types of counters (examples)

Core-events

- instructions retired
- elapsed core clock ticks
- core frequency
- memory subsystem (L1, L2)

UnCore-events

- LLC
- Read/written bytes from/to memory controller(s)
- Data traffic transferred by the QPI links.

Predefined architectural events

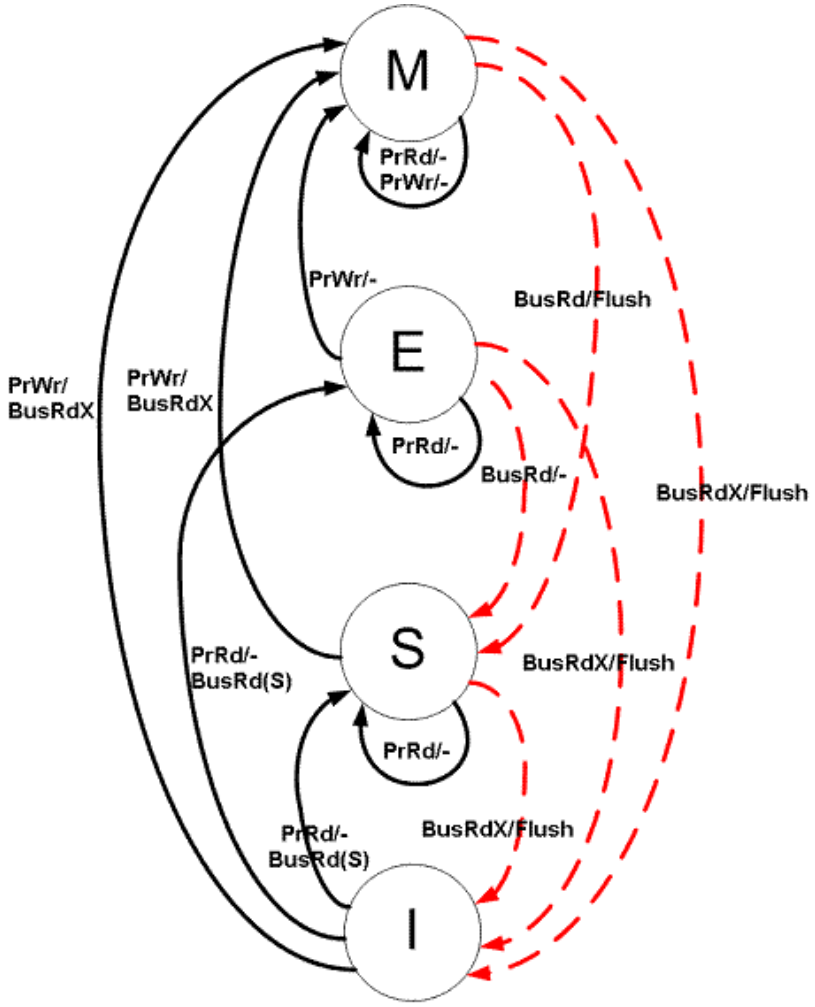
- Instructions retired (`INST_RETIRED.ANY`)
  - The number of instructions executed and completed, as needed by the flow
    - Useful instructions completed
- Unhalted clock cycles for the core (`CPU_CLK_UNHALTED.THREAD`)
  - The number of cycles when *the thread* is not in HALT state
    - **Note: variable frequency**
- Unhalted clock cycles for any core (`CPU_CLK_UNHALTED.THREAD_ANY`)
  - The number of cycles when there is *a thread* that is not in HALT state
    - **Note: variable frequency**
- Unhalted reference cycles (`CPU_CLK_UNHALTED.REF_TSC`)
  - The number of “reference cycles” when the core is not in HALT state
    - **Fixed/reference frequency**
  - *Can approximate elapsed time while the core was not in a halt state.*

Predefined architectural events

- LLC Reference
  - Calls to LLC
  - May include speculative execution noise
- LLC Misses
  - Calls to the LLC that have resulted in a miss
  - May include speculative execution noise
- Branch Instructions Retired
  - Branch instructions at retirement (that is, executed)
- Branch Misses Retired
  - Branch instructions at retirement after misprediction

Wait – how about LLC?

- LLC is shared across cores …
- How does that work?
  - MESI protocol*
    - M = Modified
    - E = Exclusive
    - S = Shared
    - I = Invalid
- Special counters
  - `UNC_CBO_CACHE_LOOKUP.ANY_MESI`
  - `UNC_CBO_CACHE_LOOKUP.READ_I`
  - …



https://en.wikipedia.org/wiki/MESI_protocol

Wait – how about NUMA?

- NUMA = non uniform memory access
  - Local = local socket
  - Remote = different socket

- Load requests are the same …
- … but serviced from LOCAL or REMOTE

- Counters:
  - `OFFCORE_RESPONSE_0_LOCAL_DRAM`
  - `OFFCORE_RESPONSE_1_REMOTE_DRAM`

Other events … many …

- Different per architecture
- Tables available in Intel manuals …
  - …and other sources
- Examples?

```
AVX_INSTS_LOADS, 0xC6, 0x1, PMC
AVX_INSTS_STORES, 0xC6, 0x2, PMC
AVX_INSTS_CALC, 0xC6, 0x4, PMC
AVX_INSTS_ALL, 0xC6, 0x7, PMC
QPI_RATE, 0x0, 0x0, QBOX0FIX0|QBOX1FIX0
SNOOPS_RSP_AFTER_DATA_LOCAL, 0xA, 0x1, BBOX
SNOOPS_RSP_AFTER_DATA_REMOTE, 0xA, 0x2, BBOX
…
```

More details …

- Description of counters for processors:

Intel Manual (vol 3B – part 2)

https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.html

- Description from PERFMON:

https://perfmon-events.intel.com/

https://download.01.org/perfmon/

Warnings : complexity

- High-complexity of hardware => many different types of counters
  - It is rarely the case that a single event tells a complete story
- Intel splits events in *architectural* and *non-architectural*
  - i.e., processor independent vs. processor dependent
- Different generations => different *non-architectural* counters
  - Different names
  - Different meanings
- Typically used to confirm/infirm hypotheses
  - A counter on its own won't tell you much if you don't have any expectation …

Warnings : clocks

- Processor clocks …
  - Can stop ticking for power-saving
  - Can change frequency on the fly
- Several clocks are available:
  - Base Clock – the nominal frequency of the processor
  - Maximum Clock – the highest possible frequency of the processor
  - Bus Clock (some systems) – fixed frequency
  - Core Crystal Clock – fixed frequency
- Processor clockticks are counted as …
  - Non-halted Clockticks – not halted/not in power saving
  - Non-sleep Clockticks – not sleep/not in power saving
  - Reference Clockicks – using a fixed frequency
- Time-Stamp Counter – counter

Tools & methods [1]

- Low-level assembly code
  - Might require extra rights for some operations
- An example (Intel forum*):

```
#define rdpmc(counter,low,high) \
  __asm__ __volatile__("rdpmc" \
  : "=a" (low), "=d" (high) \
  : "c" (counter))
```

  - c = counter code (from documentation)
  - a, d = result

- To use: repeatedly sample counter …

https://software.intel.com/en-us/forums/software-tuning-performance-optimization-platform-monitoring/topic/595214

Tools and methods [2]

- PAPI (http://icl.cs.utk.edu/papi/overview/index.html)
  - Portable interface across devices
  - Simple API to access most counters & operations
- An example:

```
int event[NUM_EV]={PAPI_TOT_INS, PAPI_TOT_CYC, PAPI_L1_DCM };
long long values[NUM_EV];

/* Start counting events */
PAPI_start_counters(event, NUM_EV);
//call function
PAPI_read_counters(values, NUM_EV);

printf("Total instructions: %lld\n", values[0]);
/* Stop counting events */
PAPI_stop_counters(values, NUM_EVENTS)
```

https://gist.github.com/dubik/1360496

Tools & methods [3]

- High-level tools
  - Intel VTune
  - LIKWID
    - Lots of tools to simplify collection of performance counter data
      - **`likwid-perfctr`**
      - **`likwid-powermeter`**
      - **`likwid-pin`**
      - **`likwid-topology`**
      - **`...`**
    - Lots of documentation
      - `https://github.com/RRZE-HPC/likwid/wiki/TutorialStart`
    - Installed on DAS5
      - Can install locally (on DAS5) too

https://developer.amd.com/amd-uprof/

Likwid-perfctr in more detail

- Different modes of operation, e.g.:
  - Non-intrusive (wrapper around application)
  - Per region (using markers *inside the code*)
- Can collect data using …
  - Specific groups of counters
    - Usually providing derived metrics
  - Specific events
  - Different cores & core pinning
- Reports …
  - On-screen data
  - .csv files

Wait – how about other CPUs?

- All have hardware events and performance counters
- Same basic counters, different names and/or different low-level configurations

- For AMD
  - AMD μProf*
  - LIKWID

https://developer.amd.com/amd-uprof/

What about GPGPUs?

Hardware performance counters

- Same principle as for CPUs
- Collected using nvprof
  - Details depend on arguments
  - More:

  https://docs.nvidia.com/cuda/profiler-users-guide/index.html#profiling-modes

- Supported by PAPI, too!

  https://developer.nvidia.com/papi-cuda-component

`nvprof` details [1]

- Summary mode of operation:
  - Provides basic performance data for the application

```
nvprof <exec>
```

```
Performance= 35.35 GFlop/s, Time= 3.708 msec,
Size= 131072000 Ops, WorkgroupSize= 1024
threads/block
```

`nvprof` details [2]

- GPU-Trace and API-Trace
  - Provides a list of the trace of events happening on the device
  - Performance numbers for data transfers
    - Start-time, duration, throughput
  - Performance data for kernels
    - Start-time, duration, Grid-size, block-size, registers, …

```
nvprof –print-gpu-trace <exec>
```

`nvprof` details [3]

- Event metrics/summary mode
  - Events: real events in hardware
    - E.g., warps_launched, local_load, …
  - Metrics: post-processing events
    - E.g., ipc

```
nvprof --events warps_launched,local_load --metrics ipc <exec>
```

  - To view all events/metrics:

```
nvprof --query-events
nvprof --query-metrics
```

Some GPU performance counters

counter	meaning
shared_replay_overhead	*average number of replays due to shared memory conflicts for each instruction executed*
shared_load|store	*number of executed shared load (store) instructions, increments per warp on a multiprocessor*
inst_replay_overhead	*average number of replays for each instruction executed*
l1_global_load_hit	*number of cache lines that hit in L1 for global memory load accesses*
l1_global_load_miss	*number of cache lines that miss in L1 for global memory load accesses*
gld_request	*number of executed global load instructions increments per warp on a multiprocessor*
gst_request	*similar to* gld_request *for store instructions*
global_store_transaction	*number of global store transactions increments per transaction which ca be 32,64,96 or 128 bytes*
gld_requested_throughput	*requested global memory load throughput*
achieved_occupancy	*ratio of average active warps per active cycle to the maximum number of warps per SM*
l2_read_throughput	*memory read throughput at L2 cache*
l2_write_transactions	*memory write transactions at L2 cache*
ipc	*number of instructions executed per cycle*
issue_slot_utilization	*percentage of issue slots that issued at least one instruction, averaged across all cycles*
warp_execution_efficiency	*ratio of the average active threads per warp to the maximum number of threads per warp supported by the multiprocessor*

Counters & bottleneck analysis

Detect performance bottlenecks

- A bottleneck indicates the main limitation in the performance of the application.
- In most cases, an application has one single bottleneck (visible) at any time
- Bottleneck analysis => indicates problem + “theoretical” solution
  - Practical solutions are problem-driven

Integrate bottleneck analysis in P.E.

- Focus on resource utilization

- Two main resources in your system

1. Instruction execution
2. Data transfer bandwidth

- Main questions
  - What is the limiting resource in your application?
  - Do you fully utilize this resource?

Performance patterns & “signatures”

- Performance pattern
  - A specific performance behavior/problem
  - Assess the quality of code and identify relevant bottlenecks to enable a structured approach to performance optimizations
- Pattern-based P.E.
- *For each performance pattern, identify a specific behavior*
  - observed performance behavior at runtime => pattern
  - performance counters => validation of pattern
- Apply “known solutions” for performance improvement
  - check the absence of the previous pattern …

Performance patterns*

- Load Imbalance
- Bandwidth saturation
- Strided or erratic data access
- Bad Instruction Mix
- Limited instruction throughput
- Microarchitectural anomalies
- Synchronization overhead
- False cache line sharing
- Bad page placement on ccNUMA

Jan Treibig et al: "Performance Patterns and Hardware Metrics on Modern Multicore Processors: Best Practices for Performance Engineering"

*Also called anti-patterns, pathologies, problems …

Load imbalance

- Issue:
  - The workload is not equally distributed
  - Several units stall waiting for one unit to complete
- Performance behavior:
  - saturating speed-up (sooner than expected)
- Performance counters:
  - Different count of instructions retired or floating point operations among cores (FLOPS_DP, FLOPS_SP)
- Fix:
  - Reorganize work to improve load balancing

Bandwidth saturation

- Issue:
  - Bandwidth of a shared data path is exhausted
- Performance behavior:
  - Saturating speedup across cores sharing a memory interface.
- Memory bandwidth comparable to peak bandwidth
  - Measure peak with microbenchmark (MEM)
  - Can be applied for L3 or Mem
- Fix:
  - Reduce the number of load/stores

Strided/erratic data access

- Issue:
  - low data transfer efficiency (between caches and to/from memory).
  - inappropriate data structures or badly ordered loop nests
- Performance behavior:
  - Large discrepancy between simple bandwidth-based model and actual performance
- Performance counters:
  - Low bandwidth utilization despite LD/ST domination
  - Low cache hit ratios, frequent evicts/replacements (CACHE, DATA, MEM)
- Fix:
  - Improve locality and strides

Limited instruction throughput

- Issue
  - Fewer than expected instructions per cycle
- Performance behavior
  - Large discrepancy between actual performance and simple predictions based on max Flop/s or LD/ST throughput
- Performance counters
  - Low CPI near theoretical limit if instruction throughput is the problem
  - Static code analysis predicting large pressure on single execution port
  - High CPI due to bad pipelining
  - (FLOPS_DP, FLOPS_SP, DATA)
- Fix:
  - ?

Bad instruction mix

- Issue:
  - Not enough parallelism, no vectorization, expensive operations
  - Inefficient compiler code
- Performance behavior:
  - Performance insensitive to problem size fitting into different cache levels
- Performance counters:
  - Large ratio of instructions retired to FP instructions if the useful work is FP
  - Many cycles per instruction (CPI) if the problem is large-latency arithmetic
  - Scalar instructions dominating in data-parallel loops
  - (FLOPS_DP, FLOPS_SP)
- Fix:
  - Improve instruction mix (different operations, reordering, loop unrolling)

Synchronization overhead

- Issue:
  - barriers at the end of parallel loops
  - locks protecting shared resources
- Performance behavior
  - Speedup going down as more cores are added
  - No speedup with small problem sizes
  - Cores busy but low FP performance
- Performance counters
  - Large non-FP instruction count (growing with number of cores used)
  - Low CPI
  - FLOPS_DP, FLOPS_DP
- Fix:
  - Remove unnecessary synchronization (especially the implicit ones!)

False sharing

- Issue:
  - Different threads accessing a cache line, at least one of them modifying it
- Performance behavior
  - Very low speedup or slowdown even with small core counts
- Performance counters
  - Frequent (remote) evicts (CACHE)
- Fix:
  - Revisit the working set per thread
  - Data replication

Bad ccNUMA page placement

- Issue
  - nonlocal data access
  - bandwidth contention
- Performance behavior
  - Bad/no scaling across locality domains
- Performance counters
  - Unbalanced bandwidth on memory interfaces
  - High remote traffic (MEM)
- Fix:
  - Reorganize memory accesses
  - (Attempt different page placement)

What about GPGPUs?

Same idea, different patterns …

- No systematic work available*
  - Can extract from various talks … (see Canvas)

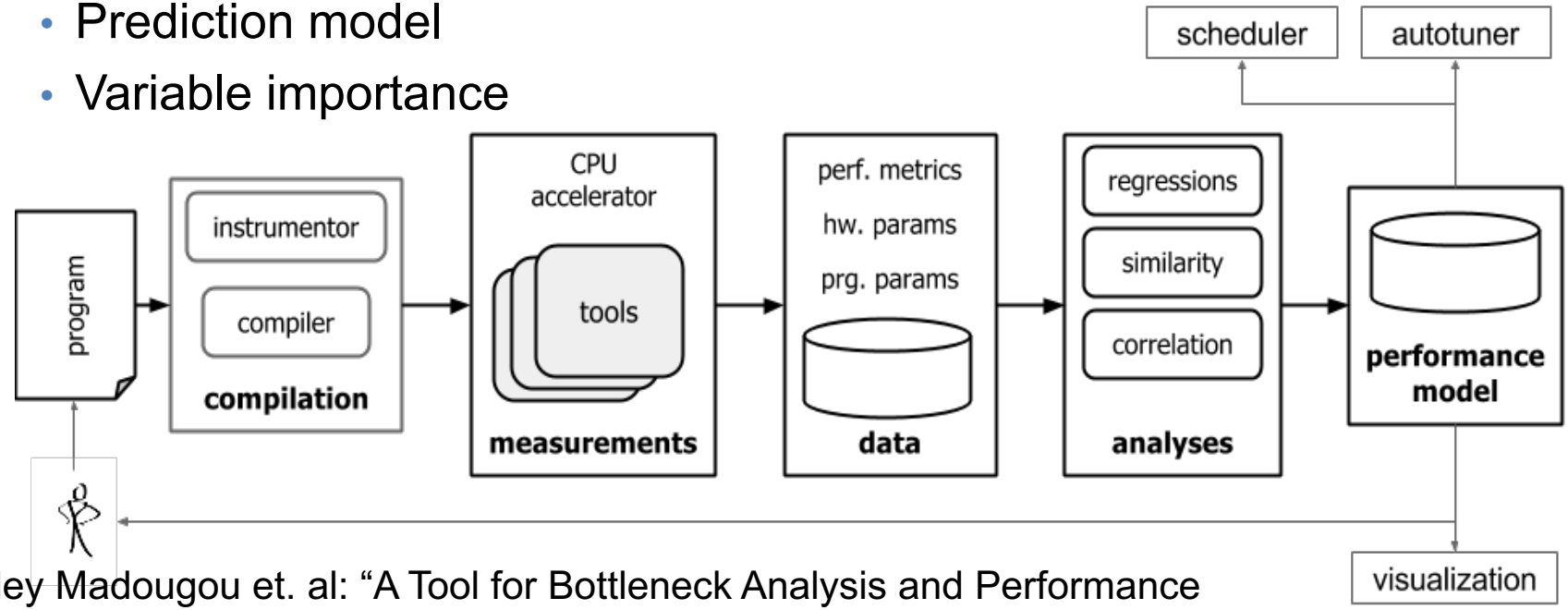
- Examples of problems
  - Thread load-imbalance
  - Thread divergence
  - High branching
  - Non-cached memory accesses
  - Bandwidth saturation/limitation
  - SM load imbalance
  - Insufficient workload
  - Synchronization issues
  - …

https://on-demand.gputechconf.com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf

One step further

The BlackForest framework*

- Automate the process of statistical modeling
- Target variable: execution time
- Predictor variables: performance counters
- Outcome:
  - Prediction model
  - Variable importance



*Souley Madougou et. al: "A Tool for Bottleneck Analysis and Performance Prediction for GPU-accelerated Applications" – ASHES 2016

To collect counters

- Three *CPU* methods
  - Assembly code
  - PAPI
  - External tools – like perf, likwid-perfctr, …
- Same* *GPU* methods
  - nvprof / nsight
  - *Supported by PAPI, too!
    https://developer.nvidia.com/papi-cuda-component
  - *Supported from Likwid 5.0
- Similar principles everywhere:
  - Run program
  - Read/Sample counters of interest
  - Use to support/disprove hypothesis

**Claim not tested in practice.*