Top-down Microarchitecture Analysis through Linux perf and toplev tools

Ahmad Yasin

CPU Architect, Intel Corporation

Haifa::C++ Meetup

March 14th, 2018

Outline

- Motivation & Background
- Top-down Microarchitecture Analysis (TMA)
- Demo 1: Linux's **`perf stat`**
- TMA Hierarchy and Locating Issues
- Demo 2: **`pmu-tools/toplev`**
- Summary & reference

Skylake processor & memory subsystem

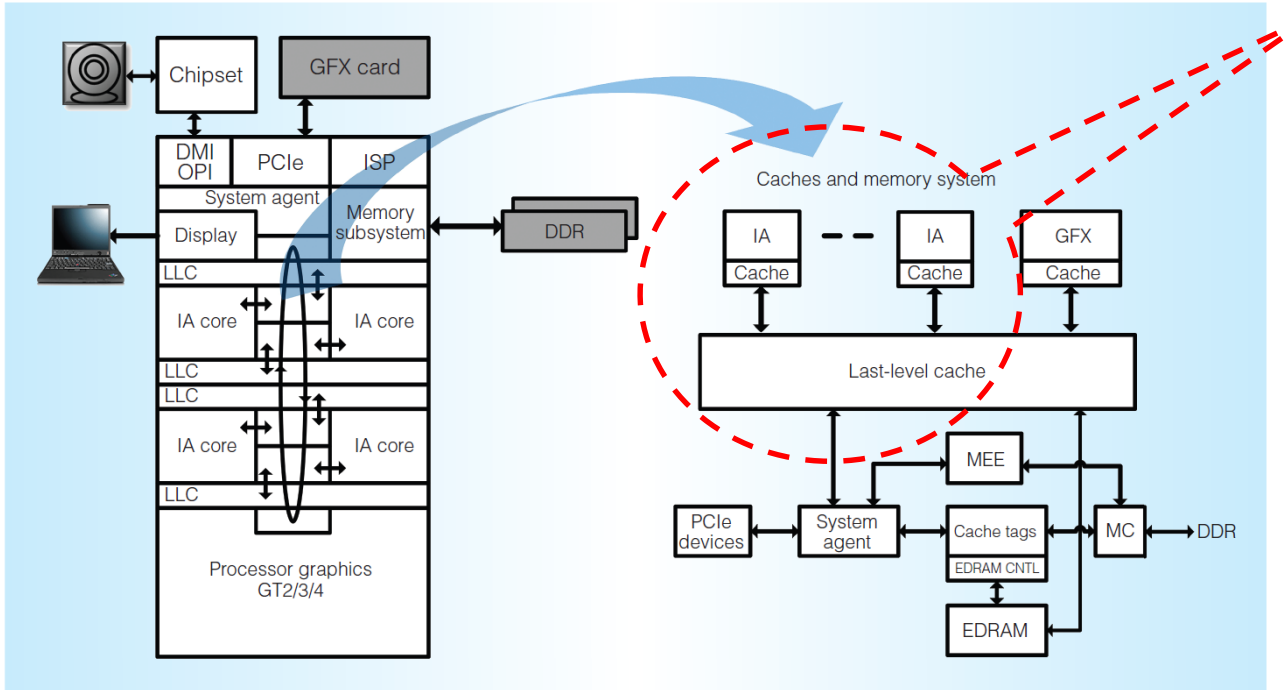
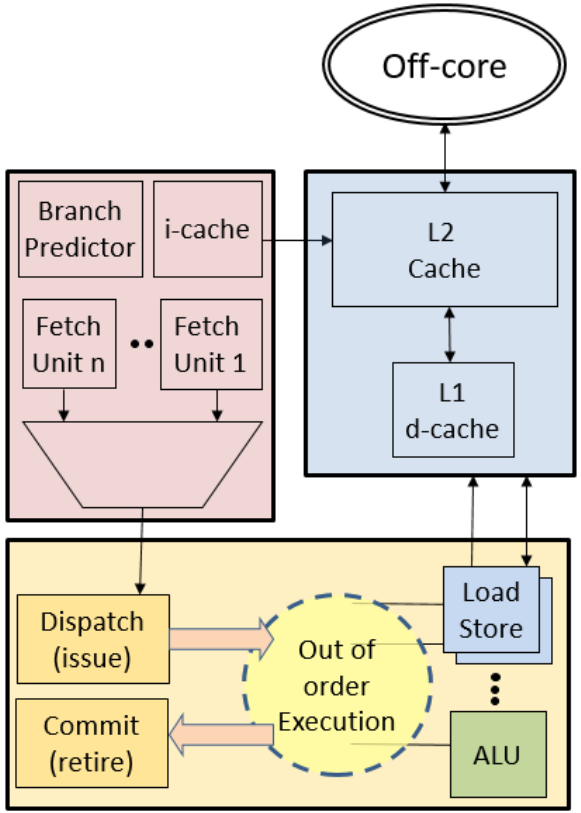


Figure 1. Skylake block diagram with zoom into cache and memory subsystem. The memory subsystem is shown with the eDRAM-based memory side cache.

Source: Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake. Jack Doweck, Wen-Fu Kao, Allen Kuan-yu Lu, Julius Mandelblat, Anirudha Rahatekar, Lihu Rappoport, Efraim Rotem, Ahmad Yasin, Adi Yoaz. IEEE Micro, Volume 37, Issue 2, 2017. [IEEE]

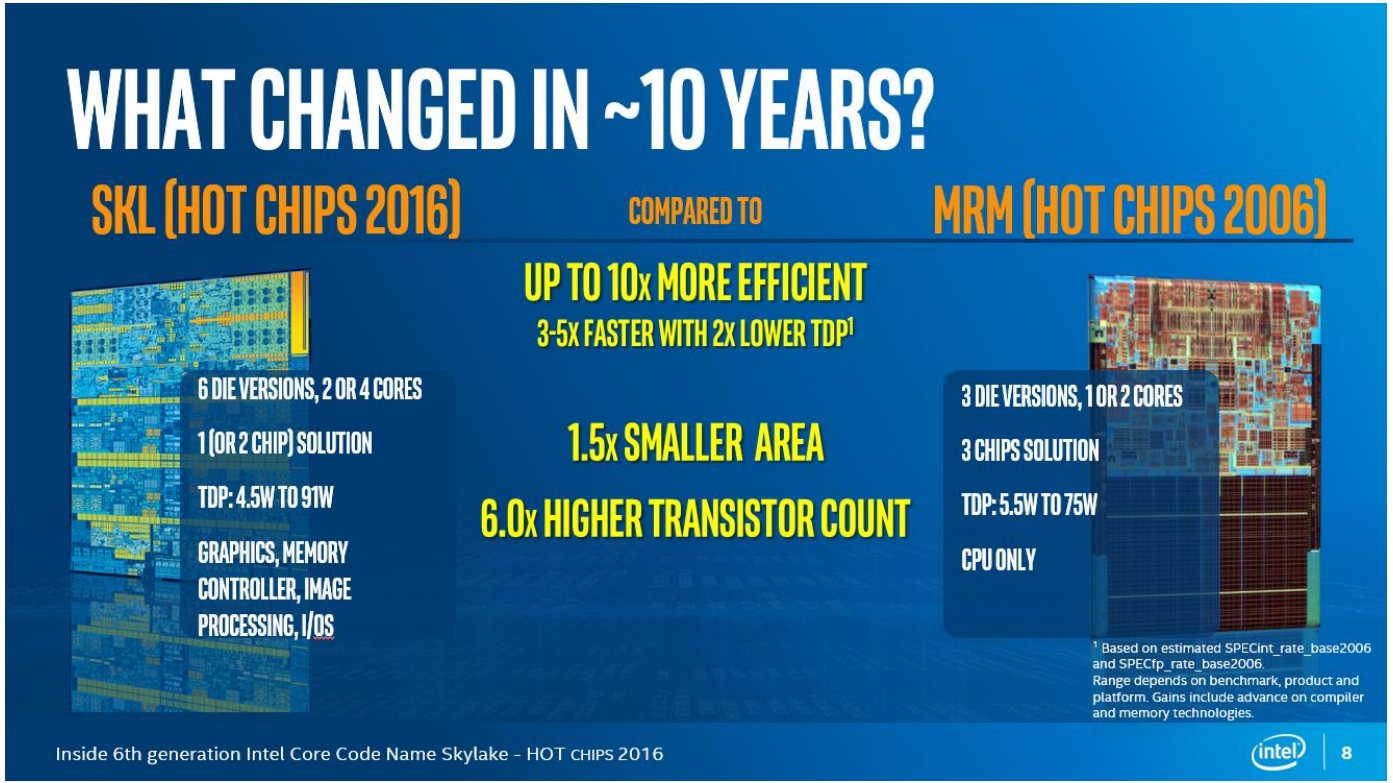
Modern Out-of-Order cores

- Pipelined
- Superscalar
- OOO Execution
- Speculation
- Multiple Caches
- Memory Pre-fetching and Disambiguation
- Vector Operations



Source: Fine-Grain Power Breakdown of Modern Out-of-Order Cores and Its Implications on Skylake-Based Systems. Jawad Haj-Yihia, Ahmad Yasin, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO) Journal, Volume 13 Issue 4, December 2016

Wait… That was over simplified



Source: Inside 6th generation Intel Core code named Skylake: New Microarchitecture and Power Management, Jack Doweck, Ittai Anati, David Blythe, Hong Jiang, Wen-fu Kao, Julius Mandelblat, Lihu Rappoport, Efraim Rotem, Ahmad Yasin. Hot Chips 2016.

From 100s of performance counters (events)

Performance Monitoring Events for the Sixth Generation Intel Core Processors Based on the Skylake Microarchitecture - V36

INST_RETIRED.ANY
CPU_CLK_UNHALTED.THREAD
CPU_CLK_UNHALTED.THREAD_ANY
CPU_CLK_UNHALTED.REF_TSC
LD_BLOCKS.STORE_FORWARD
LD_BLOCKS.NO_SR
LD_BLOCKS_PARTIAL.ADDRESS_ALIAS
DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK
DTLB_LOAD_MISSES.WALK_COMPLETED_4K
DTLB_LOAD_MISSES.WALK_COMPLETED_2M_4M
DTLB_LOAD_MISSES.WALK_COMPLETED_1G
DTLB_LOAD_MISSES.WALK_COMPLETED
DTLB_LOAD_MISSES.WALK_PENDING
DTLB_LOAD_MISSES.WALK_ACTIVE
DTLB_LOAD_MISSES.STLB_HIT
INT_MISC.RECOVERY_CYCLES
INT_MISC.RECOVERY_CYCLES_ANY
INT_MISC.CLEAR_RESTEER_CYCLES
UOPS_ISSUED.ANY
UOPS_ISSUED.STALL_CYCLES
UOPS_ISSUED.VECTOR_WIDTH_MISMATCH
UOPS_ISSUED.SLOW_LEA
ARITH.DIVIDER_ACTIVE
L2_RQSTS.DEMAND_DATA_RD_MISS
L2_RQSTS.RFO_MISS
L2_RQSTS.CODE_RD_MISS
L2_RQSTS.ALL_DEMAND_MISS
L2_RQSTS.PF_MISS
L2_RQSTS.MISS
L2_RQSTS.DEMAND_DATA_RD_HIT
L2_RQSTS.RFO_HIT
L2_RQSTS.CODE_RD_HIT
L2_RQSTS.PF_HIT
L2_RQSTS.ALL_DEMAND_DATA_RD
L2_RQSTS.ALL_RFO
L2_RQSTS.ALL_CODE_RD
L2_RQSTS.ALL_DEMAND_REFERENCES
L2_RQSTS.ALL_PF
L2_RQSTS.REFERENCES
LONGEST_LAT_CACHE.MISS
LONGEST_LAT_CACHE.REFERENCE
SW_PREFETCH_ACCESS.NTA
SW_PREFETCH_ACCESS.T0
SW_PREFETCH_ACCESS.T1_T2
SW_PREFETCH_ACCESS.PREFETCHW
CPU_CLK_UNHALTED.THREAD_P
CPU_CLK_UNHALTED.THREAD_P_ANY
CPU_CLK_UNHALTED.RING0_TRANS
CPU_CLK_THREAD_UNHALTED.REF_XCLK
CPU_CLK_THREAD_UNHALTED.REF_XCLK_ANY
CPU_CLK_UNHALTED.REF_XCLK
CPU_CLK_UNHALTED.REF_XCLK_ANY
CPU_CLK_THREAD_UNHALTED.ONE_THREAD_ACTIVE
CPU_CLK_UNHALTED.ONE_THREAD_ACTIVE
L1D_PEND_MISS.PENDING
L1D_PEND_MISS.PENDING_CYCLES
L1D_PEND_MISS.PENDING_CYCLES_ANY
L1D_PEND_MISS.FB_FULL
DTLB_STORE_MISSES.MISS_CAUSES_A_WALK
DTLB_STORE_MISSES.WALK_COMPLETED_4K
DTLB_STORE_MISSES.WALK_COMPLETED_2M_4M
DTLB_STORE_MISSES.WALK_COMPLETED_1G
DTLB_STORE_MISSES.WALK_COMPLETED
DTLB_STORE_MISSES.WALK_PENDING
DTLB_STORE_MISSES.WALK_ACTIVE
DTLB_STORE_MISSES.STLB_HIT
LOAD_HIT_PRE.SW_PF
EPT.WALK_PENDING
L1D.REPLACEMENT
TX_MEM.ABORT_CONFLICT
TX_MEM.ABORT_CAPACITY
TX_MEM.ABORT_HLE_STORE_TO_ELIDED_LOCK
TX_MEM.ABORT_HLE_ELISION_BUFFER_NOT_EMPTY
TX_MEM.ABORT_HLE_ELISION_BUFFER_MISMATCH
TX_MEM.ABORT_HLE_ELISION_BUFFER_UNSUPPORTED_ALIGNMENT
TX_MEM.HLE_ELISION_BUFFER_FULL
TX_EXEC.MISC1
TX_EXEC.MISC2
TX_EXEC.MISC3
TX_EXEC.MISC4
TX_EXEC.MISC5
RS_EVENTS.EMPTY_CYCLES
RS_EVENTS.EMPTY_END
OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD
OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_DATA_RD
OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD_GE_6
OFFCORE_REQUESTS_OUTSTANDING.DEMAND_CODE_RD
OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_CODE_RD
OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO
OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DEMAND_RFO
OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD
OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_DATA_RD
OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD
OFFCORE_REQUESTS_OUTSTANDING.CYCLES_WITH_L3_MISS_DEMAND_DATA_RD
OFFCORE_REQUESTS_OUTSTANDING.L3_MISS_DEMAND_DATA_RD_GE_6
IDQ.MITE_UOPS
IDQ.MITE_CYCLES
IDQ.DSB_UOPS
IDQ.DSB_CYCLES
IDQ.MS_DSB_CYCLES
IDQ.ALL_DSB_CYCLES_4_UOPS
IDQ.ALL_DSB_CYCLES_ANY_UOPS
IDQ.MS_MITE_UOPS
IDQ.ALL_MITE_CYCLES_4_UOPS
IDQ.ALL_MITE_CYCLES_ANY_UOPS
IDQ.MS_CYCLES
IDQ.MS_SWITCHES
IDQ.MS_UOPS
ICACHE_16B.IFDATA_STALL
ICACHE_64B.IFTAG_HIT
ICACHE_64B.IFTAG_MISS
ICACHE_64B.IFTAG_STALL
ITLB_MISSES.MISS_CAUSES_A_WALK
ITLB_MISSES.WALK_COMPLETED_4K
ITLB_MISSES.WALK_COMPLETED_2M_4M
ITLB_MISSES.WALK_COMPLETED_1G
ITLB_MISSES.WALK_COMPLETED
ITLB_MISSES.WALK_PENDING
ITLB_MISSES.WALK_ACTIVE
ITLB_MISSES.STLB_HIT
ILD_STALL.LCP
IDQ_UOPS_NOT_DELIVERED.CORE
IDQ_UOPS_NOT_DELIVERED.CYCLES_0_UOPS_DELIV.CORE
IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_1_UOP_DELIV.CORE
IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_2_UOP_DELIV.CORE
IDQ_UOPS_NOT_DELIVERED.CYCLES_LE_3_UOP_DELIV.CORE
IDQ_UOPS_NOT_DELIVERED.CYCLES_FE_WAS_OK
UOPS_DISPATCHED_PORT.PORT_0
UOPS_DISPATCHED_PORT.PORT_1
UOPS_DISPATCHED_PORT.PORT_2
UOPS_DISPATCHED_PORT.PORT_3
UOPS_DISPATCHED_PORT.PORT_4
UOPS_DISPATCHED_PORT.PORT_5
UOPS_DISPATCHED_PORT.PORT_6
UOPS_DISPATCHED_PORT.PORT_7
RESOURCE_STALLS.ANY
RESOURCE_STALLS.SB
CYCLE_ACTIVITY.CYCLES_L2_MISS
CYCLE_ACTIVITY.CYCLES_L3_MISS
CYCLE_ACTIVITY.STALLS_TOTAL
CYCLE_ACTIVITY.STALLS_L2_MISS
CYCLE_ACTIVITY.STALLS_L3_MISS
CYCLE_ACTIVITY.CYCLES_L1D_MISS
CYCLE_ACTIVITY.STALLS_L1D_MISS
CYCLE_ACTIVITY.CYCLES_MEM_ANY
CYCLE_ACTIVITY.STALLS_MEM_ANY
EXE_ACTIVITY.EXE_BOUND_0_PORTS
EXE_ACTIVITY.1_PORTS_UTIL
EXE_ACTIVITY.2_PORTS_UTIL
EXE_ACTIVITY.3_PORTS_UTIL
EXE_ACTIVITY.4_PORTS_UTIL
EXE_ACTIVITY.BOUND_ON_STORES
LSD.UOPS
LSD.CYCLES_ACTIVE
LSD.CYCLES_4_UOPS
DSB2MITE_SWITCHES.PENALTY_CYCLES
ITLB.ITLB_FLUSH
OFFCORE_REQUESTS.DEMAND_DATA_RD
OFFCORE_REQUESTS.DEMAND_CODE_RD
OFFCORE_REQUESTS.DEMAND_RFO
OFFCORE_REQUESTS.ALL_DATA_RD
OFFCORE_REQUESTS.L3_MISS_DEMAND_DATA_RD
OFFCORE_REQUESTS.ALL_REQUESTS
UOPS_EXECUTED.THREAD
UOPS_EXECUTED.STALL_CYCLES
UOPS_EXECUTED.CYCLES_GE_1_UOP_EXEC
UOPS_EXECUTED.CYCLES_GE_2_UOPS_EXEC
UOPS_EXECUTED.CYCLES_GE_3_UOPS_EXEC
UOPS_EXECUTED.CYCLES_GE_4_UOPS_EXEC
UOPS_EXECUTED.CORE
UOPS_EXECUTED.CORE_CYCLES_GE_1
UOPS_EXECUTED.CORE_CYCLES_GE_2
UOPS_EXECUTED.CORE_CYCLES_GE_3
UOPS_EXECUTED.CORE_CYCLES_GE_4
UOPS_EXECUTED.CORE_CYCLES_NONE
UOPS_EXECUTED.X87
OFFCORE_REQUESTS_BUFFER.SQ_FULL
OFFCORE_RESPONSE
TLB_FLUSH.DTLB_THREAD
TLB_FLUSH.STLB_ANY
INST_RETIRED.ANY_P
INST_RETIRED.PREC_DIST
INST_RETIRED.TOTAL_CYCLES_PS
OTHER_ASSISTS.ANY
UOPS_RETIRED.RETIRE_SLOTS
UOPS_RETIRED.STALL_CYCLES
UOPS_RETIRED.TOTAL_CYCLES
MACHINE_CLEARS.COUNT
MACHINE_CLEARS.MEMORY_ORDERING
MACHINE_CLEARS.SMC
BR_INST_RETIRED.ALL_BRANCHES
BR_INST_RETIRED.CONDITIONAL
BR_INST_RETIRED.NEAR_CALL
BR_INST_RETIRED.ALL_BRANCHES_PEBS
BR_INST_RETIRED.NEAR_RETURN
BR_INST_RETIRED.NOT_TAKEN
BR_INST_RETIRED.NEAR_TAKEN
BR_INST_RETIRED.FAR_BRANCH
BR_MISP_RETIRED.ALL_BRANCHES
BR_MISP_RETIRED.CONDITIONAL
BR_MISP_RETIRED.NEAR_CALL
BR_MISP_RETIRED.ALL_BRANCHES_PEBS
BR_MISP_RETIRED.NEAR_TAKEN
FRONTEND_RETIRED.DSB_MISS
FRONTEND_RETIRED.L1I_MISS
FRONTEND_RETIRED.L2_MISS
FRONTEND_RETIRED.ITLB_MISS
FRONTEND_RETIRED.STLB_MISS
FRONTEND_RETIRED.LATENCY_GE_2
FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_2
FRONTEND_RETIRED.LATENCY_GE_4
FRONTEND_RETIRED.LATENCY_GE_8
FRONTEND_RETIRED.LATENCY_GE_16
FRONTEND_RETIRED.LATENCY_GE_32
FRONTEND_RETIRED.LATENCY_GE_64
FRONTEND_RETIRED.LATENCY_GE_128
FRONTEND_RETIRED.LATENCY_GE_256
FRONTEND_RETIRED.LATENCY_GE_512
FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_1
FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_3
FP_ARITH_INST_RETIRED.SCALAR_DOUBLE
FP_ARITH_INST_RETIRED.SCALAR_SINGLE
FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE
FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE
FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE
FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE
HLE_RETIRED.START
HLE_RETIRED.COMMIT
HLE_RETIRED.ABORTED
HLE_RETIRED.ABORTED_MEM
HLE_RETIRED.ABORTED_TIMER
HLE_RETIRED.ABORTED_UNFRIENDLY
HLE_RETIRED.ABORTED_MEMTYPE
HLE_RETIRED.ABORTED_EVENTS
RTM_RETIRED.START
RTM_RETIRED.COMMIT
RTM_RETIRED.ABORTED
RTM_RETIRED.ABORTED_MEM
RTM_RETIRED.ABORTED_TIMER
RTM_RETIRED.ABORTED_UNFRIENDLY
RTM_RETIRED.ABORTED_MEMTYPE
RTM_RETIRED.ABORTED_EVENTS
FP_ASSIST.ANY
HW_INTERRUPTS.RECEIVED
ROB_MISC_EVENTS.LBR_INSERTS
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_4
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_8
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_16
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_32
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_64
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_128
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_256
MEM_TRANS_RETIRED.LOAD_LATENCY_GT_512
MEM_INST_RETIRED.STLB_MISS_LOADS
MEM_INST_RETIRED.STLB_MISS_STORES
MEM_INST_RETIRED.LOCK_LOADS
MEM_INST_RETIRED.SPLIT_LOADS
MEM_INST_RETIRED.SPLIT_STORES
MEM_INST_RETIRED.ALL_LOADS
MEM_INST_RETIRED.ALL_STORES
MEM_LOAD_RETIRED.L1_HIT
MEM_LOAD_RETIRED.L2_HIT
MEM_LOAD_RETIRED.L3_HIT
MEM_LOAD_RETIRED.L1_MISS
MEM_LOAD_RETIRED.L2_MISS
MEM_LOAD_RETIRED.L3_MISS
MEM_LOAD_RETIRED.FB_HIT
MEM_LOAD_L3_HIT_RETIRED.XSNP_MISS
MEM_LOAD_L3_HIT_RETIRED.XSNP_HIT
MEM_LOAD_L3_HIT_RETIRED.XSNP_HITM
MEM_LOAD_L3_HIT_RETIRED.XSNP_NONE
MEM_LOAD_MISC_RETIRED.UC
BACLEARS.ANY
L2_TRANS.L2_WB
L2_LINES_IN.ALL
L2_LINES_OUT.SILENT
L2_LINES_OUT.NON_SILENT
L2_LINES_OUT.USELESS_PREF
L2_LINES_OUT.USELESS_HWPF
SQ_MISC.SPLIT_LOCK

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

To: Abstracted Metrics (~few dozens)

Abstracted Metrics wrap Generational u-arch and PMU differences which helps to standarize Performance Analysis

	Key	Level1	Level2	Level3	Level4
4	FE	Frontend_Bound			
5	FE		Frontend_Latency		
9	FE		Frontend_Bandwidth		
10	BAD	Bad_Speculation			
11	BAD		Branch_Mispredicts		
12	BAD		Machine_Clears		
13	BE	Backend_Bound			
14	BE/Mem		Memory_Bound		
15	BE/Mem			L1_Bound	
17	BE/Mem			L2_Bound	
18	BE/Mem			L3_Bound	
19	BE/Mem			MEM_Bound	
20	BE/Mem				MEM_Bandwidth
21	BE/Mem				MEM_Latency
22	BE/Mem			Store_Bound	
23	BE/Core		Core_Bound		
24	BE/Core			Divider	
25	BE/Core			Ports_Utilization	
26	RET	Retiring			
27	RET		Base		
28	RET			FP_Arith	
29	RET				FP_Scalar
30	RET				FP_Vector

Source: https://download.01.org/perfmon/TMA_Metrics.xlsx

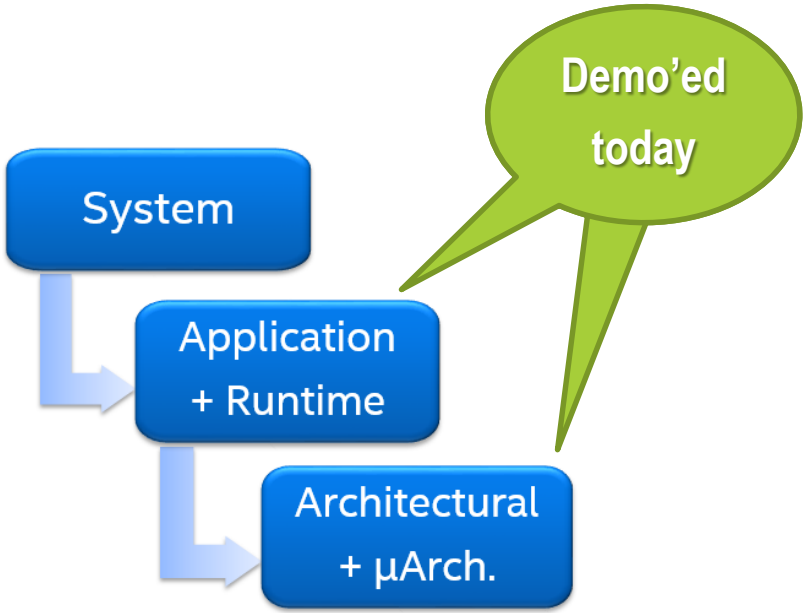
Performance Analysis

Top-down Microarchitecture Analysis (TMA)

What is Performance Analysis?

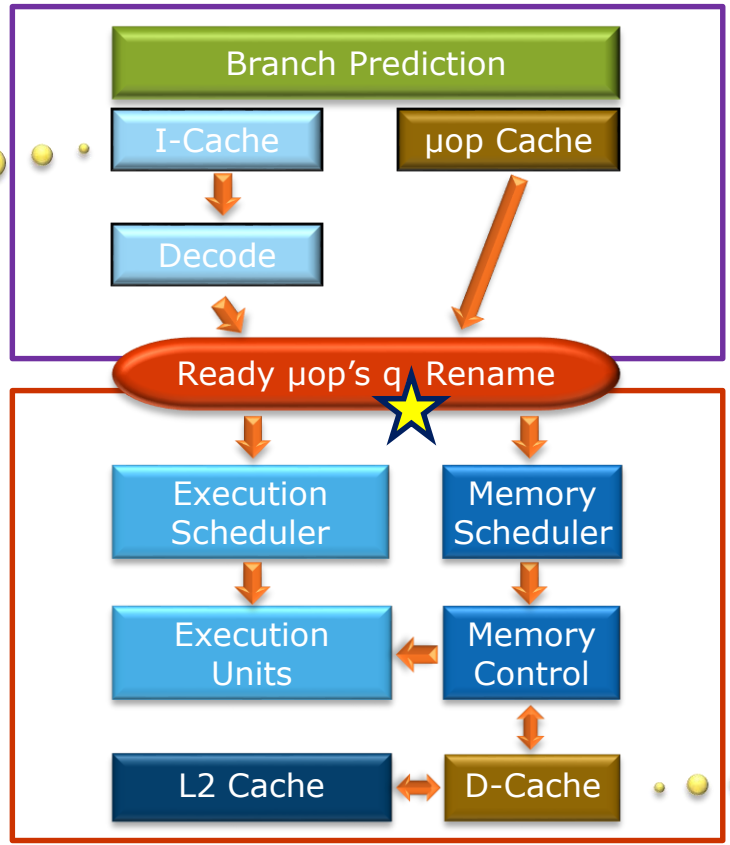
- A definition
  - "A performance analysis methodology is a procedure that you can follow to analyze system or application performance. These generally provide a starting point and then guidance to root cause, or causes. Different methodologies are suited for solving different classes of issues, and you may try more than one before accomplishing your goal.
  - Analysis without a methodology can become a fishing expedition, where metrics are examined ad hoc, until the issue is found – if it is at all."

Source: Brendan D. Gregg,
http://www.brendangregg.com/methodology.html



Skylake Core

+ an instruction cache miss in next function g()



Front-end of processor pipeline

Back-end of processor pipeline

A data cache miss in current function, say f()

TMA is designed to help developer to focus on areas that matter

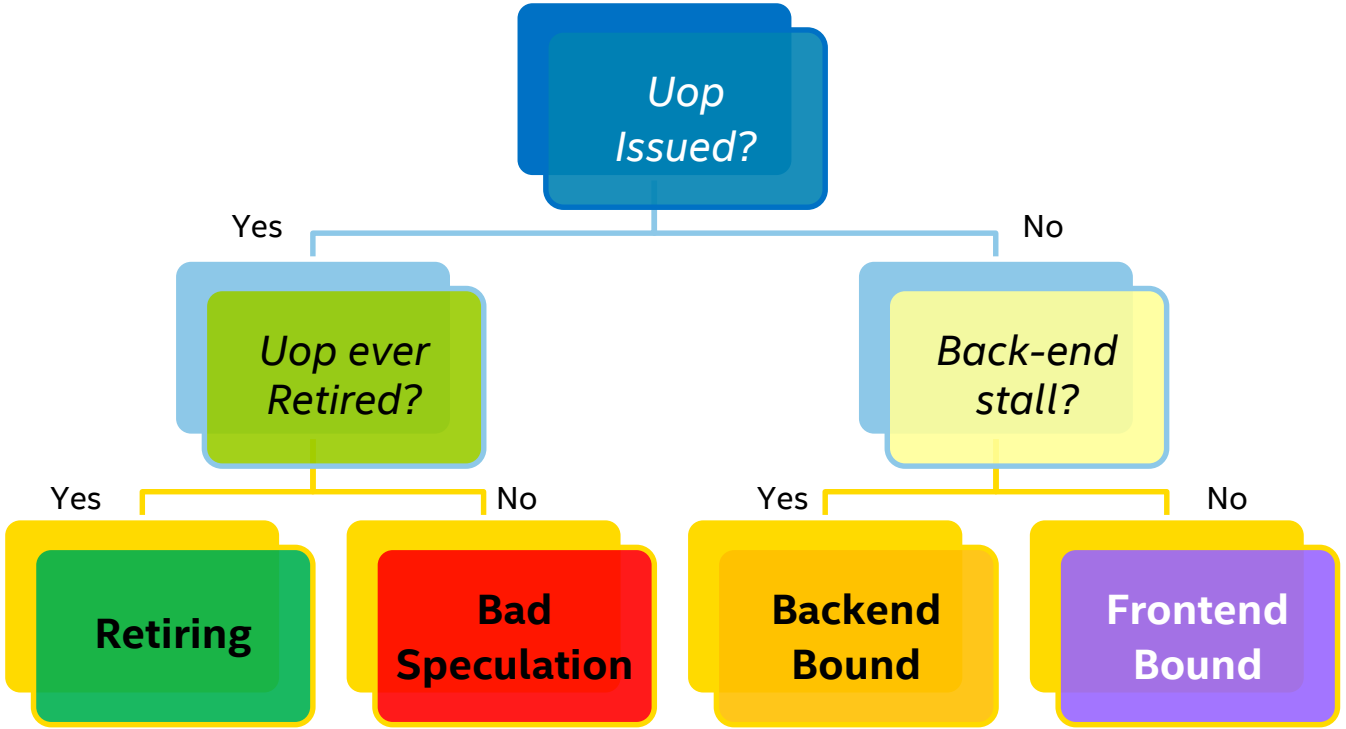
Challenges

- Naïve approach (from in-order cores land)

$$Stall\_Cycles = \Sigma\ Penalty_i * MissEvent_i$$

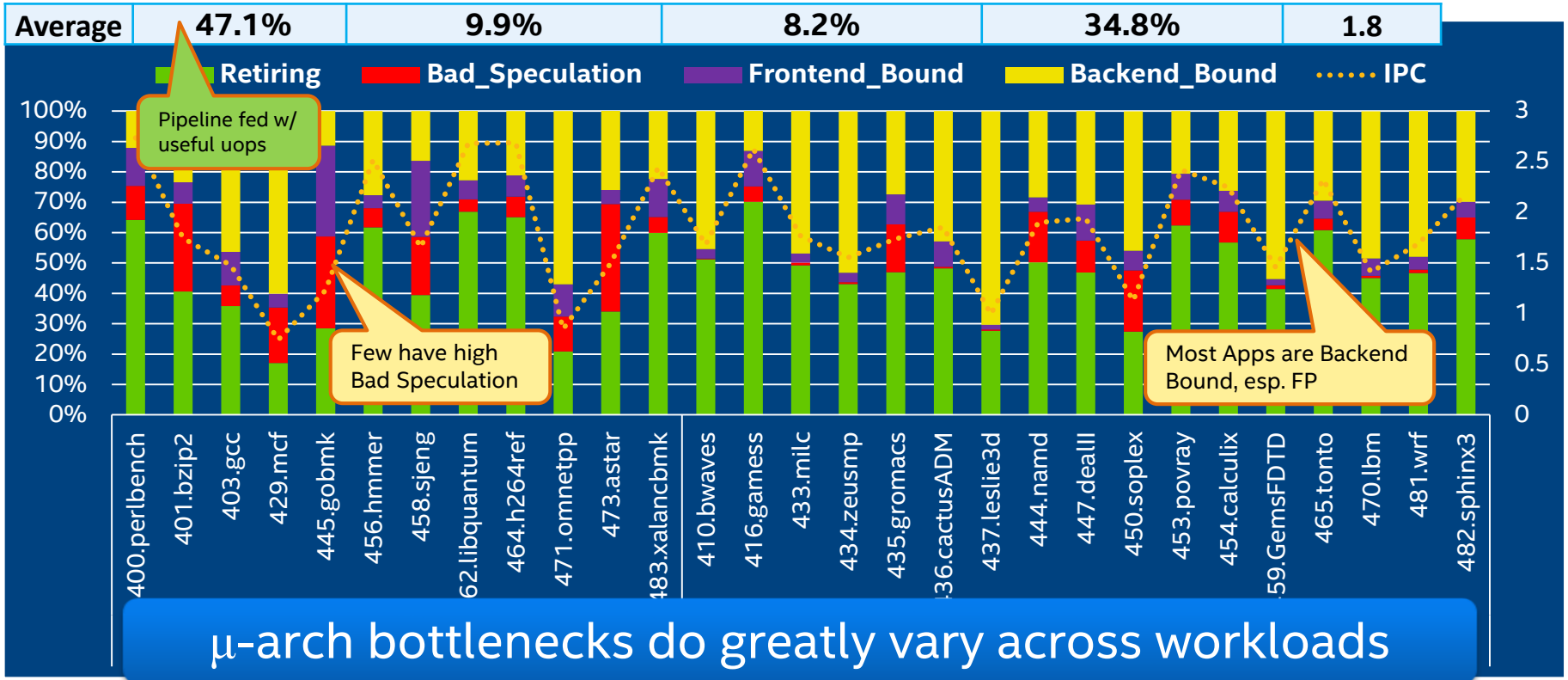
- Example
  - Branch Misprediction penalty = 50 * # Pipeline Clears // JEClear
- Unsuitable for modern out-of-order cores due to (Gaps):
  1) *Stalls Overlap*
  2) Speculative Execution
  3) Workload-dependent penalties
  4) Predefined set of miss-events
  5) Superscalar inaccuracy

Top Level Breakdown



Uop := micro-operation. Each x86 instruction is decoded into uop(s)
Uop Issue := last front-end stage where a uop is ready to acquire back-end resources
Back-end stall := Any backend resource fills up which blocks issue of new uops

TMA Top Level for SPEC CPU2006



μ-arch bottlenecks do greatly vary across workloads

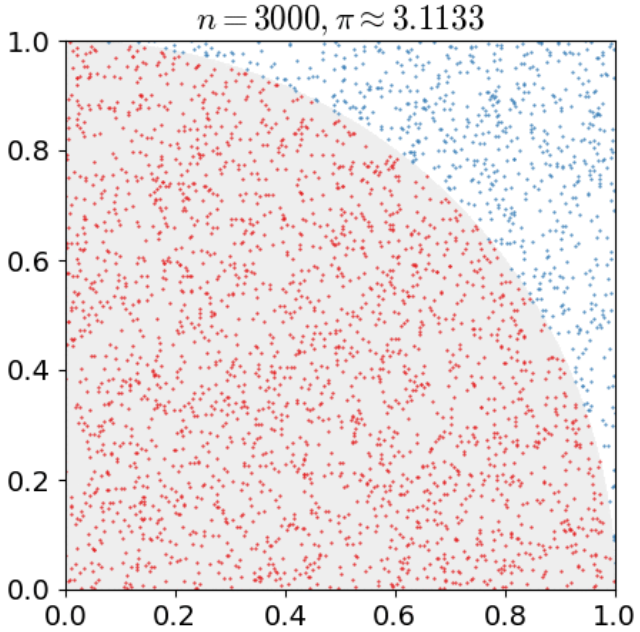
SPEC CPU2006 v1.2, rate 1-copy, Intel Complier 14 targeting AVX2, Skylake @ 3 GHz

picalc

- *Pseudo Code*

```
count=0
do n_trials:
   x = random from [0, 1]
   y = random from [0, 1]
   if x²+y² < 1: count++
return 4.0 * count / n_trials
```

- Another Monte Carlo method for computing π is to draw a circle inscribed in a square, and randomly place dots in the square. The ratio of dots inside the circle to the total number of dots will approximately equal π/4. Source: Wikipedia



Linux perf Demo

```
% perf stat picalc 0  # default

% echo 0 > /proc/sys/kernel/nmi_watchdog
% perf stat --topdown -a -- ./picalc 0     # Topdown μarch Analysis – supported starting Linux kernel 4.8

% perf stat -M GFLOPs -- ./picalc 0        # Metrics, single threaded

# Metrics & Groups, multithreaded
% perf stat -M GFLOPs -- ./picalc 1
% perf stat -M IPC -- ./picalc 1
% perf stat -M Summary -- ./picalc 1
```

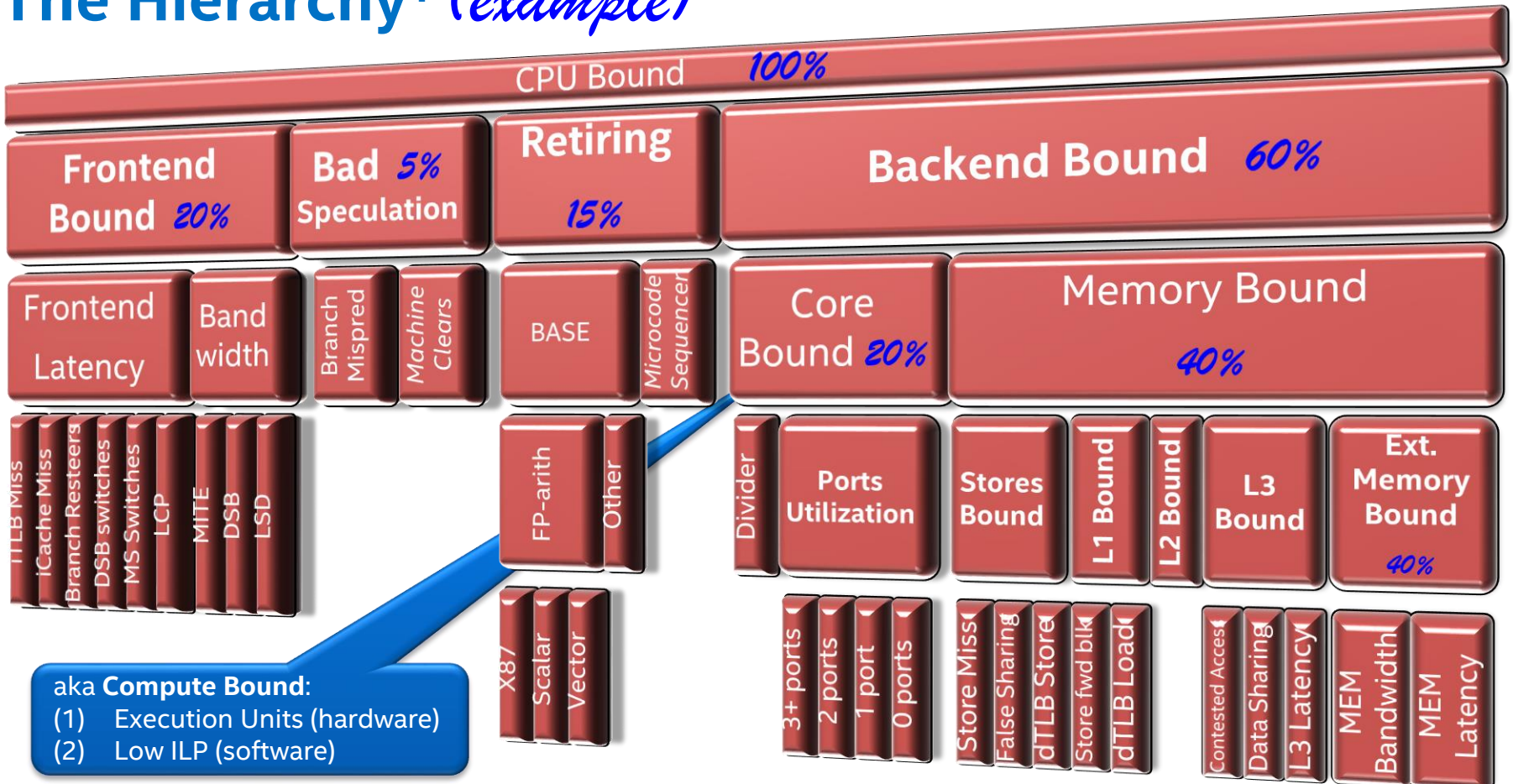
```
$ sudo perf stat -a --topdown ./main

nmi_watchdog enabled with topdown. May give wrong results.
Disable with echo 0 > /proc/sys/kernel/nmi_watchdog

 Performance counter stats for 'system wide':

                   retiring             bad speculation       frontend bound       backend bound
S0-C0          2     74.5%                 0.2%                 1.9%                 23.4%
S0-C1          2     17.3%                 6.3%                48.9%                 27.5%
S0-C2          2     16.3%                 7.4%                50.9%                 25.4%
S0-C3          2     15.2%                 8.0%                50.5%                 26.3%
```

Informative Metrics

- Non-tree metrics that helps to understand performance
- Also posted in:
  - https://download.01.org/perfmon/TMA_Metrics.xlsx
- Samples listed in table

Category	Example Metric	Brief Description
System	CPU_Utilization	# CPUs utilized
	GFLOPs	Floating Point performance
	MEM_BW_Use	Total Memory bandwidth
	SMT_2T_Utilization	Hyperthreading usage
Memory	Load_Miss_Real_Latency	# cycles per demand load miss
	MLP	Miss Level Parallelism
Thread	IPC (CPI)	Instructions Per Cycle
Core	CoreIPC	Physical core IPC
	ILP	Instruction Level Parallelism

The Hierarchy[1] (example)



[1] A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

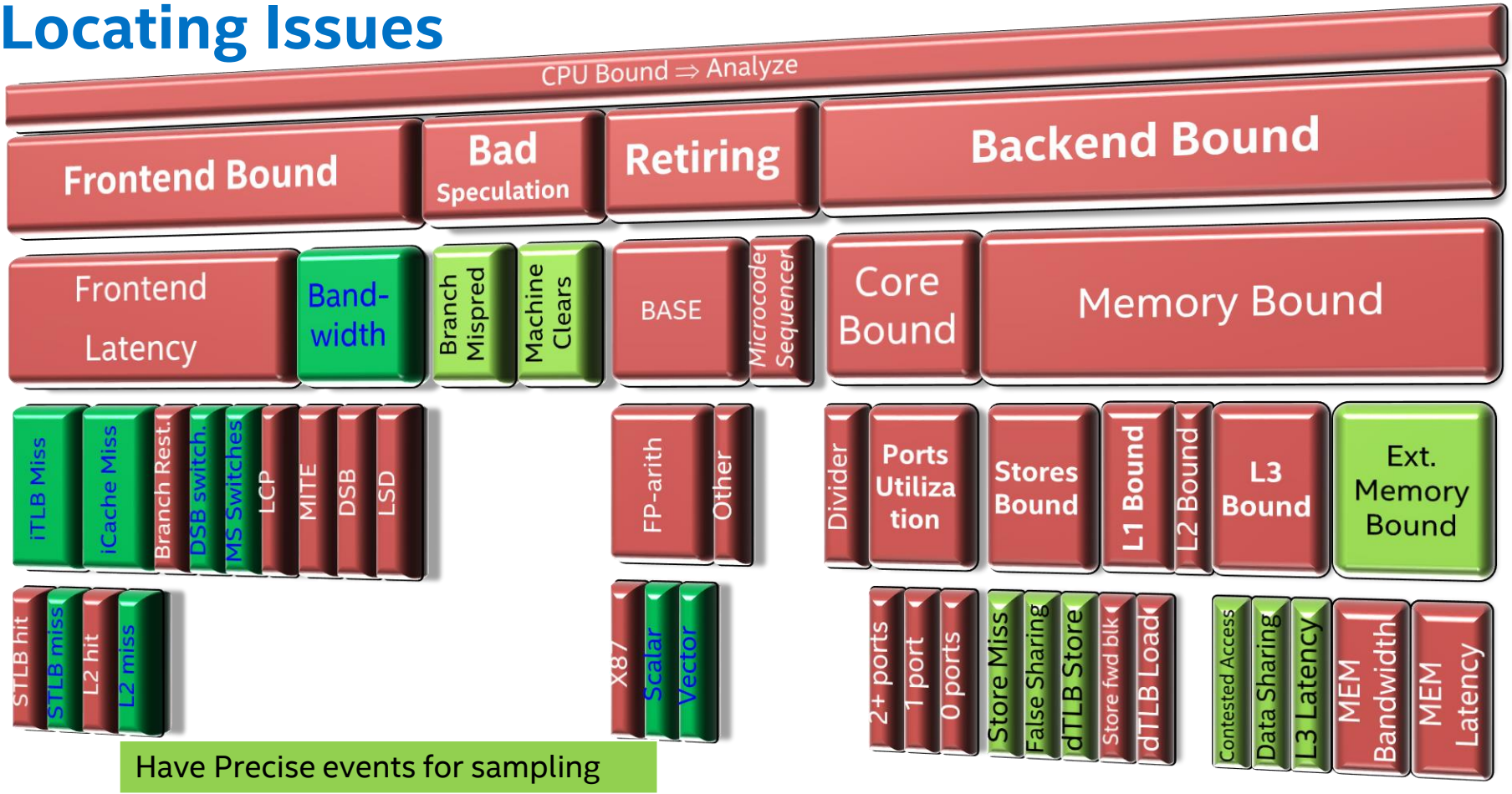
Intel Skylake offers an advanced PMU

Table 1. Comparison of Skylake's performance monitoring unit versus the predecessor.

Feature		Haswell	Skylake
PerfMon	Arch PerfMon version	3	4
	Events coverage (of microarchitecture)		Richer
	Events quality		Better
	Top Down Analysis (TMAM)	Basic (SMT-off)	Accurate (SMT on)
PEBS	Front-end events coverage	No	Yes
	Time-stamp counter included	No	Yes
LBR	Timing information	No	Yes
	Number of entries	16	32

Source: Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake. Jack Doweck, Wen-Fu Kao, Allen Kuan-yu Lu, Julius Mandelblat, Anirudha Rahatekar, Lihu Rappoport, Efraim Rotem, Ahmad Yasin, Adi Yoaz. IEEE Micro, Volume 37, Issue 2, 2017. [IEEE]

Locating Issues



Have Precise events for sampling

Precise events added in Skylake

Demo II: pmu-tools/toplev

```
% toplev.py ./picalc 1        # Default: 1 level, print what matters

% toplev.py -l2 -- ./picalc 1        # 2 levels
% toplev.py -v –l3 -- ./picalc 1     # 3 levels, print everything
## the deeper the level, the higher the counter multiplexing rate

% toplev.py -l2 -m -- ./picalc 1     # 2 levels with info metrics
% toplev.py -l4 --no-desc --show-sample -- ./picalc 1    # 4 levels,
no descriptions, show the right ‘perf record’ command for my code

% toplev.py -mvl5 --no-multiplex -- ./picalc 1 # Collect everything
and do not multiplex counters (do multiple runs)
```

toplev for multithreaded pi: Good vs. False Sharing

```
% toplev.py -x, -l4 --nodes +IPC,+GFLOPs ./picalc 1 | cut -d, -f1-3,7 | sed 's/_Bound\./\./g'
# 3.32-full on Intel Core i7-6700K CPU @ 4.00GHz
Using TMA version 3.32-full, level 4.
./picalc: #trials=1000.0 Million, mode=1, scale=1
Parallel1: pi=3.141585 Time=2.029217

Backend_Bound,41.66,% Slots,5.19
IPC,2.097,Metric,5.06
Backend.Core_Bound,37.86,% Slots,5.15
Backend.Core.Divider,49.43,% Clocks,5.16
GFLOPs,2.405,Metric,5.06
```

```
% toplev.py -x, -l4 --nodes +IPC,+GFLOPs ./picalc 3 | cut -d, -f1-3,7 | sed 's/_Bound\./\./g'
# 3.32-full on Intel Core i7-6700K CPU @ 4.00GHz
Using TMA version 3.32-full, level 4.
./picalc: #trials=1000.0 Million, mode=3, scale=1
Parallel+false sharing: pi=3.141585 Time=7.430190
Bad_Speculation,11.99,% Slots,5.05
Backend_Bound,68.78,% Slots,5.05
IPC,0.639,Metric,5.02
Backend.Memory_Bound,57.60,% Slots,5.05
Backend.Memory.Store_Bound,55.43,% Stalls,5.05
Backend.Memory.Store.False_Sharing,15.52,% CE*,5.02
Backend.Memory.Store.Store_Latency,73.52,% CE*,4.99
GFLOPs,0.667,Metric,5.02
                                        *CE = Clocks_Estimated
```

False Sharing

- Consider a code like:

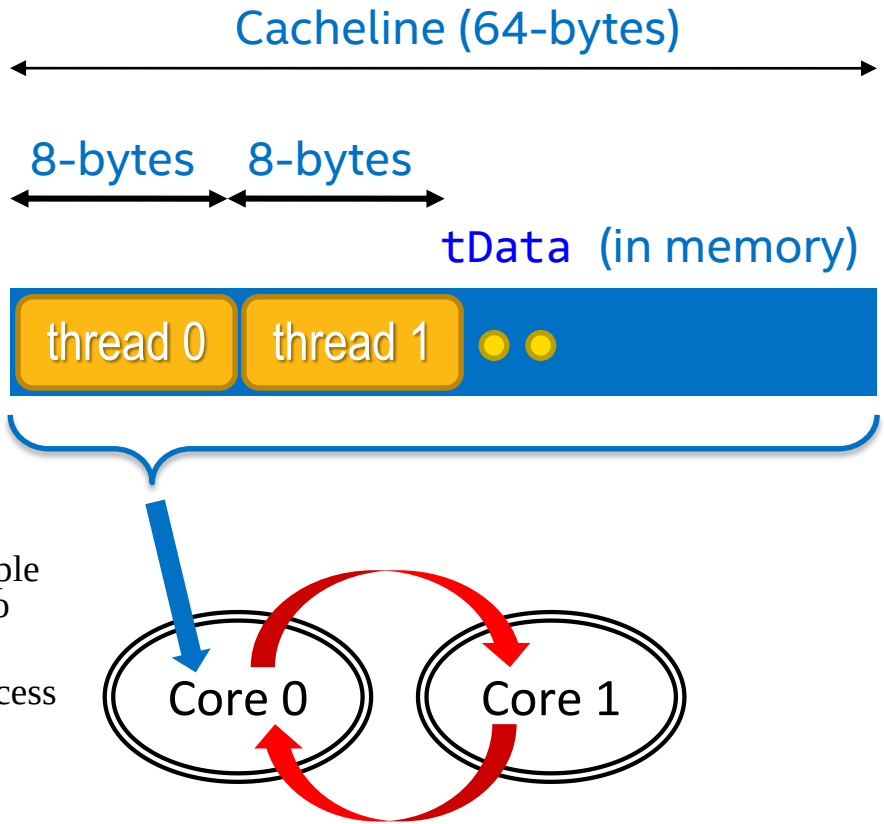
```
int tData[MAX_THREADS];
...
// a code of thread x:
tData[thread_id]++;
```

- More Info + Case Study:

  "False Sharing is a multithreading hiccup, where multiple threads contend on different data-elements mapped into the same cache line.

  It can be easily avoided by padding to make threads access different lines."

  Source: Yasin, SPASS 2014. [paper]



Performance Analysis

Sample use-cases & Summary

Optimizing Matrix Multiply (through VTune)

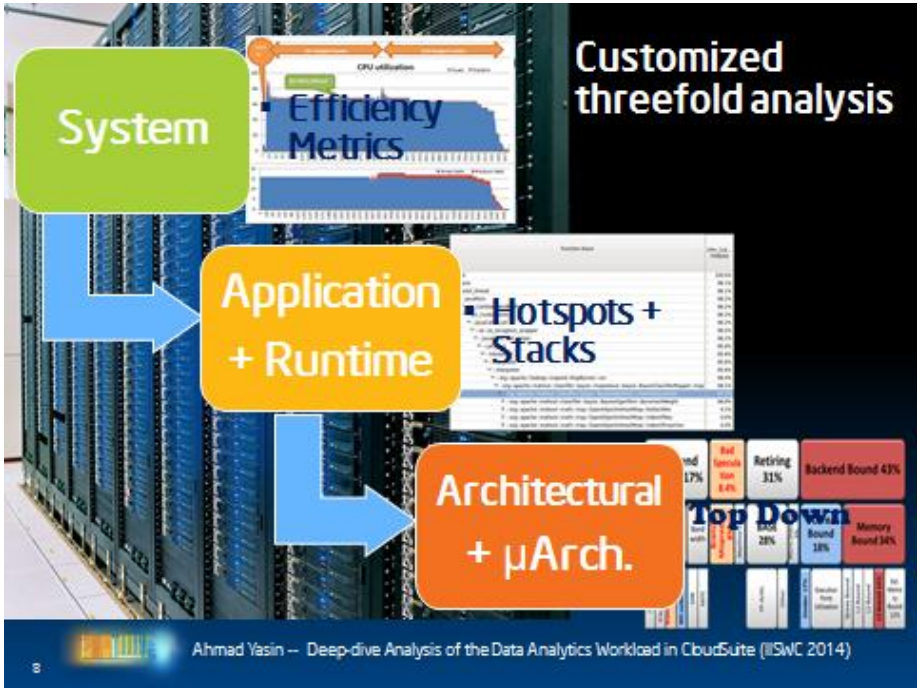
Step: Optimization	Time [s]	Speed up	CPI (*1)	Instructions [Billions]	DRAM Bound (*3)	BW Utilization (*4) [GB/s]	CPU Util-ization (*5)
1: None (textbook version)	73.9	1.0x	3.71	52.08	80.1%	7.2	1
2:(*2) Loop Interchange	7.68	9.6x	0.37	56.19	10.4%	10.5	1
3: Vectorize inner loop (SSE)	*6.87*	*10.8x*	*0.92*	*20.83*	*20.2%*	*11.6*	*1*
4: Vectorize inner loop (AVX2)	6.39	11.6x	1.40	12.73	18.2%	11.8	1
5: Use Fused Multiply Add (FMA)	6.06	12.2x	1.93	8.42	47.7%	12.6	1
6: Parallelize outer loop (OpenMP)	3.59	20.6x	3.02	8.59	61.6%	13.8	2.8

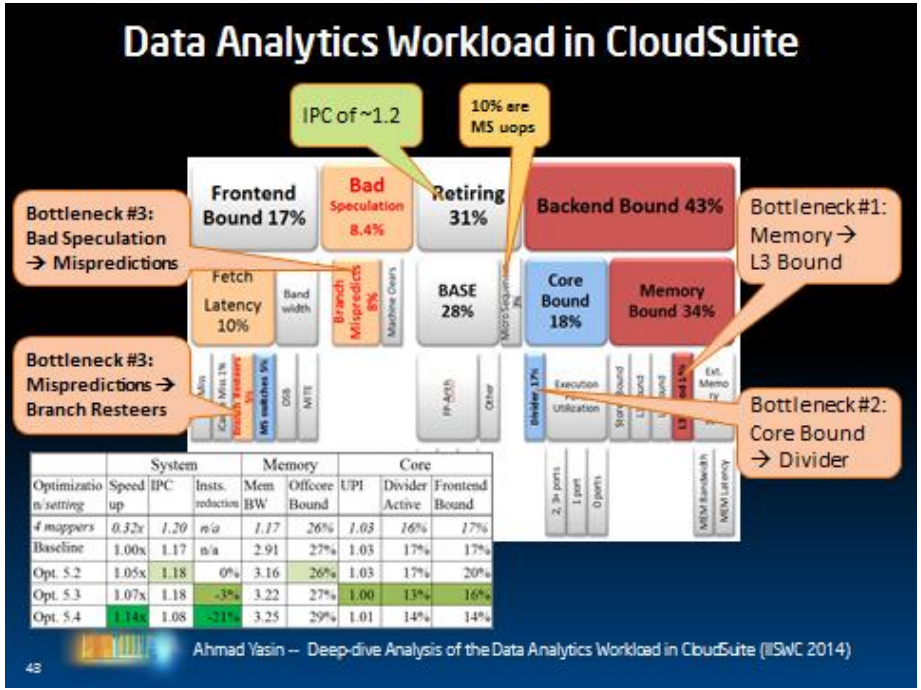
(*1) Cycles Per Instruction
(*2) Had to set 'CPU sampling interval, ms' to 0.1 starting this step since run time went below 1 minute
(*3) TopDown's Backend_Bound.Memory_Bound.DRAM_Bound metric under VTune's General Exploration viewpoint
(*4) Per 'Average Bandwidth' (for DRAM) under Vtune's 'Memory Usage' viewpoint.
(*5) Per 'Average Effective CPU Utilization' line in Effective CPU Usage Histogram

See full presentation: http://cs.haifa.ac.il/~yosi/PARC/yasin.pdf

A Server Workload Optimization

Deep-dive Analysis of the Data Analytics Workload in CloudSuite - Ahmad Yasin, Yosi Ben-Asher, Avi Mendelson. *In IEEE International Symposium on Workload Characterization, IISWC 2014.* [paper] [slides]





Optimization setting	System: Speed up	System: IPC	System: Insts. reduction	Memory: Mem BW	Memory: Offcore Bound	Core: UPI	Core: Divider Active	Core: Frontend Bound
4 mappers	*0.32x*	*1.20*	*n/a*	*1.17*	*26%*	*1.03*	*16%*	*17%*
Baseline	1.00x	1.17	n/a	2.91	27%	1.03	17%	17%
Opt. 5.2	1.05x	1.18	0%	3.16	26%	1.03	17%	20%
Opt. 5.3	1.07x	1.18	-3%	3.22	27%	1.00	13%	16%
Opt. 5.4	1.14x	1.08	-21%	3.25	29%	1.01	14%	14%

Datacenter Profiling

- Profiling a Warehouse-Scale Computer - S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G. Wei and D. Brooks, in International Symposium on Computer Architecture (ISCA), June 2015.
  - A highly-cited work by Google and Harvard

- First to *profile* a production datacenter
  - Mixture of μ-arch bottlenecks
    - Stalled on data most often
    - Heavy pressure on i-cache
    - Compute in bursts
    - Low memory BW utilization

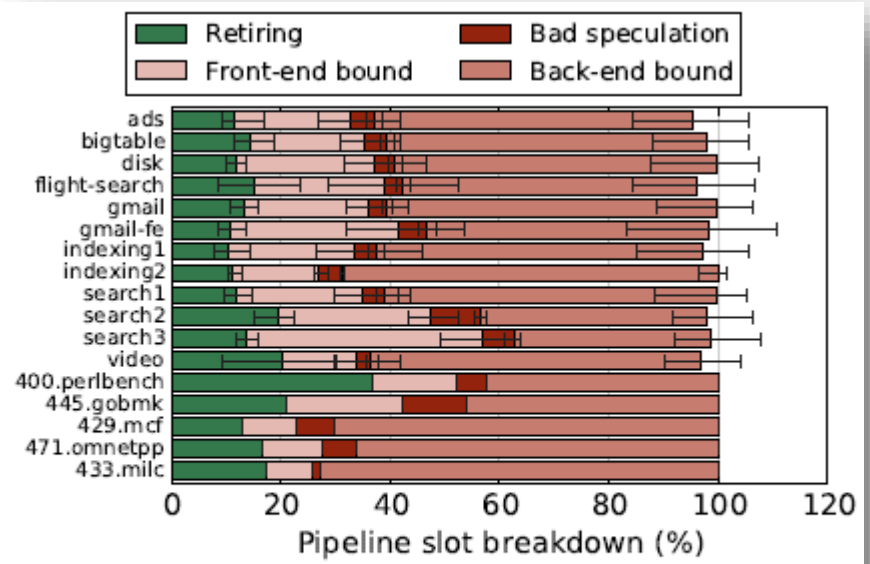


Figure 4: Top-level bottleneck breakdown. SPEC CPU2006 benchmarks do not exhibit the combination of low retirement rates and high front-end boundedness of WSC ones.

Summary

- Tools can provide insights on the actual execution to enable software developers to optimize their applications and thus further increase performance
- Top-down Microarchitectural Analysis (TMA) simplifies performance analysis and eliminates the “guess work”
- Linux perf events standardizes the access to, and exploits advanced features of the Performance Monitoring Unit. This lead to powerful off-the-shelf tools (perf tool, toplev)

Useful pointers

- Top-down Analysis
  - A Top-Down Method for Performance Analysis and Counters Architecture, Ahmad Yasin. In IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014. [paper] [slides]
  - Software Optimizations Become Simple with Top-Down Analysis Methodology on Intel® Microarchitecture Code Name Skylake, Ahmad Yasin. Intel Developer Forum, IDF 2015. [Recording] [session direct link] [link#2]
  - TMA Metrics spreadsheet: https://download.01.org/perfmon/
- Recent Lectures:
  - Perf Analysis in Out-of-order cores: http://webcourse.cs.technion.ac.il/234267/Winter2016-2017/ho/WCFiles/Perf%20Analysis%20in%20OOO%20cores%20-%20Ahmad%20Yasin.pdf
  - Using Intel PMU through VTune: http://cs.haifa.ac.il/~yosi/PARC/yasin.pdf
- Linux tools
  - **Toplev** by Andi Kleen: https://github.com/andikleen/pmu-tools/wiki/toplev-manual
  - latest perf tool
    - `git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git; cd linux/tools/perf/; make`
- Free Intel tools for students, including VTune:

  >>> https://software.intel.com/en-us/qualify-for-free-software/student