Practical Parallel Programming
course - Winter 2017
CS @ Haifa University

# Performance Analysis in Modern Multicores

Ahmad Yasin
CPU Architect, Intel Corporation

14 January 2018

experience
what's inside™

## Outline

- Architecture
- Performance Monitoring
- **Hands on 1:** VTune
  - Introduction, Advanced Hotspots, custom analysis, OpenMP support
- Top-down Microarchitecture Analysis (TMA) method
- **Hands on 2:** Matrix Multiply Optimizations
  - General Exploration (TMA) reflecting Parallelization, Vectorization, μarch tuning
- Summary & Pointers

# Skylake processor & memory subsystem

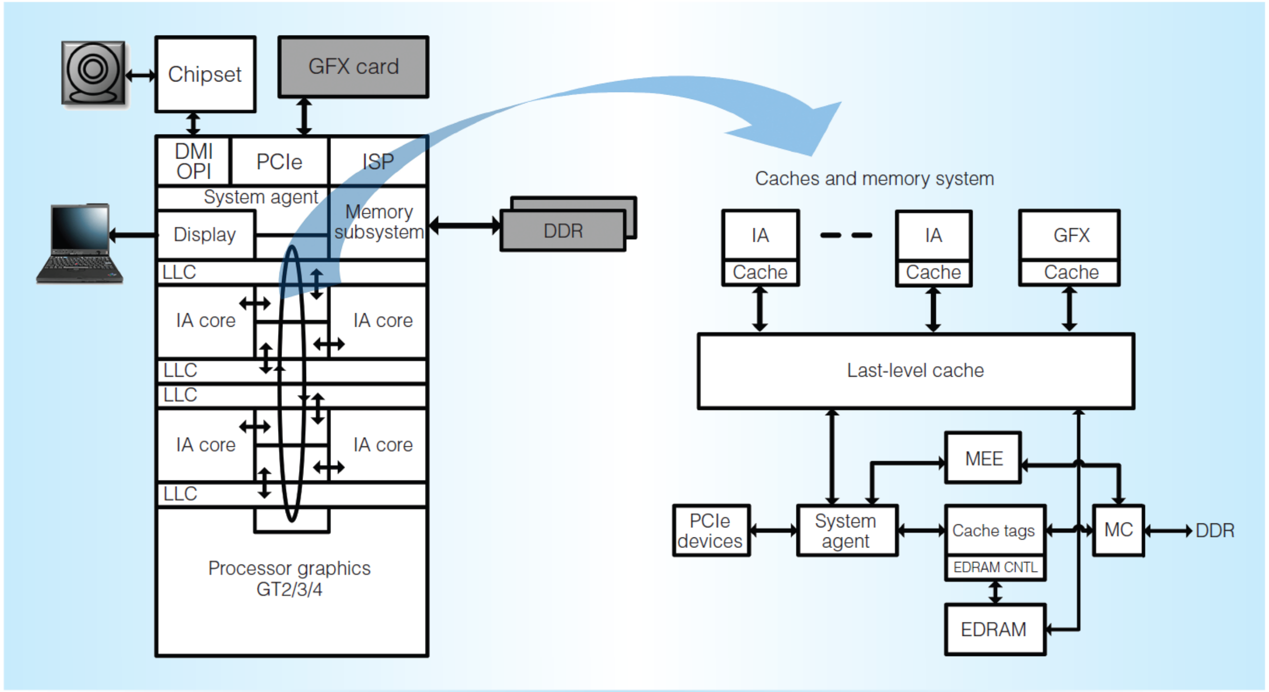
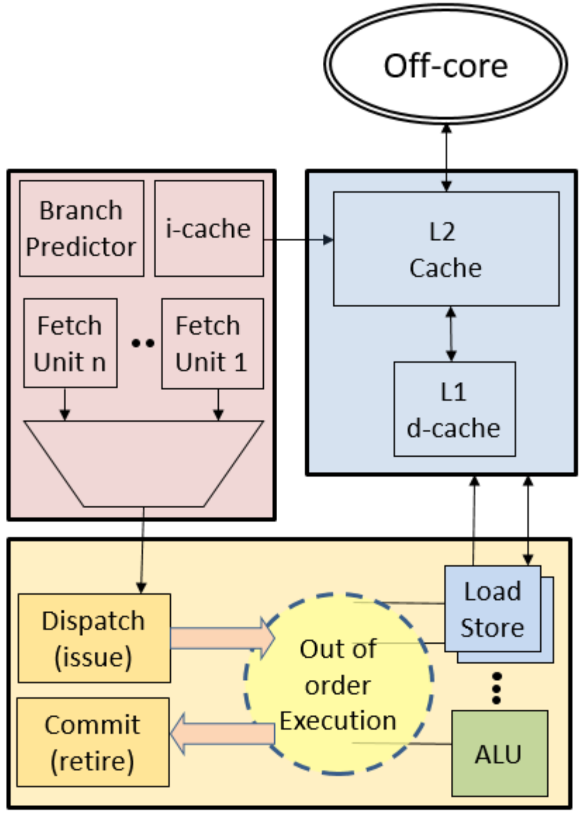


Figure 1. Skylake block diagram with zoom into cache and memory subsystem. The memory subsystem is shown with the eDRAM-based memory side cache.

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

## Skylake Core microarchitecture

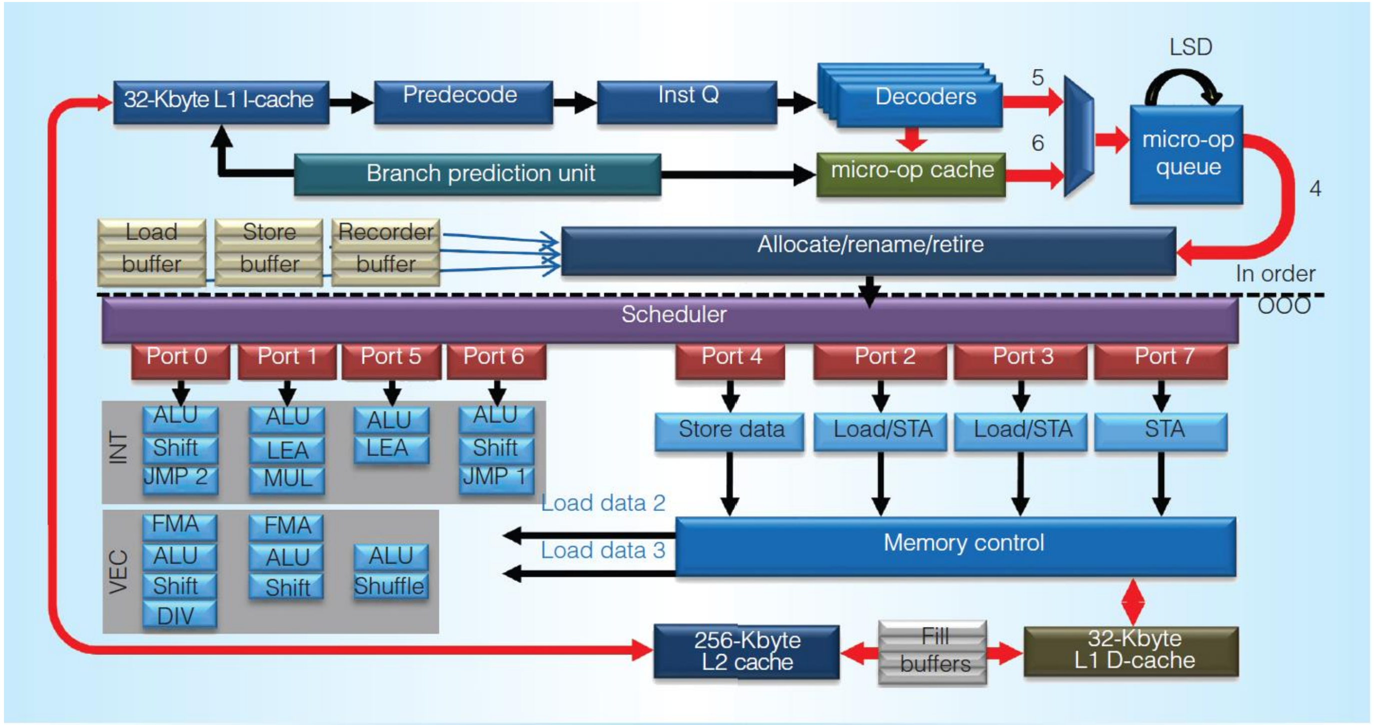


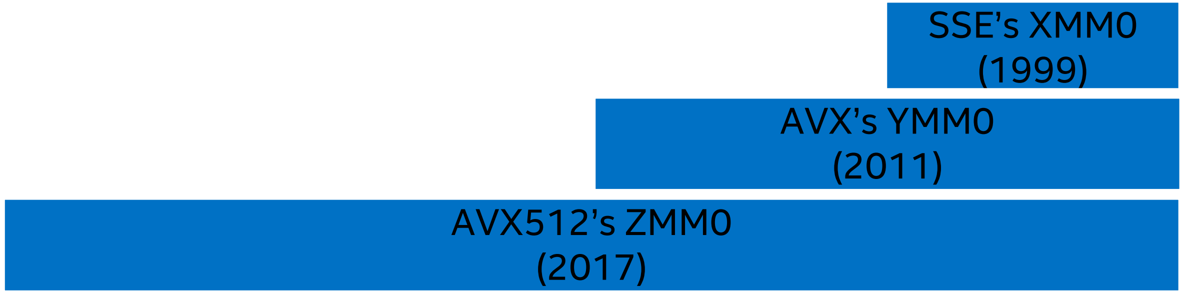
Figure 4. Skylake core block diagram.

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]

# X86 vectors for Floating Point (FP)

- FP
  - Single Precision (SP)
    - 32-bits
    - 8 elements in AVX2
    - ‘float’ in C
  - Double Precision
    - 64-bits
    - 4 elements in AVX2
    - ‘double’ in C

| ISA | Max Vector width | FP-SP elements | Introduced in processor |
|---|---|---|---|
| SSE | 128-bit | 4 | |
| AVX | 256-bit FP only | 8 | Sandy Bridge (2nd gen Core) |
| AVX2 | 256-bit (adds integer, FMA) | 8 | Haswell (4th gen Core) |
| AVX512 | 512-bit | 16 | Skylake Server (Xeon Scalable) |



## Performance Monitoring

# Performance Counters 101

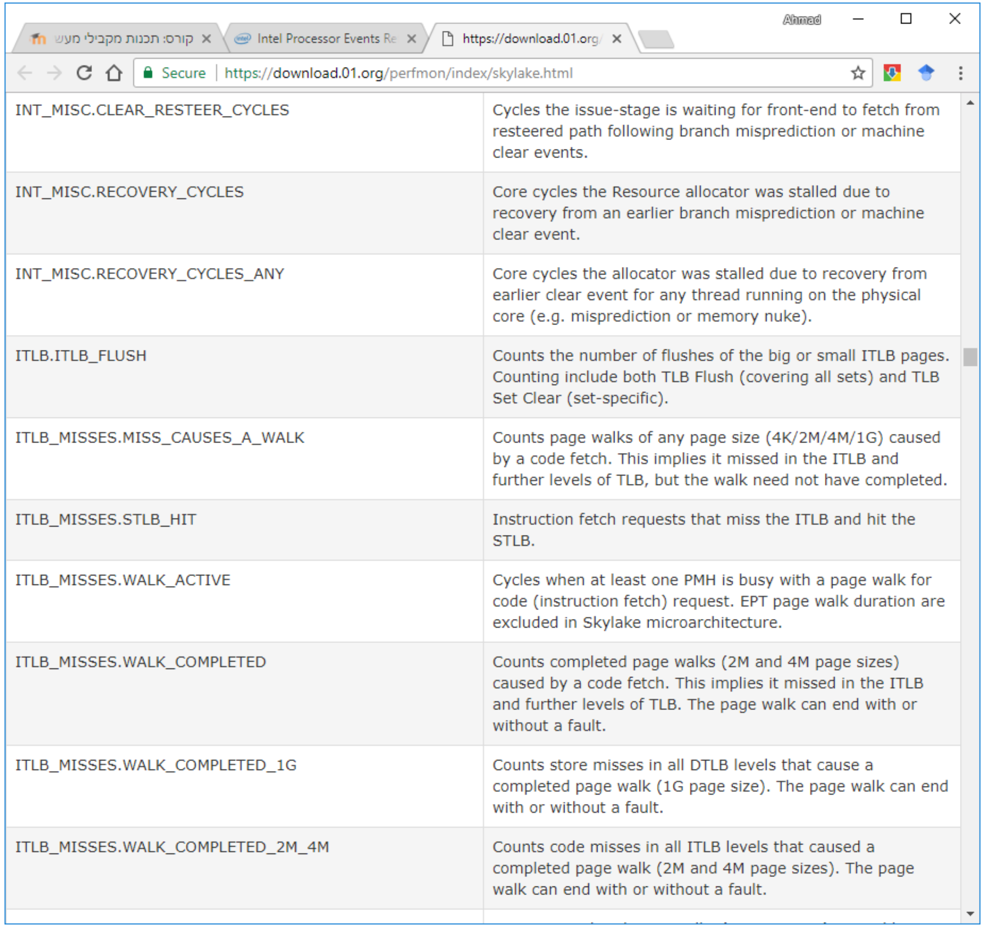
- **PMU** – Performance Monitoring Unit
  - **Non-intrusive** generic set of capabilities to retrieve info related to CPU execution
  - Info: Mostly on **microarchitecture performance** but also Arch., Energy, and more
  - Probably the single such capability to provide what's going on under the hood
  - Example usage: avoid u-arch inefficiencies via **code tuning**
- Terms
  - General-purpose **counters**
  - Predefined HW **events**
  - Counting vs. Sampling modes
  - **PMI** (PerfMon Interrupt) & **Samples**, Sample-After-Value

## Intel® Core™ Processor PMU

- Architectural PMU (Performance Monitoring Unit)
  - 3 fixed counters
    - Instructions Retired, Core- and Reference-Unhalted-Cycles
  - 4 general-purpose counters
    - Expands to 8 when SMT is off (since Sandy Bridge)
  - Global Status and Controls
- Rich list of performance events
  - Most useful events are model-specific
  - A subset of 7 events are architected
- Advanced mechanisms
  - PEBS: Precisely tags performance events & profiling data into software location
  - LBR: Non-intrusive branch recording facility

# Perf. Counters
(or PerfMon events)

- Links
  - VTune Index to PerfMon Tables of all Intel processors: https://software.intel.com/en-us/vtune-amplifier-help-intel-processor-events-reference
  - E.g. Skylake: https://download.01.org/perfmon/index/skylake.html


קורס: תכנות מקבילי מעש
Intel Processor Events Re
https://download.01.org/
Ahmad
Secure
https://download.01.org/perfmon/index/skylake.html

| | |
|---|---|
| INT_MISC.CLEAR_RESTEER_CYCLES | Cycles the issue-stage is waiting for front-end to fetch from resteered path following branch misprediction or machine clear events. |
| INT_MISC.RECOVERY_CYCLES | Core cycles the Resource allocator was stalled due to recovery from an earlier branch misprediction or machine clear event. |
| INT_MISC.RECOVERY_CYCLES_ANY | Core cycles the allocator was stalled due to recovery from earlier clear event for any thread running on the physical core (e.g. misprediction or memory nuke). |
| ITLB.ITLB_FLUSH | Counts the number of flushes of the big or small ITLB pages. Counting include both TLB Flush (covering all sets) and TLB Set Clear (set-specific). |
| ITLB_MISSES.MISS_CAUSES_A_WALK | Counts page walks of any page size (4K/2M/4M/1G) caused by a code fetch. This implies it missed in the ITLB and further levels of TLB, but the walk need not have completed. |
| ITLB_MISSES.STLB_HIT | Instruction fetch requests that miss the ITLB and hit the STLB. |
| ITLB_MISSES.WALK_ACTIVE | Cycles when at least one PMH is busy with a page walk for code (instruction fetch) request. EPT page walk duration are excluded in Skylake microarchitecture. |
| ITLB_MISSES.WALK_COMPLETED | Counts completed page walks (2M and 4M page sizes) caused by a code fetch. This implies it missed in the ITLB and further levels of TLB. The page walk can end with or without a fault. |
| ITLB_MISSES.WALK_COMPLETED_1G | Counts store misses in all DTLB levels that cause a completed page walk (1G page size). The page walk can end with or without a fault. |
| ITLB_MISSES.WALK_COMPLETED_2M_4M | Counts code misses in all ITLB levels that caused a completed page walk (2M and 4M page sizes). The page walk can end with or without a fault. |

Intel Performance tools

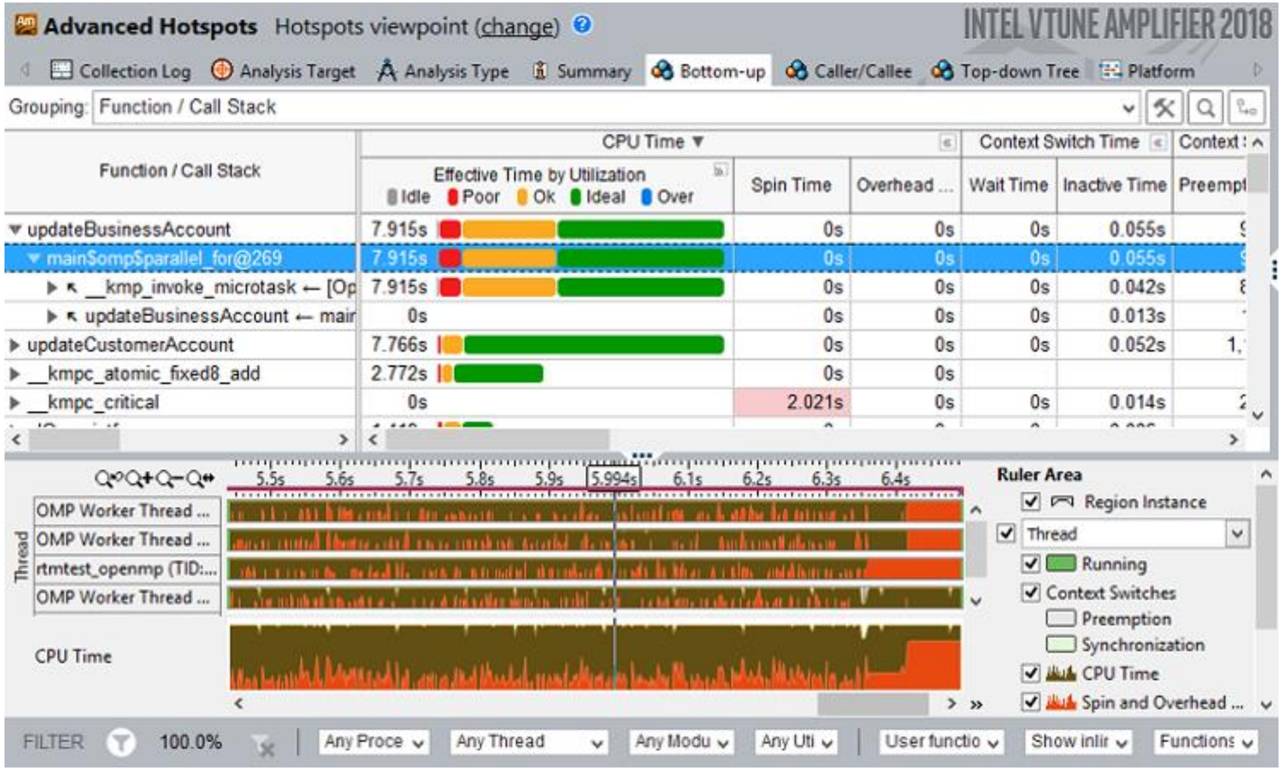
## VTune – the rich profiler

# Performance Tools Map

| **Tool** | **Description** |
|---|---|
| **Intel® VTune™ Amplifier** | Performance Profiler |
| **Intel® Top-Down** | Bottleneck analysis using  Performance Monitoring Units |
| *Intel® Emon* | Event monitor – collect HW counters |
| *Intel® Compiler* | Highly optimizing compiler, lead in autovectorization |
| Intel® Architecture Code Analyzer | Static performance analyzer, good for theoretical performance threshold. |
| Intel® Intrinsics Guide | Intel GUI tools that helps writing in intrinsics |
| Intel® Integrated Performance Primitives (Intel® IPP) | low-level building blocks for image processing, signal processing, and data processing |
| Intel® Math Kernel Library | highly optimized, threaded, and vectorized math functions that maximize performance on each processor family |
| Intel® Threading Building Blocks (Intel® TBB) | Widely used C++ template library for task parallelism |

Ack: Asaf Hargil

# VTune



| Function / Call Stack | CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over) | CPU Time: Spin Time | CPU Time: Overhead ... | Context Switch Time: Wait Time | Context Switch Time: Inactive Time | Context: Preempt |
|---|---|---|---|---|---|---|
| ▼ updateBusinessAccount | 7.915s | 0s | 0s | 0s | 0.055s | 9 |
| ▼ main$omp$parallel_for@269 | 7.915s | 0s | 0s | 0s | 0.055s | 9 |
| ▶ ↖ __kmp_invoke_microtask ← [Op | 7.915s | 0s | 0s | 0s | 0.042s | 8 |
| ▶ ↖ updateBusinessAccount ← mair | 0s | 0s | 0s | 0s | 0.013s | 1 |
| ▶ updateCustomerAccount | 7.766s | 0s | 0s | 0s | 0.052s | 1, |
| ▶ __kmpc_atomic_fixed8_add | 2.772s | 0s | 0s | | | |
| ▶ __kmpc_critical | 0s | 2.021s | 0s | 0s | 0.014s | 2 |

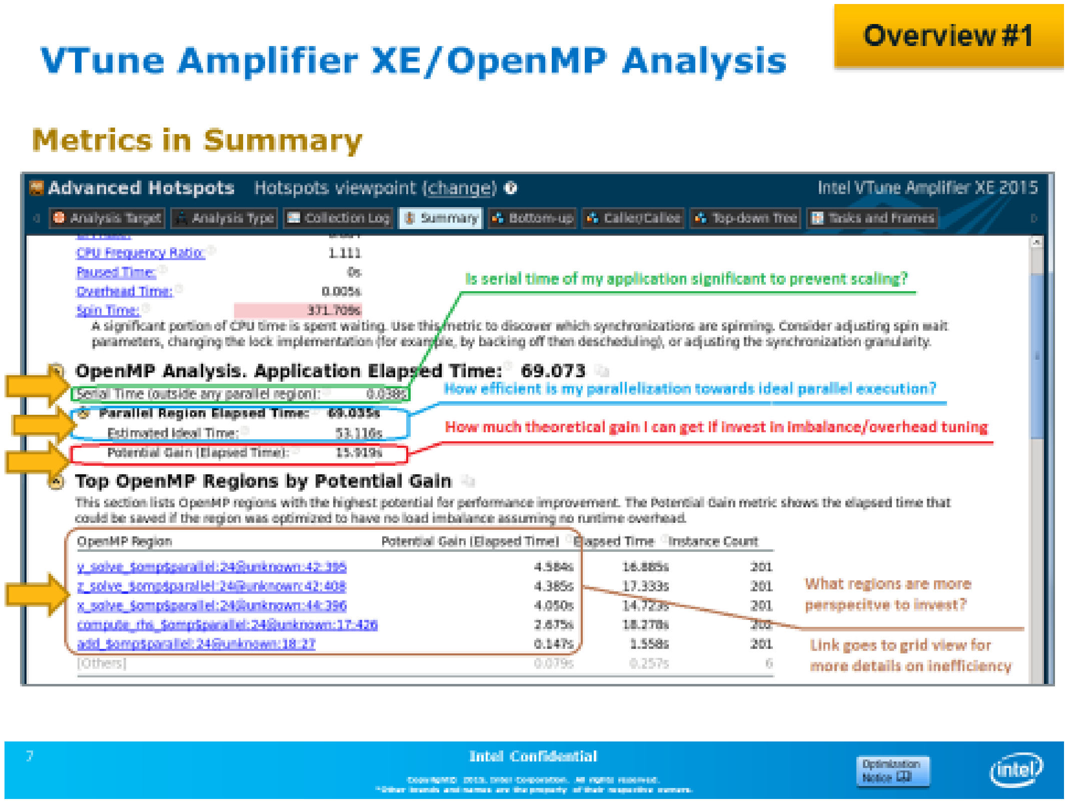
VTune 2018 Introduction

## Hands On #1

## VTune small tips

- Installation
  - Disable Virtualization (in BIOS) for Windows 10
- Measurements
  - Close all applications (e.g. Chrome) and unneeded services or even do a clean boot
  - Disable SMT (Hyperthreading) and Turbo Frequency if you can
    - Often introducing indeterminism and run-to-run variations
    - Remember to turn it on once done
  - Use affinity esp. when utilizing subset of cores/threads
    - E.g. `taskset 0x<mask>` in Linux or `start /affinity 0x<mask>` in Windows
- Optimizations
  - Try one optimization/change at a time!

# VTune for OpenMP



Overview #1
VTune Amplifier XE/OpenMP Analysis
Metrics in Summary
Advanced Hotspots Hotspots viewpoint (change)
Intel VTune Amplifier XE 2015
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Tasks and Frames
Is serial time of my application significant to prevent scaling?
How efficient is my parallelization towards ideal parallel execution?
How much theoretical gain I can get if invest in imbalance/overhead tuning
What regions are more perspecitve to invest?
Link goes to grid view for more details on inefficiency
Intel Confidential
Optimization Notice

| | |
|---|---|
| CPU Frequency Ratio: | 1.111 |
| Paused Time: | 0s |
| Overhead Time: | 0.005s |
| Spin Time: | 371.709s |

A significant portion of CPU time is spent waiting. Use this metric to discover which synchronizations are spinning. Consider adjusting spin wait parameters, changing the lock implementation (for example, by backing off then descheduling), or adjusting the synchronization granularity.

**OpenMP Analysis. Application Elapsed Time: 69.073**

| | |
|---|---|
| Serial Time (outside any parallel region): | 0.038s |
| **Parallel Region Elapsed Time:** | **69.035s** |
| Estimated Ideal Time: | 53.116s |
| Potential Gain (Elapsed Time): | 15.919s |

**Top OpenMP Regions by Potential Gain**

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

| OpenMP Region | Potential Gain (Elapsed Time) | Elapsed Time | Instance Count |
|---|---|---|---|
| y_solve_$omp$parallel:24@unknown:42:396 | 4.584s | 16.885s | 201 |
| z_solve_$omp$parallel:24@unknown:42:408 | 4.385s | 17.333s | 201 |
| x_solve_$omp$parallel:24@unknown:44:396 | 4.050s | 14.723s | 201 |
| compute_rhs_$omp$parallel:24@unknown:17:426 | 2.675s | 18.278s | 202 |
| add_$omp$parallel:24@unknown:18:27 | 0.147s | 1.558s | 201 |
| [Others] | 0.079s | 0.257s | 6 |

7

Copyright© 2015, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Source: Enhanced OpenMP Analysis in VTune Amplifier XE 2015 - Ahmad Yasin for Corey Alsamariae and Dmitry Prohorov (Intel Corporation). Software Development Conference - Tel Aviv, 1st July 2015.

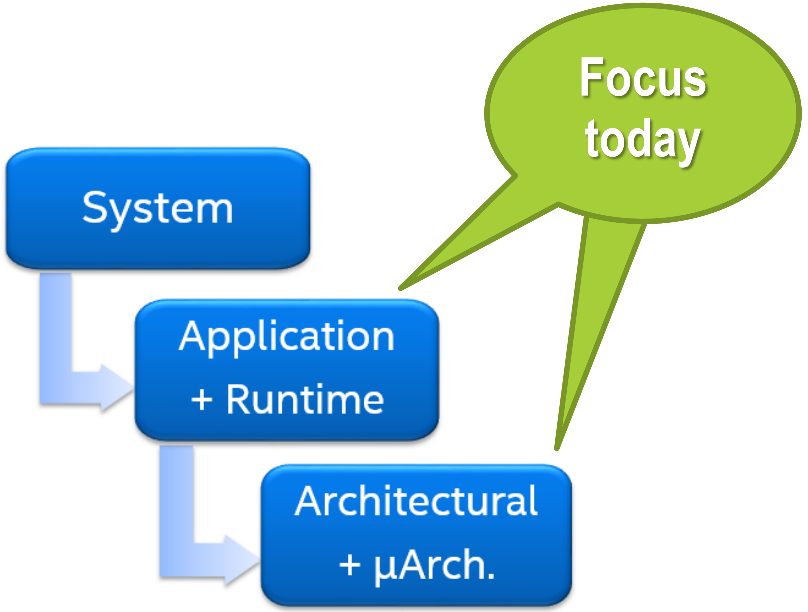
Performance Analysis

# Top-down Microarchitecture Analysis (TMA)

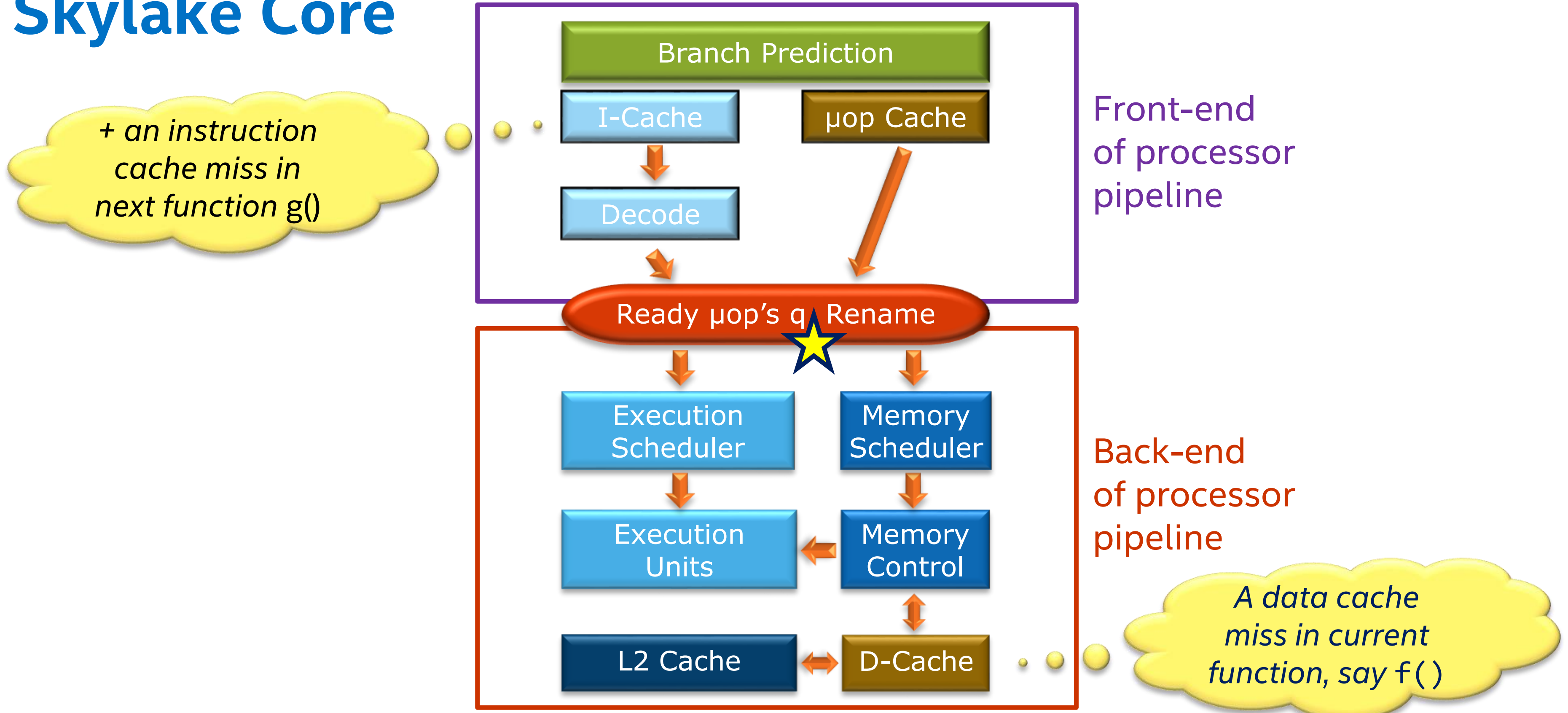
## What is Performance Analysis?

- 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](http://www.brendangregg.com/methodology.html)



# Skylake Core



+ an instruction cache miss in next function g()
Branch Prediction
I-Cache
µop Cache
Decode
Front-end of processor pipeline
Ready µop's q Rename
Execution Scheduler
Memory Scheduler
Execution Units
Memory Control
L2 Cache
D-Cache
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)

$$\text{Stall\_Cycles} = \Sigma\ \text{Penalty}_i * \text{MissEvent}_i$$

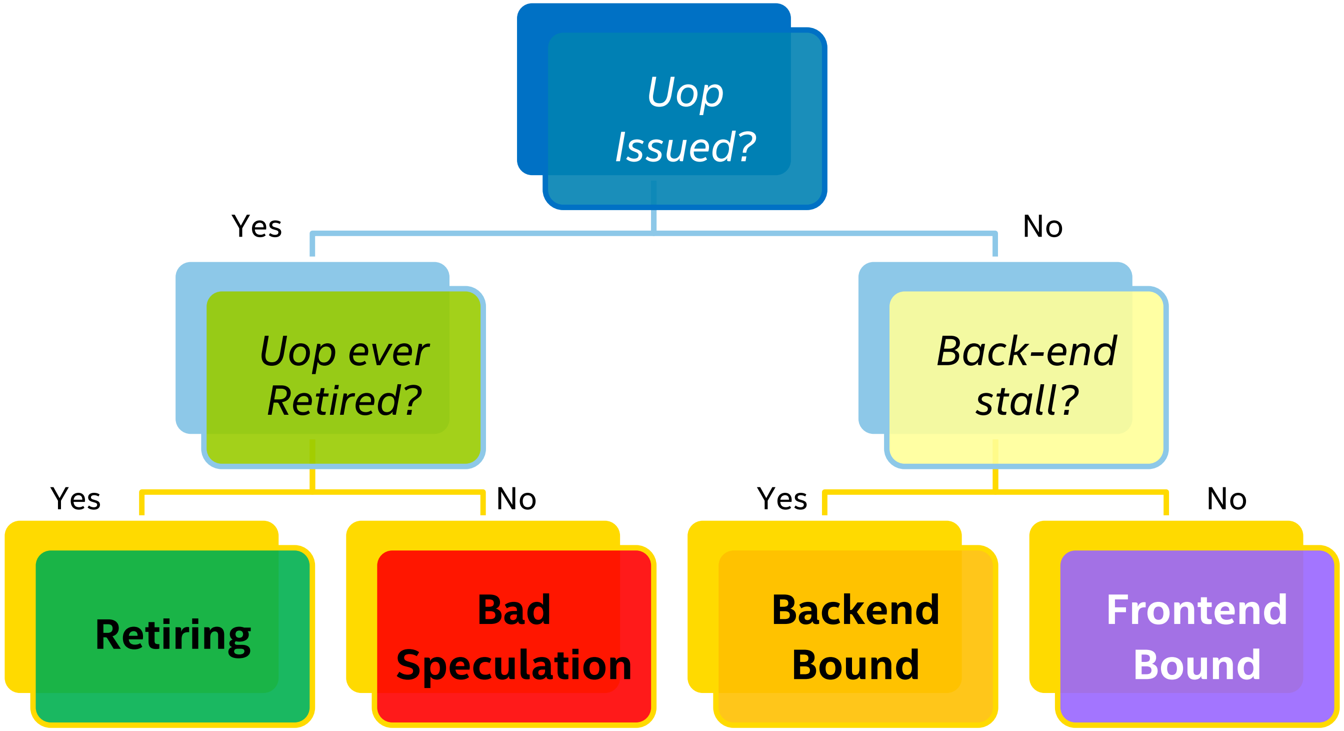
- Example
  - Branch Misprediction penalty = 50 * # Pipeline Clears // JEClear
- Unsuitable for modern out-of-order cores due to (Gaps):
  1) *Stalls Overlap*
  2) Speculative Execution
  3) Workload-dependent penalties
  4) Predefined set of miss-events
  5) Superscalar inaccuracy

## Top Down Analysis

- Identifies true µ-arch bottlenecks in a simple, structured hierarchical process
  - Simplicity avoids the µ-arch high-learning curve
    - i.e. Analysis is made easier for users who may lack hardware expertise
  - The hierarchy abstracts bottlenecks to cover many µ-arch's
  - The structured process eliminates "guess work"

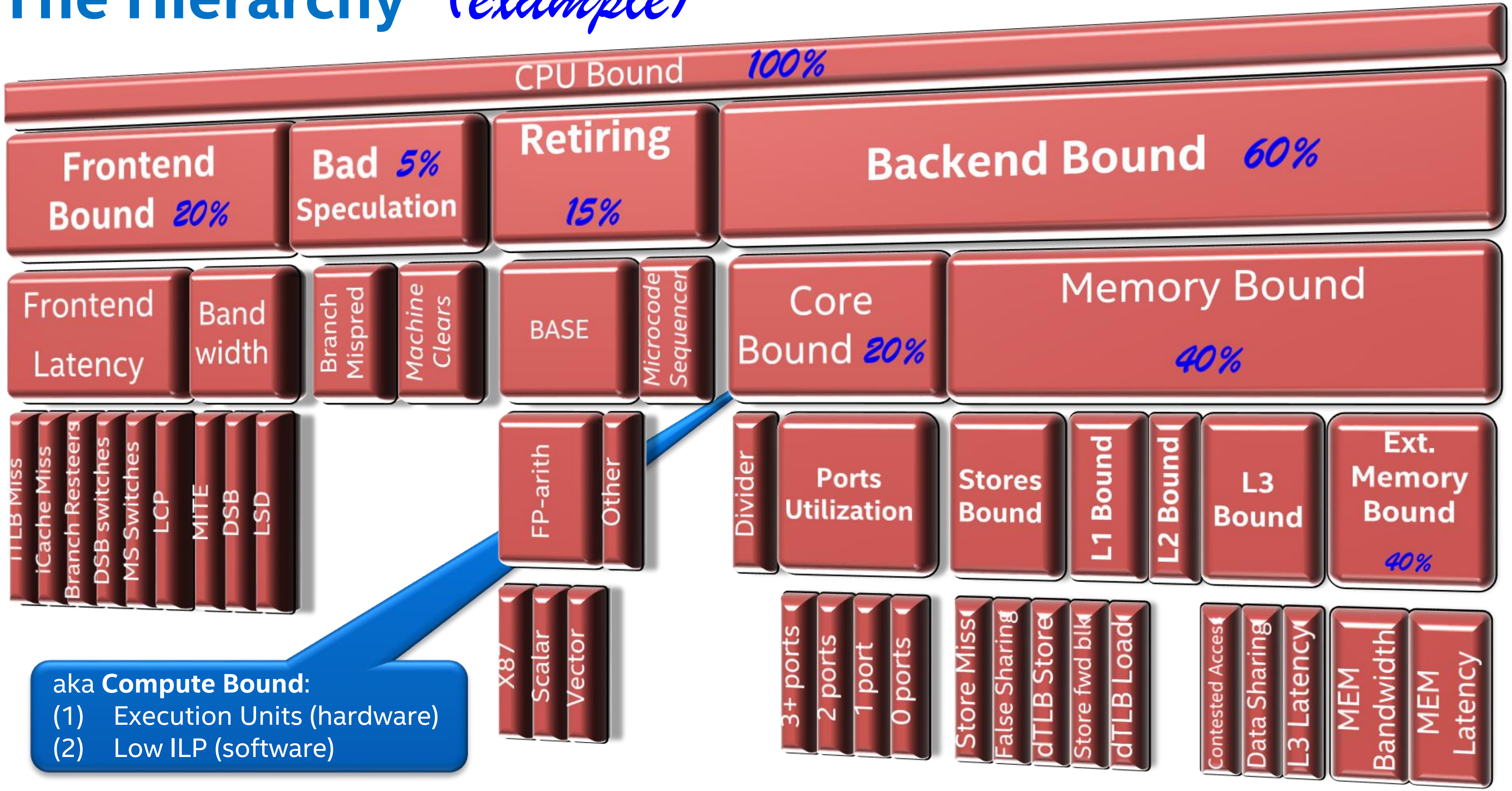
- Addressing Gaps
  - Generic performance metrics abstract the many hazards into categories, using
  - Top-down oriented perf-counters that count:
    - when matters; e.g. stalls vs cycles
    - where matters; e.g. single point of division
    - at finer-grain; e.g. superscalar width.
  - Bad Speculation metric at the top

# 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

## The Hierarchy[1] *(example)*

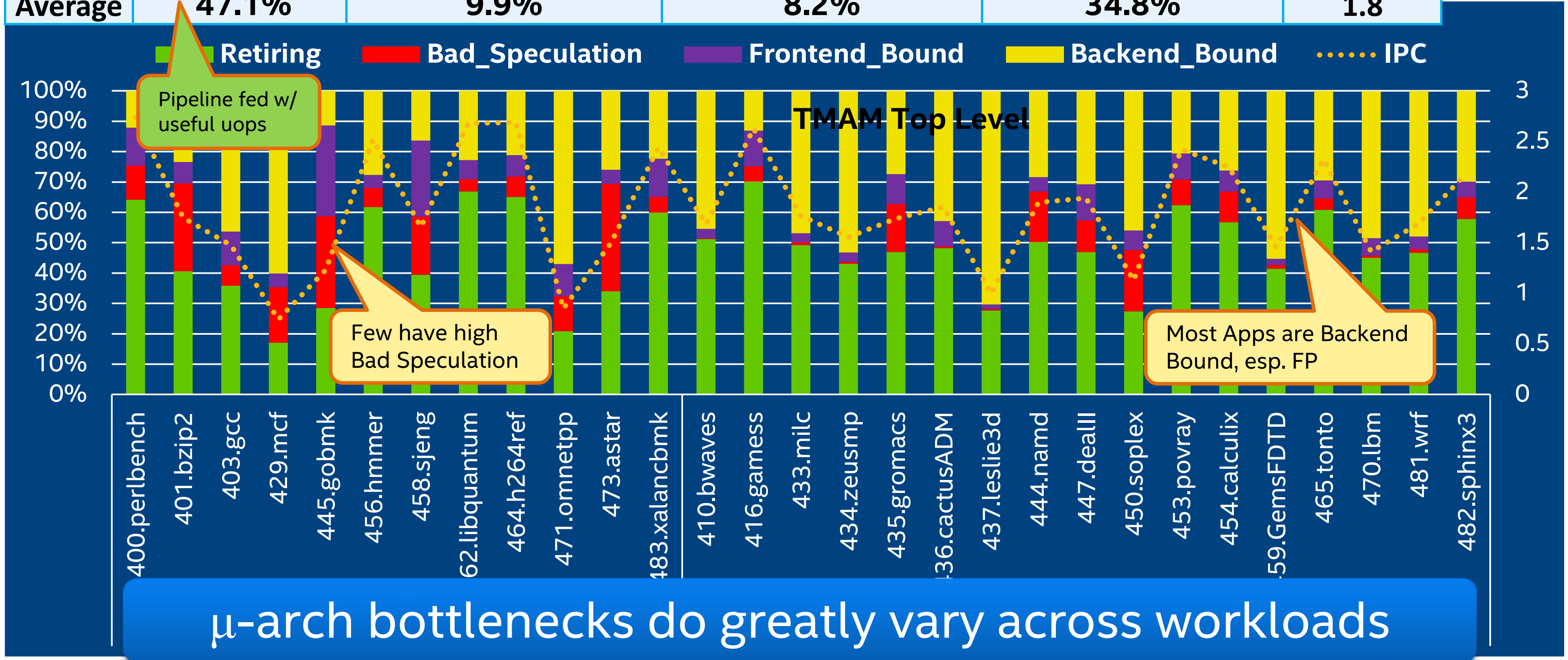


aka **Compute Bound**:
(1) Execution Units (hardware)
(2) Low ILP (software)

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

# Top Level for SPEC CPU2006

| Average | 47.1% | 9.9% | 8.2% | 34.8% | 1.8 |
|---|---|---|---|---|---|

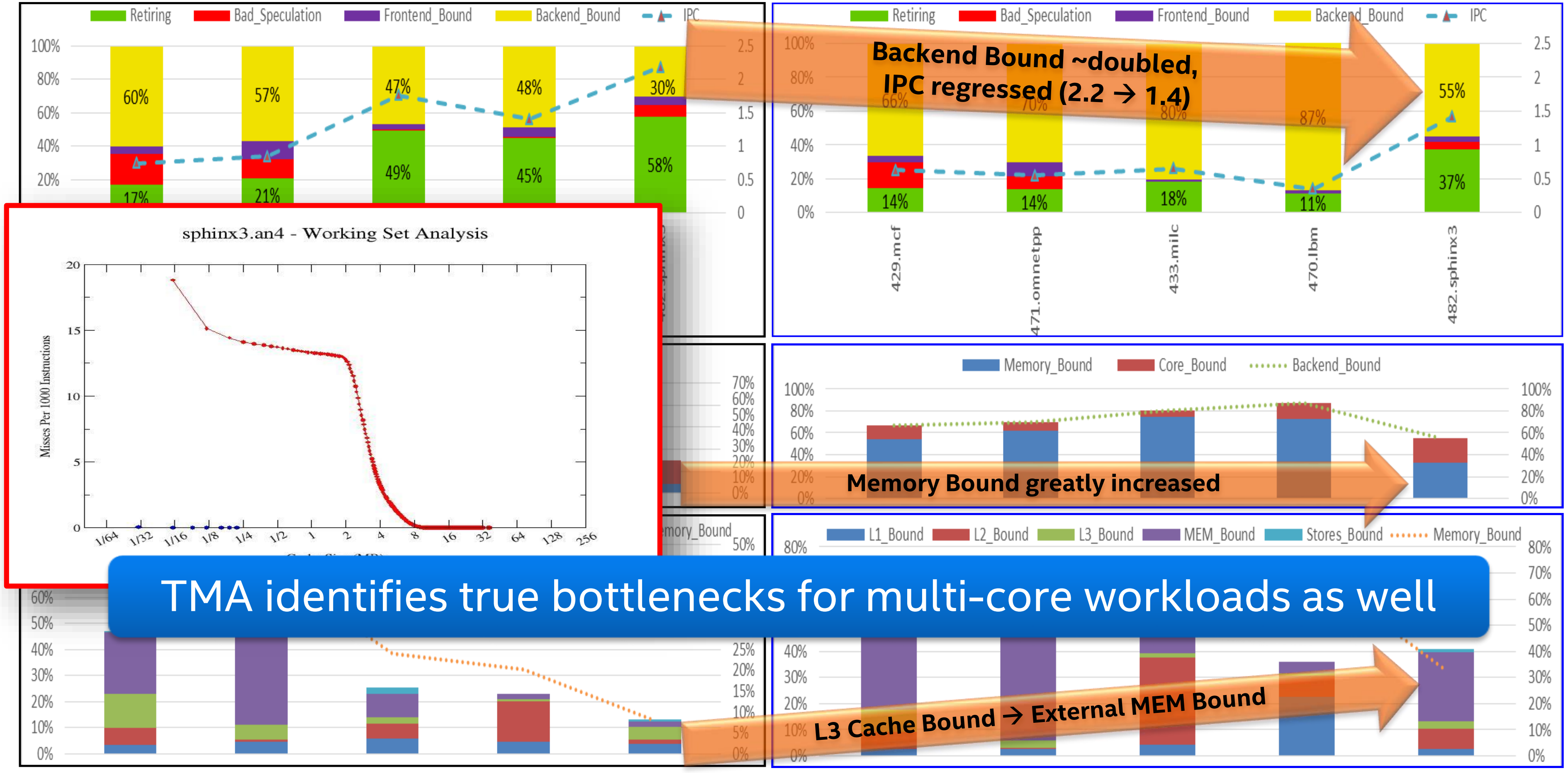


μ-arch bottlenecks do greatly vary across workloads

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

# Memory Bound (1-core vs. 4-cores)

IPC := Instructions Per Cycle



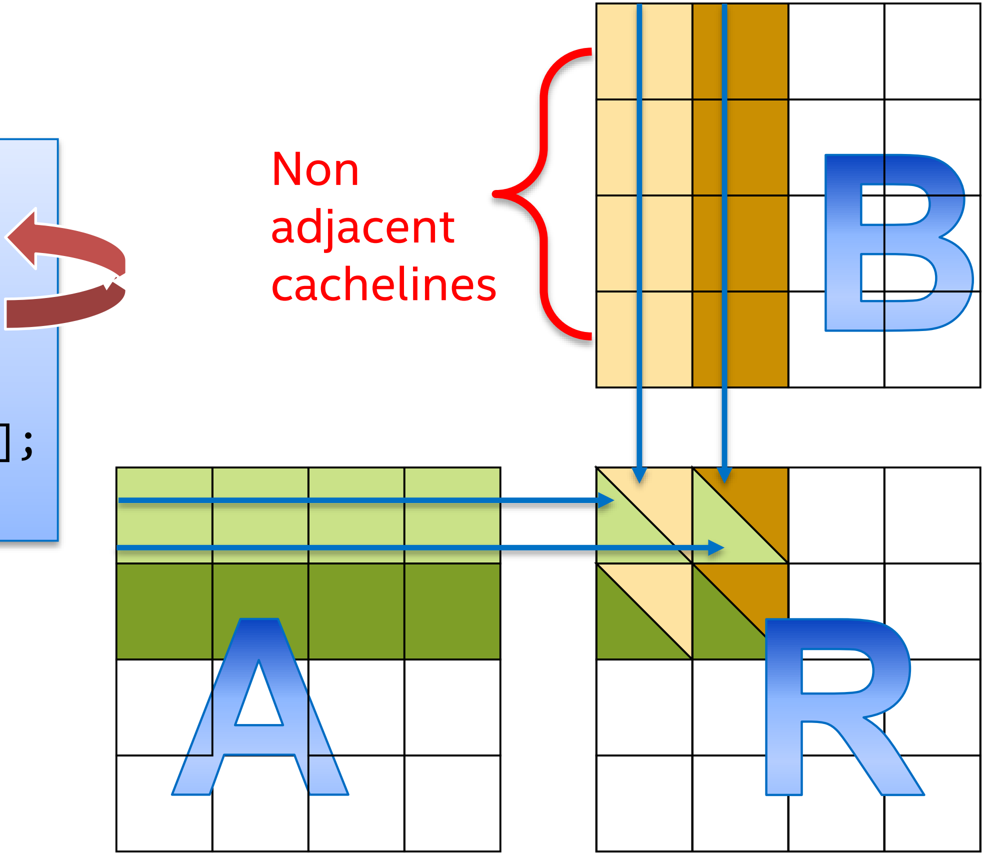
TMA identifies true bottlenecks for multi-core workloads as well

## Performance Optimization example: Matrix Multiply

## **Hands On #2**

# Matrix Multiply insight

```
for (int i=0; i < rows; i++)
  for (int j=0; j < cols; j++)
    for (int k=0 ; k < cols; k++)
      R[i][j] = R[i][j] +
                  A[i][k] * B[k][j];
```



Fix memory access pattern (Loop Interchange optimization)

# Experimental setup

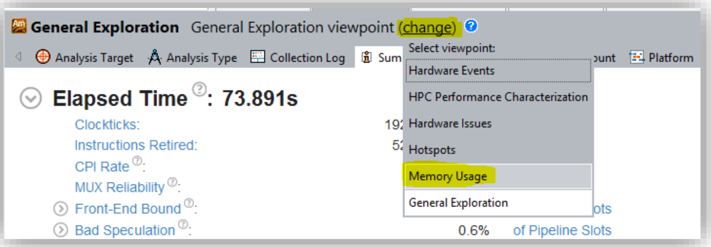
| | | | |
|---|---|---|---|
| HW | **Processor** | **Intel(R) Core(TM) i5-6440HQ CPU @ 2.60GHz** | |
| | | uarch | Skylake |
| | | # Cores | 4 (1 thread/core) |
| | | L3 Cache | 6 MB |
| | | Frequency | Base of 2.6 GHz w/ Turbo Boost (on) up to 3.5 GHz |
| | Memory | Type | DDR4 |
| | | DRAM Frequency | 1067 MHz |
| | | Size | 8 GB |
| SW | **OS** | **Microsoft Windows 10 Enterprise** | |
| | | Version | 10.0.14393 Build 14393 |
| | | Virtualization (Hyper-V) | Disabled |
| | Compiler | Intel® C++ Compiler 16.0 | Version 16.0.1.146 Build 20151021 |
| | | OpenMP Version | 4.0 |
| | **Test Code** | **Matrix Multiply** | |
| | | Data-type | FP Double Precision |
| | | Single matrix size | 32 MB |
| | Tools | VTune | Intel VTune Amplifier 2018 (build 542108) |
| | | TopDown | TMA Metrics version 3.31 |

# Matrix Multiply Optimizations Summary

| 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.
Measured 'Observed Maximum' was 14 [GB/s]. See more on next foil.
(*5) Per 'Average Effective CPU Utilization' line in Effective CPU Usage Histogram
*All Optimization steps are incremental, e.g. Step 6 is on top of Step 5, with the exception that step 4's baseline is step 2.*

# Why CPI increased with OpenMP (Step 6)?

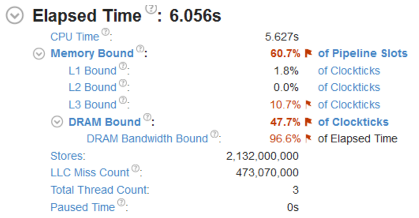

General Exploration General Exploration viewpoint (change)

Analysis Target | Analysis Type | Collection Log | Sum... | ...ount | Platform

Elapsed Time: 73.891s

| | |
|---|---|
| Clockticks: | 19... |
| Instructions Retired: | 5... |
| CPI Rate: | |
| MUX Reliability: | |
| Front-End Bound: | ...ots |
| Bad Speculation: | 0.6% of Pipeline Slots |

## Step 5


Elapsed Time: 6.056s

| | | |
|---|---|---|
| CPU Time: | 5.627s | |
| Memory Bound: | 60.7% | of Pipeline Slots |
| L1 Bound: | 1.8% | of Clockticks |
| L2 Bound: | 0.0% | of Clockticks |
| L3 Bound: | 10.7% | of Clockticks |
| DRAM Bound: | 47.7% | of Clockticks |
| DRAM Bandwidth Bound: | 96.6% | of Elapsed Time |
| Stores: | 2,132,000,000 | |
| LLC Miss Count: | 473,070,000 | |
| Total Thread Count: | 3 | |
| Paused Time: | 0s | |

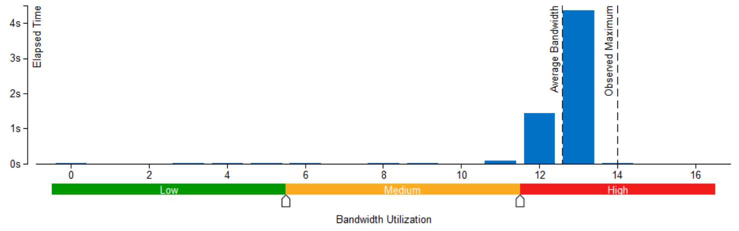
### Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

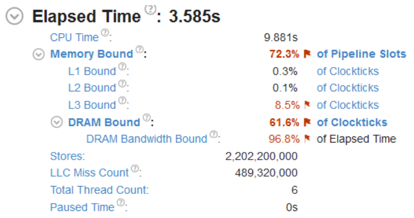
Bandwidth Domain: DRAM, GB/sec

**Bandwidth Utilization Histogram**

This histogram displays the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and Interconnect bandwidth.



## Step 6


Elapsed Time: 3.585s

| | | |
|---|---|---|
| CPU Time: | 9.881s | |
| Memory Bound: | 72.3% | of Pipeline Slots |
| L1 Bound: | 0.3% | of Clockticks |
| L2 Bound: | 0.1% | of Clockticks |
| L3 Bound: | 8.5% | of Clockticks |
| DRAM Bound: | 61.6% | of Clockticks |
| DRAM Bandwidth Bound: | 96.8% | of Elapsed Time |
| Stores: | 2,202,200,000 | |
| LLC Miss Count: | 489,320,000 | |
| Total Thread Count: | 6 | |
| Paused Time: | 0s | |

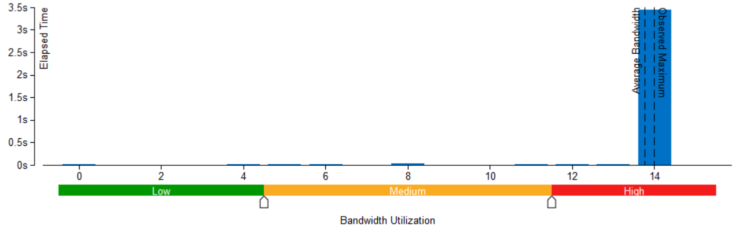
### Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

Bandwidth Domain: DRAM, GB/sec

**Bandwidth Utilization Histogram**

This histogram displays the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and Interconnect bandwidth.



## Optimized code

```
//Step 6: Loop interchange & vectorize & FMA (AVX2) inner loop; parallelize outer loop

#pragma intel optimization_parameter target_arch=CORE-AVX2
void matrix_multiply(int msize, TYPE a[][NUM], TYPE b[][NUM], TYPE c[][NUM]) {
#pragma omp parallel for
    for (int i = 0; i<msize; i++) {
        for (int k = 0; k<msize; k++) {
            #pragma ivdep
            for (int j = 0; j<msize; j++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```

Performance Analysis

## Sample use-cases & Summary

# 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

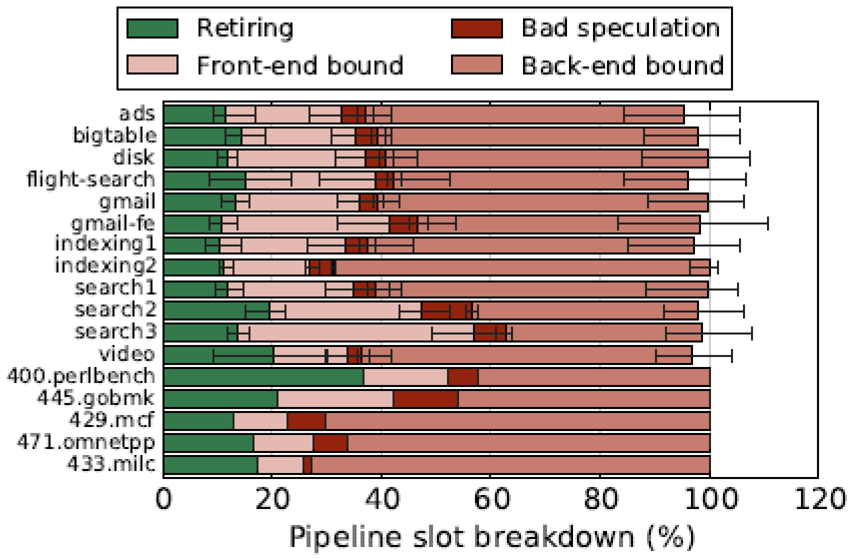
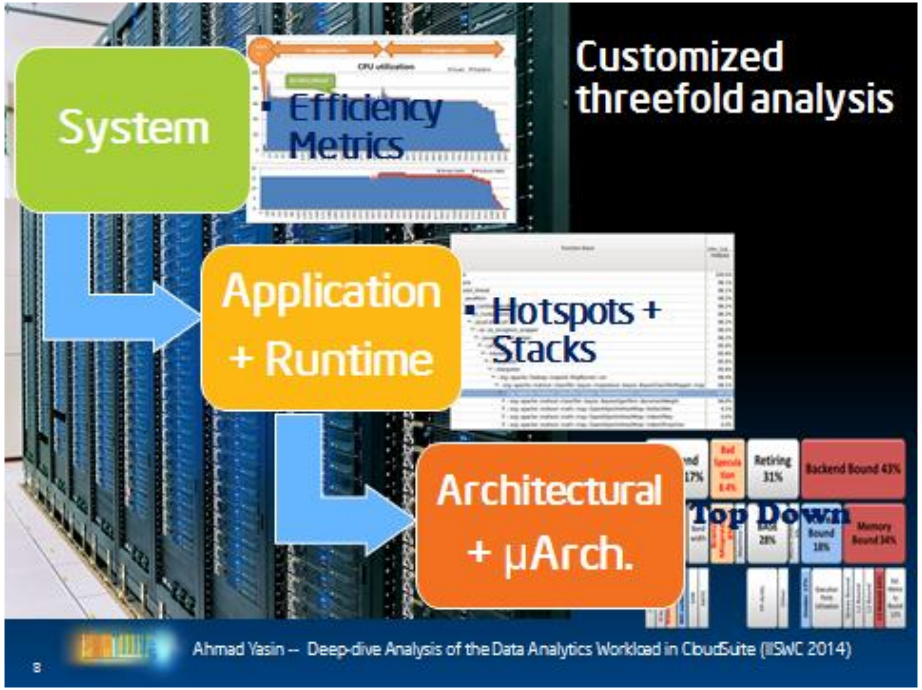


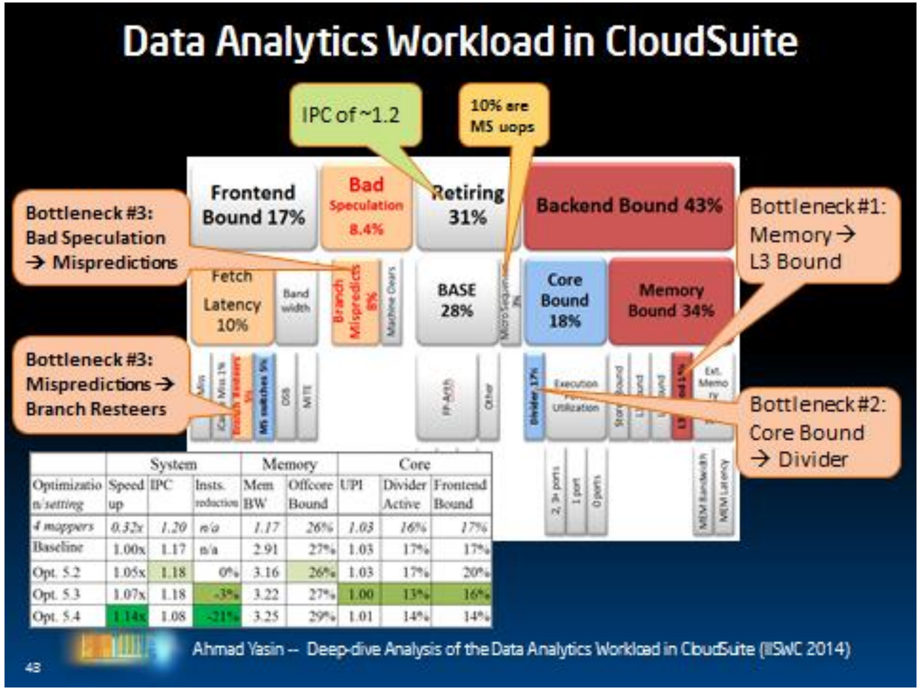
**Figure 4: Top-level bottleneck breakdown. SPEC CPU2006 benchmarks do not exhibit the combination of low retirement rates and high front-end boundedness of WSC ones.**

# 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]


Customized threefold analysis

8 Ahmad Yasin -- Deep-dive Analysis of the Data Analytics Workload in CloudSuite (IISWC 2014)


Data Analytics Workload in CloudSuite

| | System | | | Memory | | Core | | |
|---|---|---|---|---|---|---|---|---|
| Optimization/setting | Speed up | IPC | Insts. reduction | Mem BW | Offcore Bound | UPI | Divider Active | 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% |

43 Ahmad Yasin -- Deep-dive Analysis of the Data Analytics Workload in CloudSuite (IISWC 2014)

## Summary

- Modern multicores utilize sophisticated microarchitectures to increase performance
- Intel Core™ processors offer useful list of performance counters
  - and advanced monitoring capabilities.
- Tools can provide insights on the actual execution to enable software developers to optimize their applications and thus further increase performance
- VTune lumps together key profiling techniques – for free for students
- Top-down Microarchitectural Analysis (TMA) simplifies performance analysis and eliminates the “guess work”

## Useful pointers

- Free Intel tools for students, including VTune:
  >>> https://software.intel.com/en-us/qualify-for-free-software/student
- PMU events Documentations
  - https://software.intel.com/en-us/vtune-amplifier-help-intel-processor-events-reference
  - PerfMon Events – electronic files (.json, .csv etc) for tools
  - Intel® 64 and IA32 Architectures Performance Monitoring Events – for humans
- 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]
  - Session for programmers at Intel Developer Forum
    - 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] http://myeventagenda.com/sessions/0B9F4191-1C29-408A-8B61-65D7520025A8/7/5#sessionID=338
  - TMA-metric files: https://download.01.org/perfmon/
  - **toplev**: open source tool in Linux by Andi Kleen: https://github.com/andikleen/pmu-tools/wiki/toplev-manual

## Backup

# Matrix Multiply Summary [IDF'15 version]

A function is *iteratively* analyzed

1 Big matrices in memory

- `multiply1` is **Ext. Memory Bound**

2 Loop Interchange

- `multiply2` becomes **Core Bound** due to execution ports high utilization

3 Vectorization

- `multiply3` reduces ports pressure but workload is still **Core Bound** and **Memory Bound**

4 FMA – Fused Multiply-Add

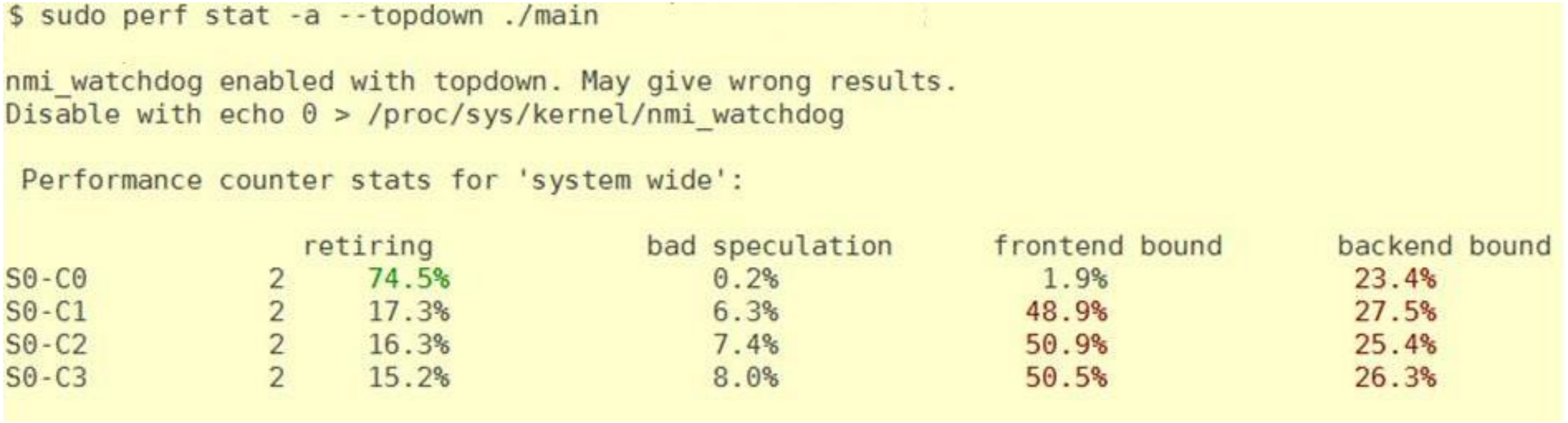
- `multiply4` further reduces instruction count with better latency/throughput in Skylake. Workload back to **Memory Bound**

| step | Optimization | Time [s] | speedup |
|---|---|---|---|
| 1 | Textbook (baseline) | 41.3 | 1.0x |
| 2 | Loop Interchange | 5.1 | 8.1x |
| 3 | +Vectorization | 4.1 | 10.1x |
| 4 | +FMA | 2.8 | 14.7x |

**Performance bottlenecks vary as the code is optimized**

## Support in Linux

- Linux kernel supports TopDown Level-1 metrics
  - Since Linux kernel 4.8
  - For shipping Core and Atom products!
  - Simply do: `perf stat -a --topdown <user-app>`



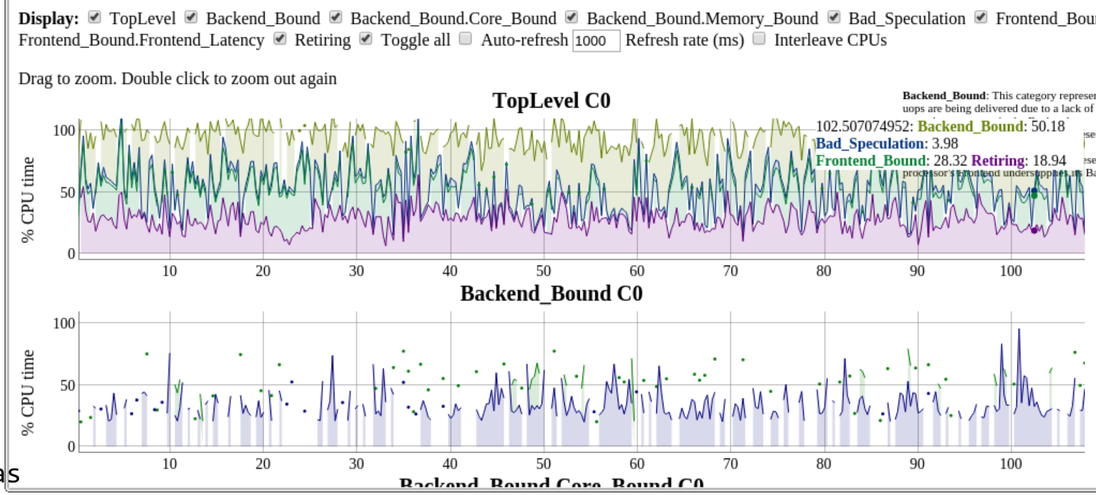
```
$ sudo perf stat -a --topdown ./main

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%
```

# pmu-tools/toplev



**Display:** ☑ TopLevel ☑ Backend_Bound ☑ Backend_Bound.Core_Bound ☑ Backend_Bound.Memory_Bound ☑ Bad_Speculation ☑ Frontend_Bou
Frontend_Bound.Frontend_Latency ☑ Retiring ☑ Toggle all ☐ Auto-refresh 1000 Refresh rate (ms) ☐ Interleave CPUs

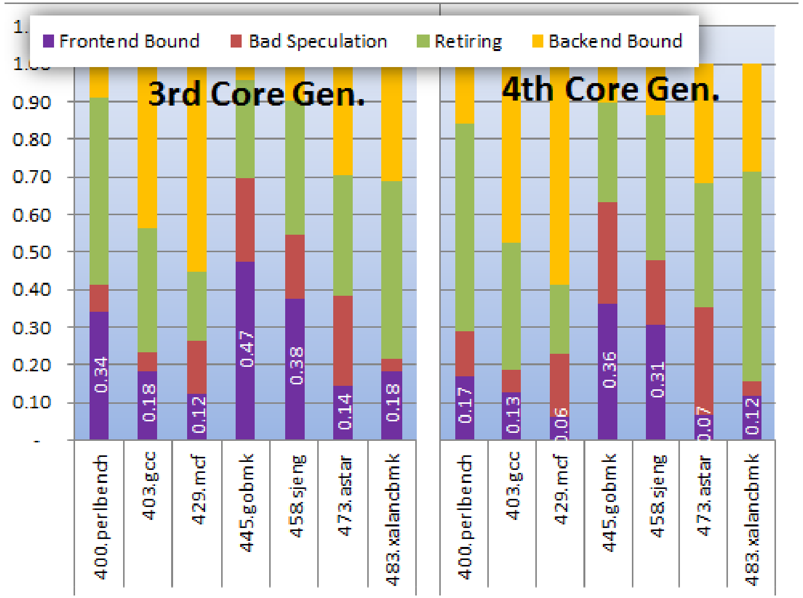
Drag to zoom. Double click to zoom out again

102.507074952: Backend_Bound: 50.18
Bad_Speculation: 3.98
Frontend_Bound: 28.32 Retiring: 18.94

```
% toplev.py -l3 --single-thread ./c-asm  numbers
BE       Backend_Bound:                           64.2%
BE/Mem   Backend_Bound.Memory_Bound:            ...
BE/Mem   Backend_Bound.Memory_Bound.L1_Bound:   49.3%
              This metric represents how often CPU was
              stalled without missing the L1 data cache...
              Sampling events:  mem_load_uops_retired.l1_hit:pp,mem_load_uops_retired.hit_lfb:pp
BE/Mem   Backend_Bound.Memory_Bound.L3_Bound:   48.7%
              This metric represents how often CPU was stalled on L3 cache
              or contended with a sibling Core...
              Sampling events:  mem_load_uops_retired.l3_hit:pp
BE/Core Backend_Bound.Core_Bound:                28.3%
BE/Core Backend_Bound.Core_Bound.Ports_Utilization: 28.3%
              This metric represents cycles fraction application was
              stalled due to Core computation issues (non divider-related)...
```

# Abstracted Metrics

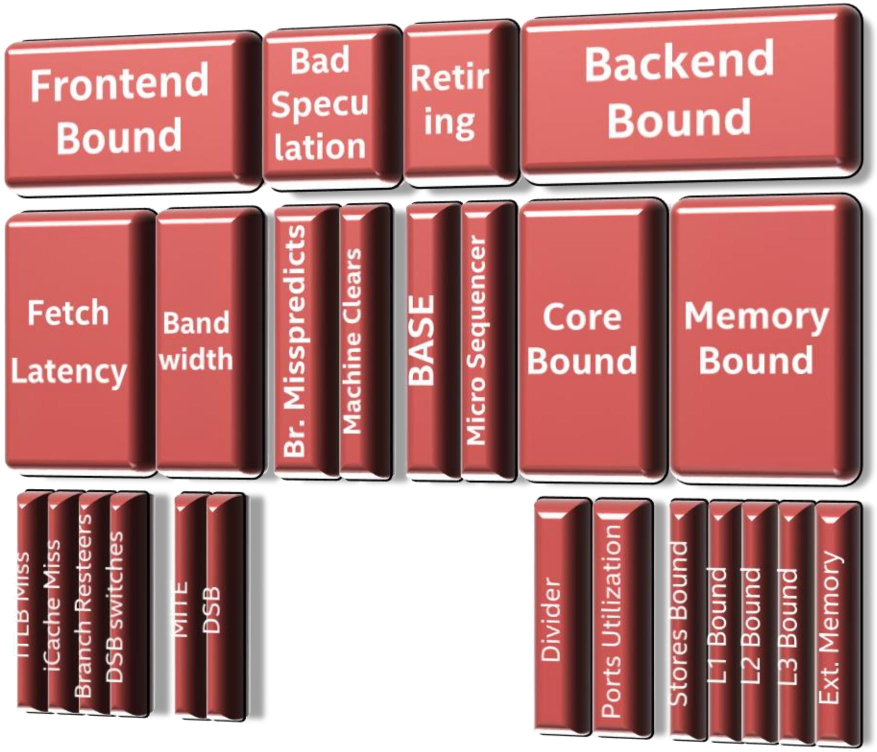
- **Frontend Bound** is an Abstracted Metric
  - High-level category of all front-end bottlenecks
- Abstracted Metrics are better for performance analysis
  - Think **IPC** (Instructions Per Cycle) not **MPKI** (Miss Per Kilo Instructions)
- Enable evaluations across
  - µarch generations
    - Haswell (4th generation Intel® Core™ processor) has improved front-end through speculative i-TLB and i-cache accesses with better timing to improve the benefits of prefetching
    - Benefiting benchmarks show clear Frontend Bound reduction
  - µarch families – e.g. Intel Core™ vs Atom™
  - Architectures – e.g. X86 vs. ARM



**Abstracted Metrics Enable cross-processors Comparisons**

# IDF'15 Summary foil

- TMA: Top-down Microarchitectural Analysis method
  - Simplifies Performance Analysis using a structured hierarchy
    - Eliminates guess work
    - Reduces microarchitecture high learning curve
  - Forward compatibility, consistency across IA processors


Frontend Bound
Bad Speculation
Retiring
Backend Bound
Fetch Latency
Bandwidth
Br. Misspredicts
Machine Clears
BASE
Micro Sequencer
Core Bound
Memory Bound
ITLB Miss
iCache Miss
Branch Resteers
DSB switches
MITE
DSB
Divider
Ports Utilization
Stores Bound
L1 Bound
L2 Bound
L3 Bound
Ext. Memory