Analyzing data latency access

William Jalby, Vincent Palomares**, Alexandre Vardoshvili, Emmanuel Oseret

University of Versailles Saint Quentin en Yvelines/ECR

**Now with INTEL

Scalable Tools Workshop 2017

DATA ACCESS LATENCY (1)

- Data access latency can have a major impact on performance (more data on that topic in the rest of the talk)

- Several hardware mechanisms have been designed to minimize latency impact (latency hiding):
  - Various buffers: Reservation Station, Reorder Buffer, Load/Store Buffer, Line Fill Buffer
  - Hardware prefetchers: on Xeon, 4 prefetchers per core: topic is serious ☺
  - Various memory levels and types (Off chip versus on Chip DRAM on KNL): useful not only for latency but also bandwidth.
  - Special PMU PEBS for tracking load latency…

DATA ACCESS LATENCY (2)

- Difficult to exactly associate with every load its latency
  - Exactly is of major importance because you want to identify the array access which is guilty
  - Some PMU events can provide such info.
- Even more difficult to associate with each load, its latency impact on the overall loop performance.
  - It is one thing to know that you are sick but it is better to know how serious it is and even better to get the right prescriptions
  - Accelerate the exploration of the different prefetching configurations
  - PMU events have hard time to provide such info.

Optimizing Data Access Latency

- Various optimization mechanisms:
  - Array restructuring, blocking, loop interchange
  - Allocating to different memory levels: On chip DRAM versus Off chip DRAM
  - Use software instructions to perform aggressive prefetching, more aggressive than hardware due to a better knowledge on loop bounds
- BIG ISSUE: how to drive use of these techniques, what is benefit of blocking, prefetching, etc. ???
  - PMU do not provide such info on potential gain.
- Many simple performance models are only dealing with bandwidth: simplistic with roofline method, more realistic by taking into account different functional units bandwidths and even better different cache levels.

LITTLE'S LAW (1)

- Interesting enough, bandwidth and latency are related through buffer usage, more precisely Little's Law
- Classical formulation: $L = \lambda W$
  - L : Average Number of customers
  - W : Average waiting time
  - λ : Average arrival rate

LITTLE'S LAW (2)

➢First use of L = λ W : model off chip cache line access
- L : Average Number of in flight cache line requests (smaller than number of LFB slots: 10!!)
- W : Average waiting time (in cycles) in LFB (close to Latency)
- λ : Average number of cache lines entering per cycle the LFB (close to Bandwidth)

This formula clearly shows that Bandwidth is a function of latency and max number of LFB slots. Clearly constant bandwidth (as assumed by Roof Line model) is a myth ☺

➢Second use: more complex: model FU within CPU.

UFS: Uop Flow Simulation

- Ignore semantics: code not executed: use instruction traces. Very simple for loops without branches in loop body.
- Off line analysis/simulation
- Works directly with asm/binary
- Cycle level simulation
- Cycle accurate simulation of core pipeline according to public/published information. If no information is available, use best fit algorithm
- Parametrizable inputs: buffer sizes, instruction latencies/back to back rates, number of ports, bandwidths……

Hardware parameters: two main options

1. Use published numbers: issues with quality and user versus system view
2. Design specific benchmark to measure the requested info.

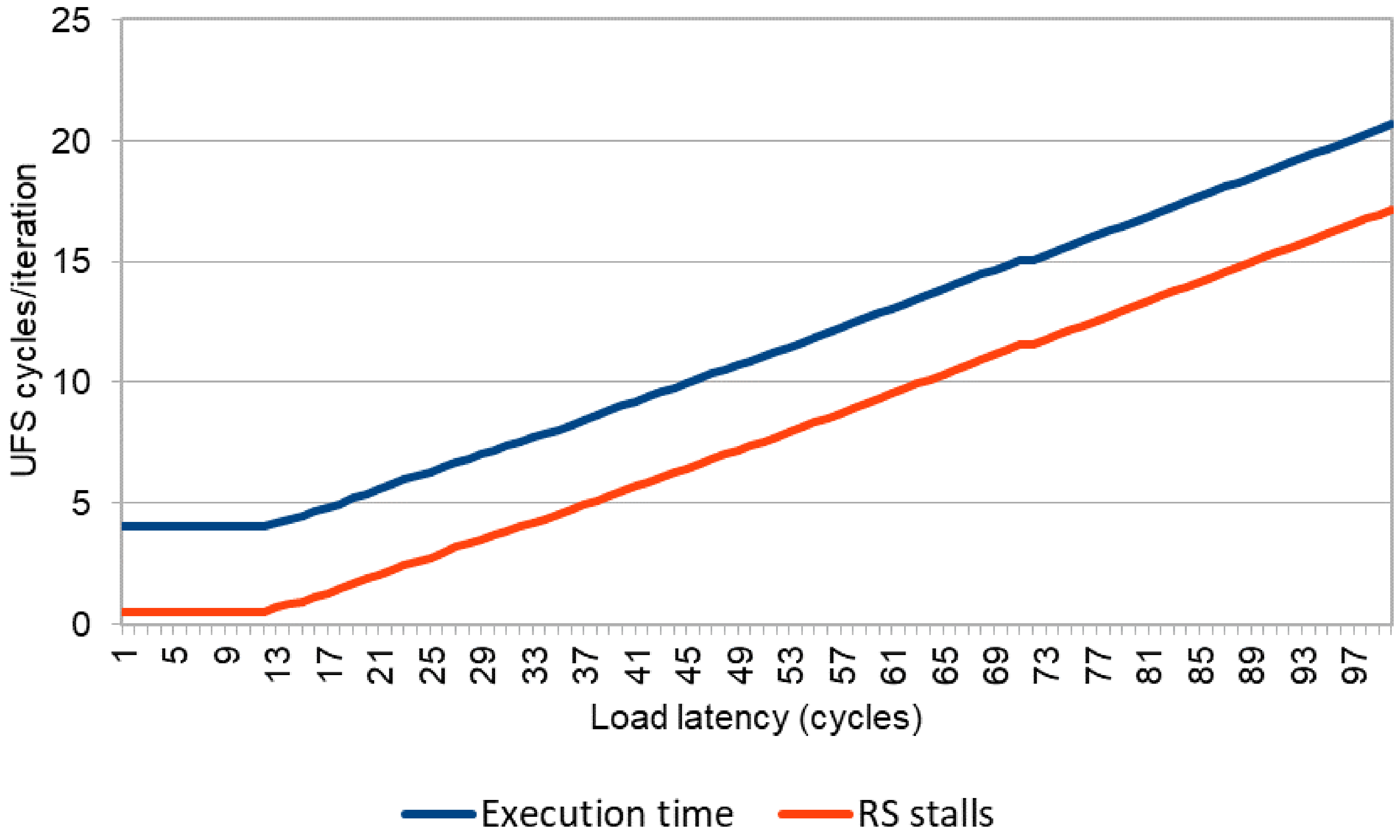
Our approach is with specific benchmarks.

Why bother with UFS: Yet Another Simulator

- Much faster than most of standard simulators (but does much less)
- Due to its speed, ability to perform massive parameter studies (Sensitivity Analysis)
- Globally good accuracy most of the time, within 10% of real measurements

- Much better than counters for understanding real issues: the difficulty is that buffer saturation is not the source of the problem but in general the consequence of a problem…. Excessive load latencies will quickly lead to Load Buffer and/or RS overflow. However, well controlled latencies will avoid such overflow and lead to better performance.
- ONE MAJOR ISSUE: how to optimize code to minimize RS, ROB etc…footprint/consumption

Impact of Latency: Balanc_3_de



BALANC3 : A(I) = A(I) * CST

Characterizing codelet response to latency variations

On most of the latency sensitivity plots (see previous slides), two regions can be clearly seen:

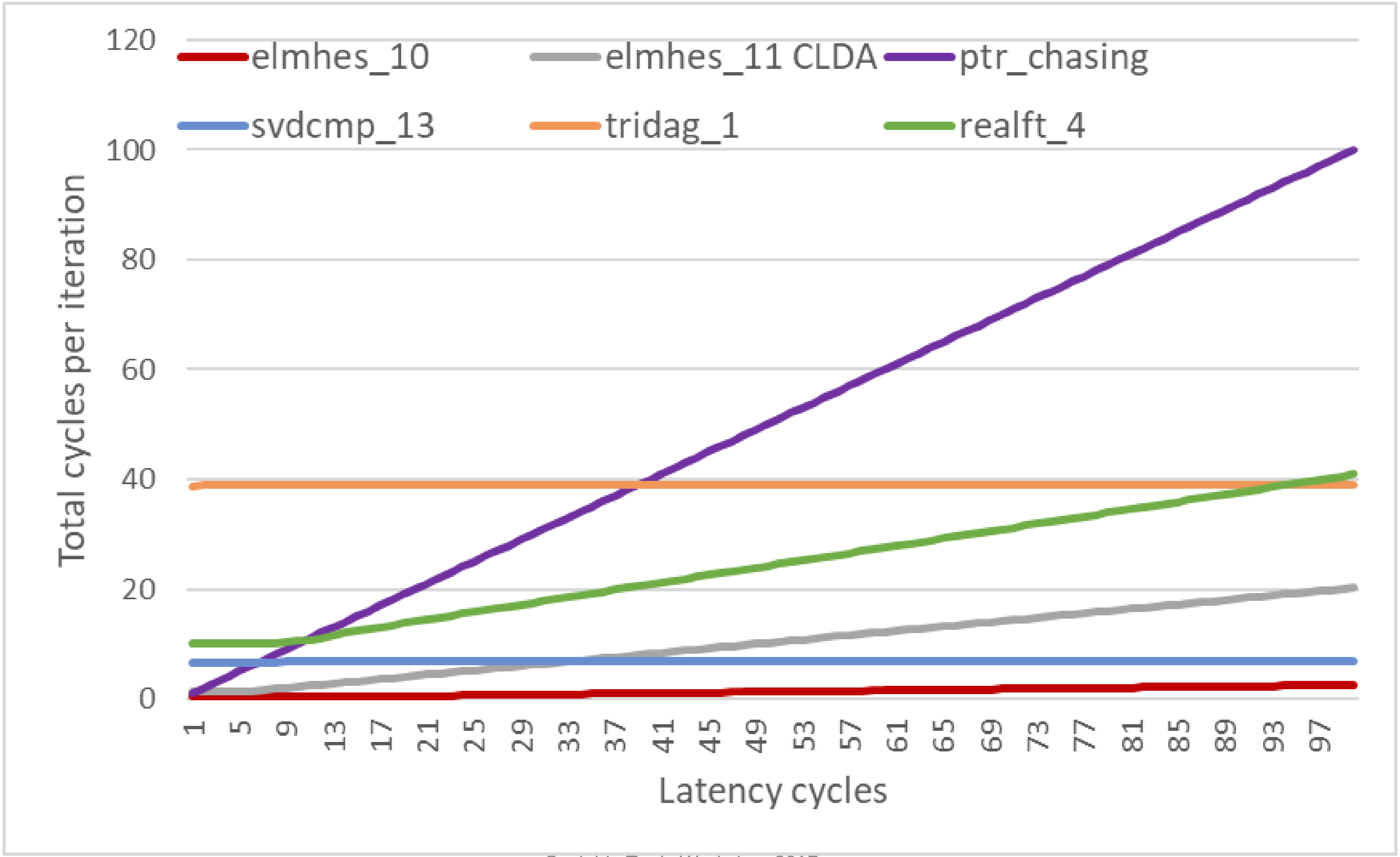
- Initial flat region where codelet is supporting higher latencies without any impact on performance (buffers filling up).
- Linear region where impact on performance varies linearly with latency increase (one of the buffer is full).

Two key parameters for characterizing a codelet:

- Lmax: maximum latency tolerated with 0 or negligible cost
- Additional cost: slope of the linear line

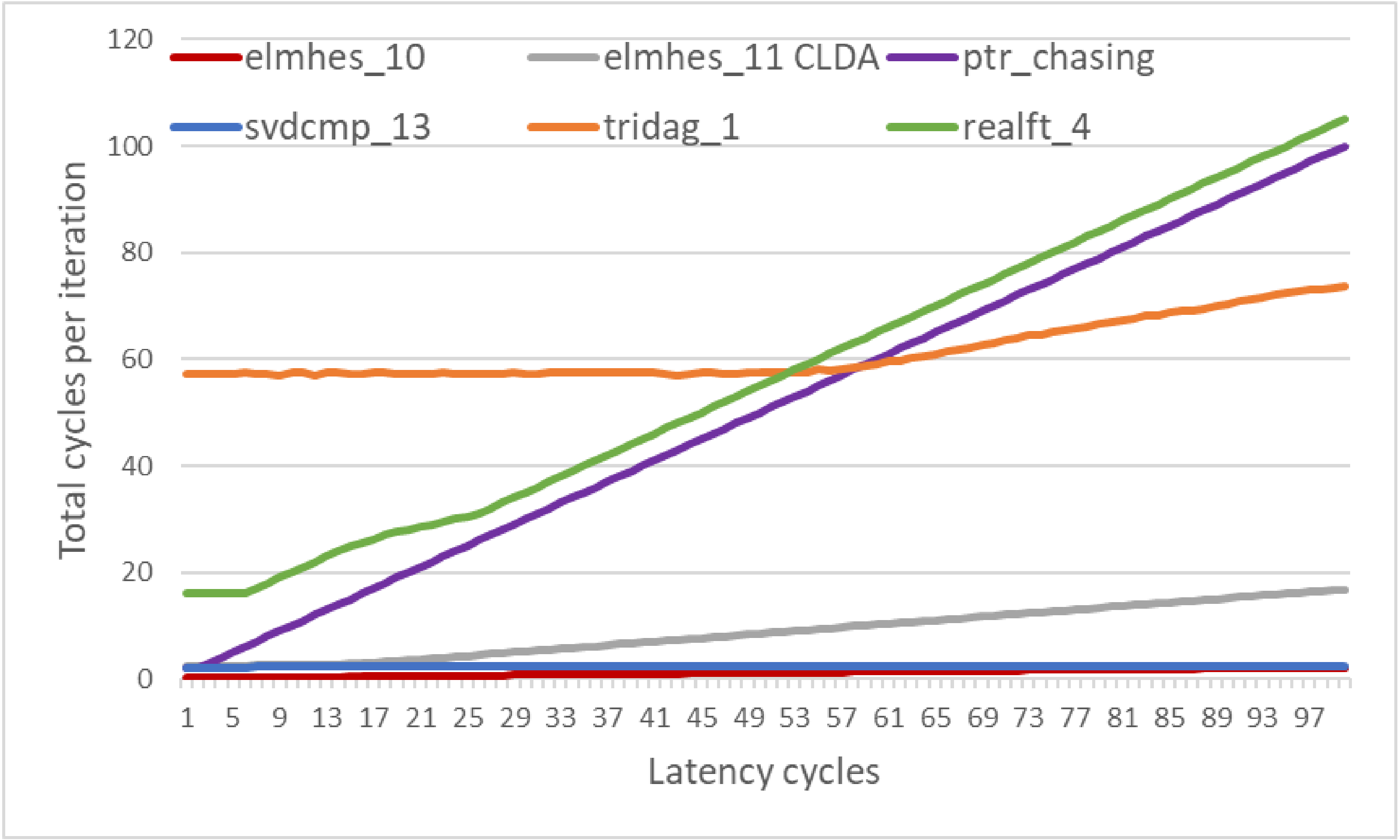
A few Numerical Recipes loops on Haswell

- For Haswell



Comparison for NR on KNL

- For KNL



LATENCY SENSITIVITY ANALYSIS

UFS allows to model the impact of varying latency: this can be done uniformly on loads and stores or individually.

This allows to understand the potential performance gain of:

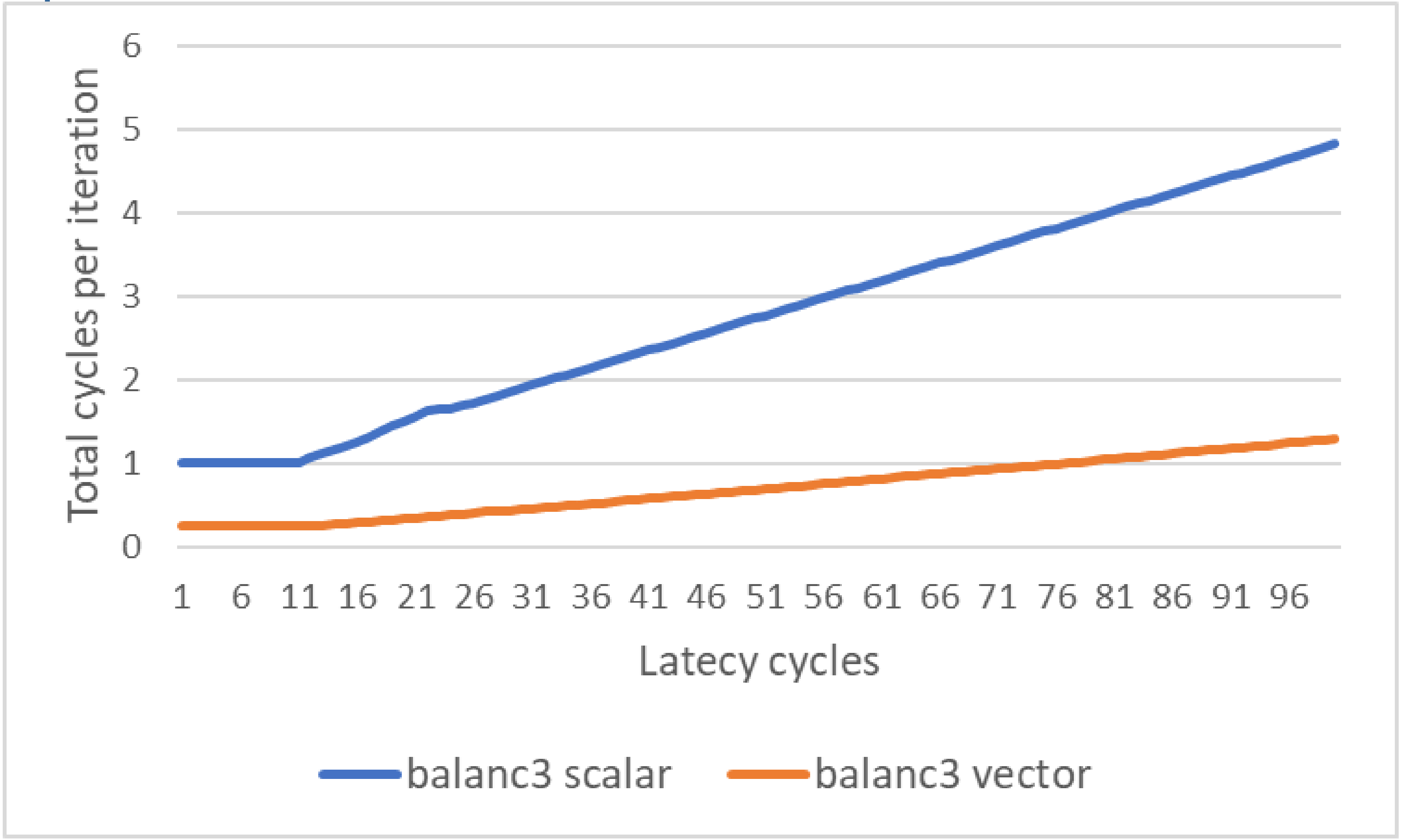
1. Better blocking (blocking for L2 instead of L3)
2. Better prefetching (add extra prefetch instructions on targeted loads)
3. Using on die DRAM versus external DRAM (cf. KNL)

Conclusions

- Various Out of Order buffers (ROB, RS, PRF, LB, SB ….) are critical to get peak performance on modern cores.
- UFS gives a detailed insight on buffer usage and can correlate usage with code.
- UFS is fast, allowing massive parameter explorations.
- UFS is excellent at exploring what if scenarii (hardware and software)
- UFS allows to characterize latency impact on loop performance.

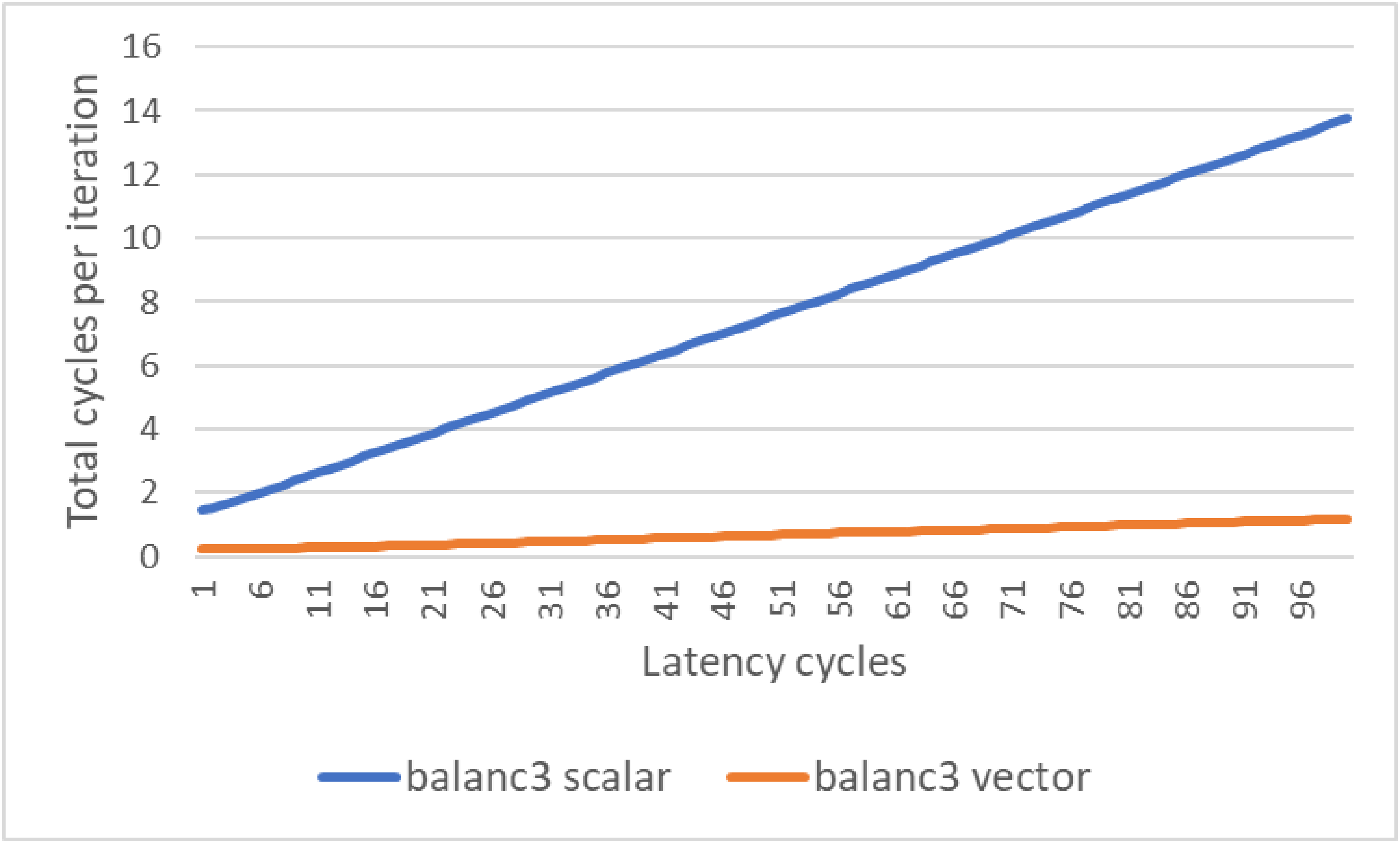
BACKUP SLIDES

Scalar versus Vector: Haswell



BALANC3 : A(I) = A(I) * CST

KNL


Total cycles per iteration
0
2
4
6
8
10
12
14
16
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
Latency cycles
balanc3 scalar
balanc3 vector

A first simple performance model (CQA)

CQA: Code Quality Analyzer Open Source: www. maqao.org

STATIC MODEL: all operands are assumed resident in L1.

Compute 3 bounds:

- Issue/Decode : divide number of uops per 4 + ceiling effect
- Execution: count number of instructions per port/FU (taking into account rate)
- Inter iterations dependencies: compute cycles

Predicted number of cycles = max of the 3 estimates above.

THROUGHPUT/BANDWIDTH BASED MODEL

CQA Output

Analyze ASM:

12 FP Mul instructions, 16 FP Add/Sub instructions, 4 Loads Instructions + 4 Address computations, 4 Store Instructions, 18 Alu Instructions

Compute the 3 bounds:

1. Issue/Decode : 58 uops after unlamination.: 14,5 cycles rounded to 15 cycles.
2. Execution: P0 (FP */ALU): 15 cycles, P1 (FP+/ALU): 16 cycles, P2 (Load): 4 cycles, P3 (Load): 4 cycles, P4 (Store): 4 cycles, P5 (Misc/ALU): 15 cycles
3. Inter iterations dependencies: 13 cycles

Predicted number of cycles = max of the 3 estimates = 16 cycles

CQA versus Measurements in L1

CQA prediction: 16 cycles

Measurement: 23,36 cycles

GAP: 23,36-16 = 7,36 cycles.

BEYOND L1: results slightly worse but to be expected

What happens??

A first answer provided by hardware events: each of the buffer has an associated event counting the number of cycles where when full it causes the front end to stall.

Measurement: Reservation Station Stalls occur for 7,85 cycles….

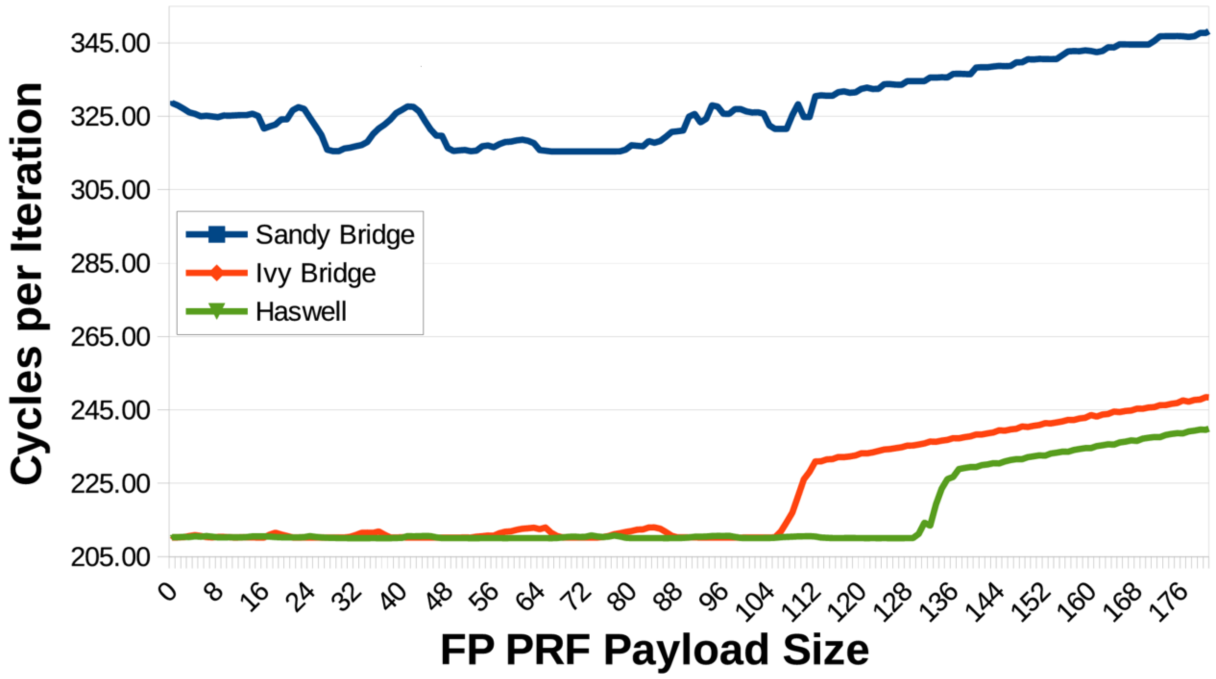
ISSUES: a stall at the front end does not necessarily result in cycles lost in the back end, multiple counting (several buffers full)

UFS results:

23,01 (using SNB buffer sizes): perfect match with measurements and points to RS full leading to wasted cycles

19,03 (using large buffers): time lost in dispatch

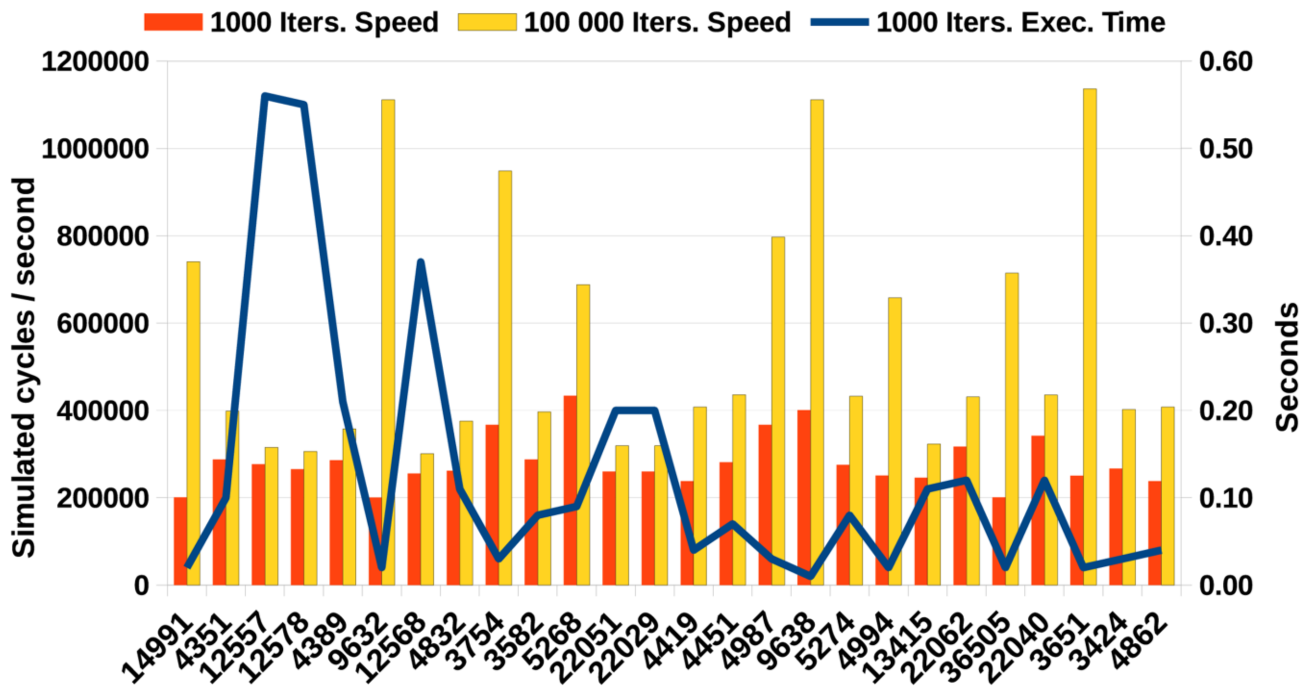
FP PRF Resource Quantification



PRINCIPLE: increase the payload to force an overflow in the target resource which in turn translates into a discontinuity in timing.

"payload" is basically some extra instructions specifically designed to run COMPLETELY in parallel with divisions *UNLESS* they saturate the target buffer.

YALES2 Speed Validation


1000 Iters. Speed
100 000 Iters. Speed
1000 Iters. Exec. Time
Simulated cycles / second
Seconds