

Performance Modeling

by: Ahmed Hassan



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

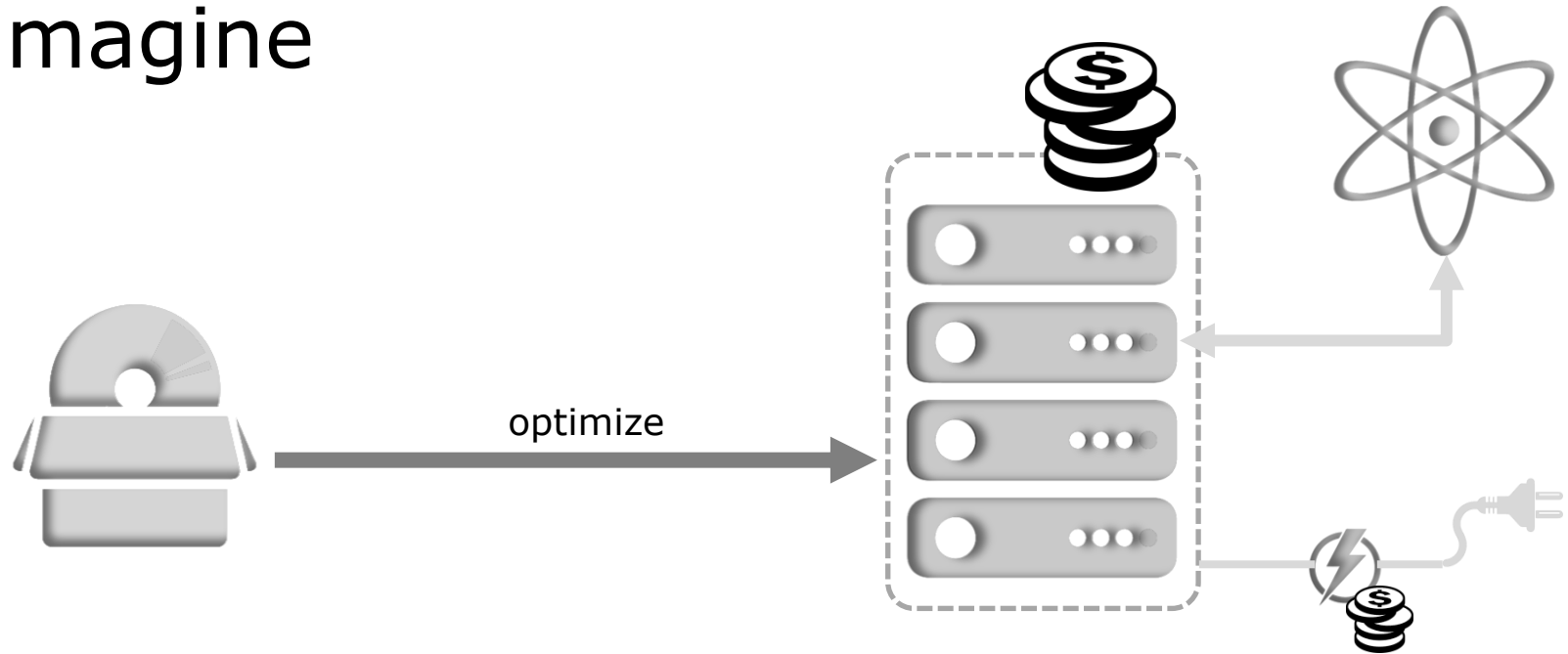
All brand names, product names and titles and copyrights used in this presentation are trademarks or trade names or copyrights of their respective holders



Agenda

- What is Performance Modeling?
 - Why Performance Modeling?
 - Prospective of Performance Modeling
 - Different Performance Models
 - Components of Performance
 - Roofline Model
 - Execution-Cache-Memory (ECM)
 - Roofline in HPC Overview
 - Software Optimization
 - Application/Service Modeling
- Introduction**
- Roofline Model**
- ECM**
- HPC**
- Software/Service Optimization**

imagine



you are to optimize applications to run on a multi-hundred-million dollar supercomputer that consumes as much energy as a small European town to solve computational problems at an international scale and advance science to the next level with “hero-runs” of [inset verb here] scientific applications that cost \$10 K and more per run... [1]

then you better plan head

Trying to extrapolate the performance from small machines to big machines.

[1] National Center for Supercomputing Applications (NCSA), University of Illinois, USA, "Performance Modeling for Systematic Performance Tuning", ACM 978-1-4503-0771-0/11/11, pp. 1 - 5.

What is Performance Modeling?

Performance modeling is a structured and repeatable approach by defining an abstract architectural model.

Why Performance Modeling?

- Guide optimization during application design.
- Evaluate tradeoffs before building the solution.
- Guiding future maintenance and expansion decisions.
- Avoid performance surprises during application execution.
- Provide a document of itemized scenarios for tracking performance goals.

Prospective of Performance Modeling



What?

System/architectural

More focus on hardware optimization

Kernel (Application/Services)

More focus on software optimization

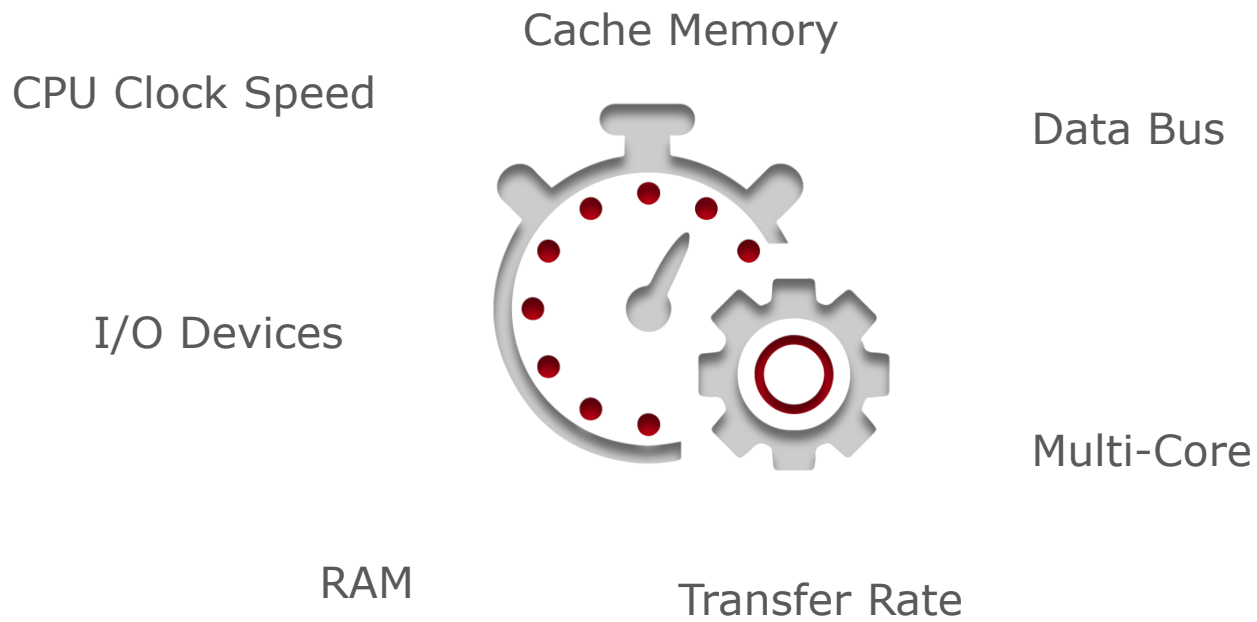


Who?

Data Centers, System Administrator

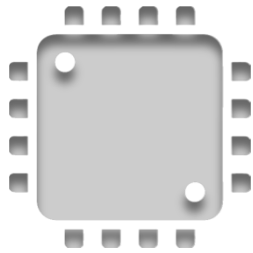
User

Factors Affecting System Performance

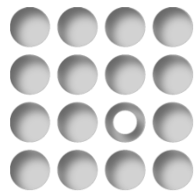


Principle Components of Performance^[2]

- Computation
- Communication
- Locality



Each architecture has a different balance between those components



Each kernel has a different balance between those components^[3]

Performance is a question of how well a kernel's characteristics map to architecture's characteristics

[2] S. Jarvis, S. Wright, Simon Hammond, High Performance Computing Systems: Performance Modeling, Benchmarking and Simulation, Springer, ISBN: 978-3-319-10213-9, pp. 19 - 26.
[3] Heike McCraw, Innovative Computing Laboratory, Department of Electrical Engineering and Computer Science, "Performance Modeling", University of Tennessee, 2013, pp. 7 - 9.

① Computation^[4]

Floating point performance (Gflop/s) is considered to be the main interest

Peak in-core performance can be achieved when:

- fully attainment ILP, DLP, FMA;
- non-FP instructions don't deplete instruction Bandwidth;
- branch mis-predictions are not often;
- threads converge.

In-core parallelism is achieved when:

- Algorithm implements Inheritance;
- The generated code has explicit.

[4] Samuel Williams, David Patterson, ParLab Summer Retreat, "The Roofline Model: A pedagogical tool for program analysis and optimization", Berkeley Par Lab

② Communication^[4]

DRAM bandwidth (GB/s) is the main interest

Peak bandwidth can be achieved when specific optimizations are implemented:

- SW Prefetching;
- NUMA allocation;
- NUMA usage;
- Memory coalescing;
- Few unit stride streams.

[4] Samuel Williams, David Patterson, ParLab Summer Retreat, "The Roofline Model: A pedagogical tool for program analysis and optimization", Berkeley Par Lab

③ Locality^[4]

Traffic: is the volume of data to/from memory
It is not # of loads and stores

To reduce communication we have to increase the locality but there is still what so called “Compulsory Traffic” which is the minimum needed amount of communication.

Hardware modification helps reducing communication through:

- More cache associativities;
- Increasing non-allocating caches;
- Reduce capacity misses by increase cache capacities.

Software optimization helps reducing communication through:

- Avoid capacity misses through blocking;
- Avoiding conflict misses through Padding;
- Increasing non-allocating stores.

[4] Samuel Williams, David Patterson, ParLab Summer Retreat, "The Roofline Model: A pedagogical tool for program analysis and optimization", Berkeley Par Lab

Integrating Performance Components



Goal

Integrating ...

in-core performance + **memory bandwidth** + **locality**

into a single understandable performance figure

Coordinates of a kernel are quasi unique to each architecture
→ Roofline model will be unique to each architecture

Must graphically show the penalty associated with not including certain software optimizations.

How GFlop/s relates to GB/s?

Flops:Bytes is the parameter that allows us to convert bandwidth (GB/s) to performance (GFlop/s).

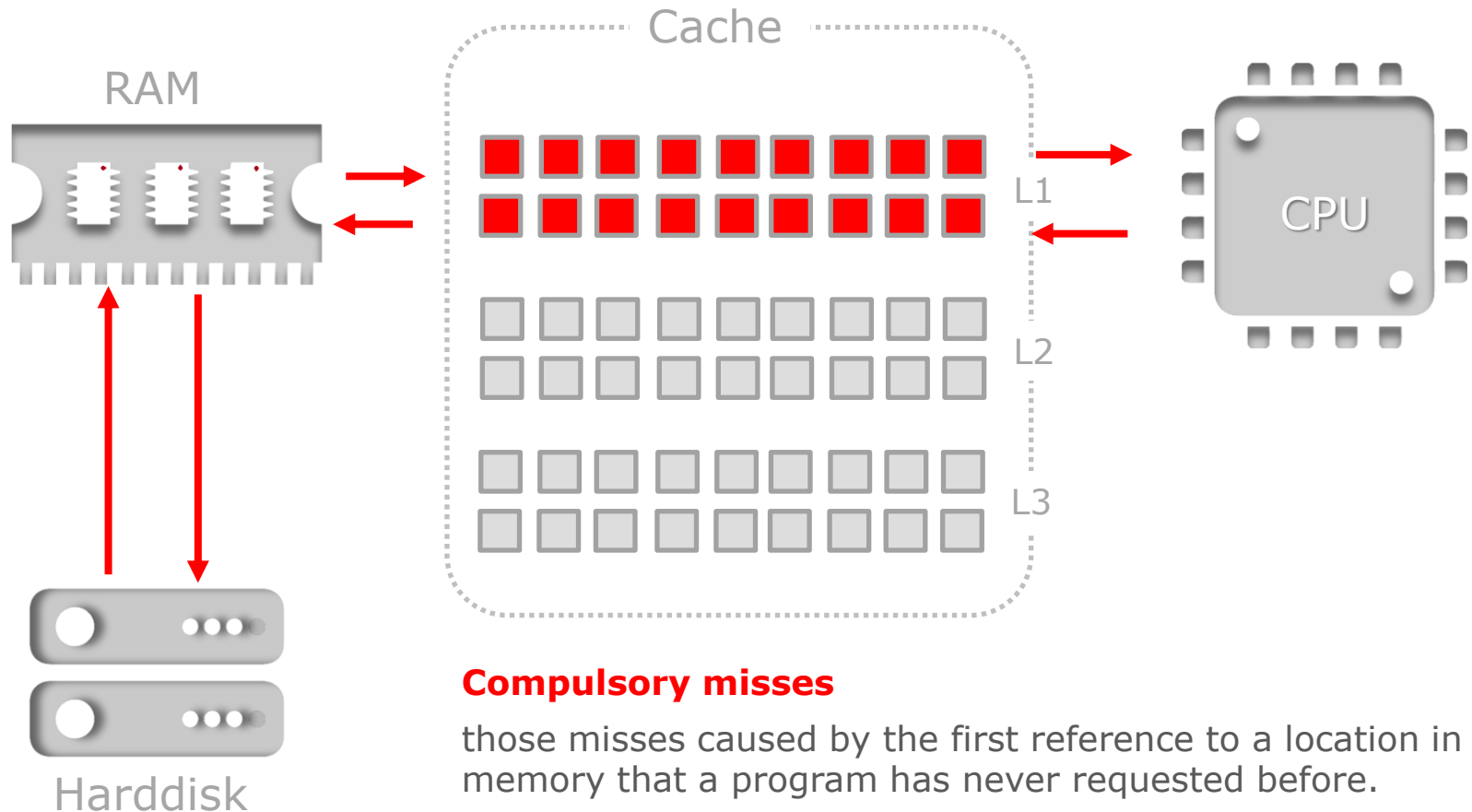
Arithmetic Intensity

Incorporate all cache (total bytes) behaviors (Compulsory misses, Capacity misses, Conflict misses) with Locality.

Integrating Performance Components



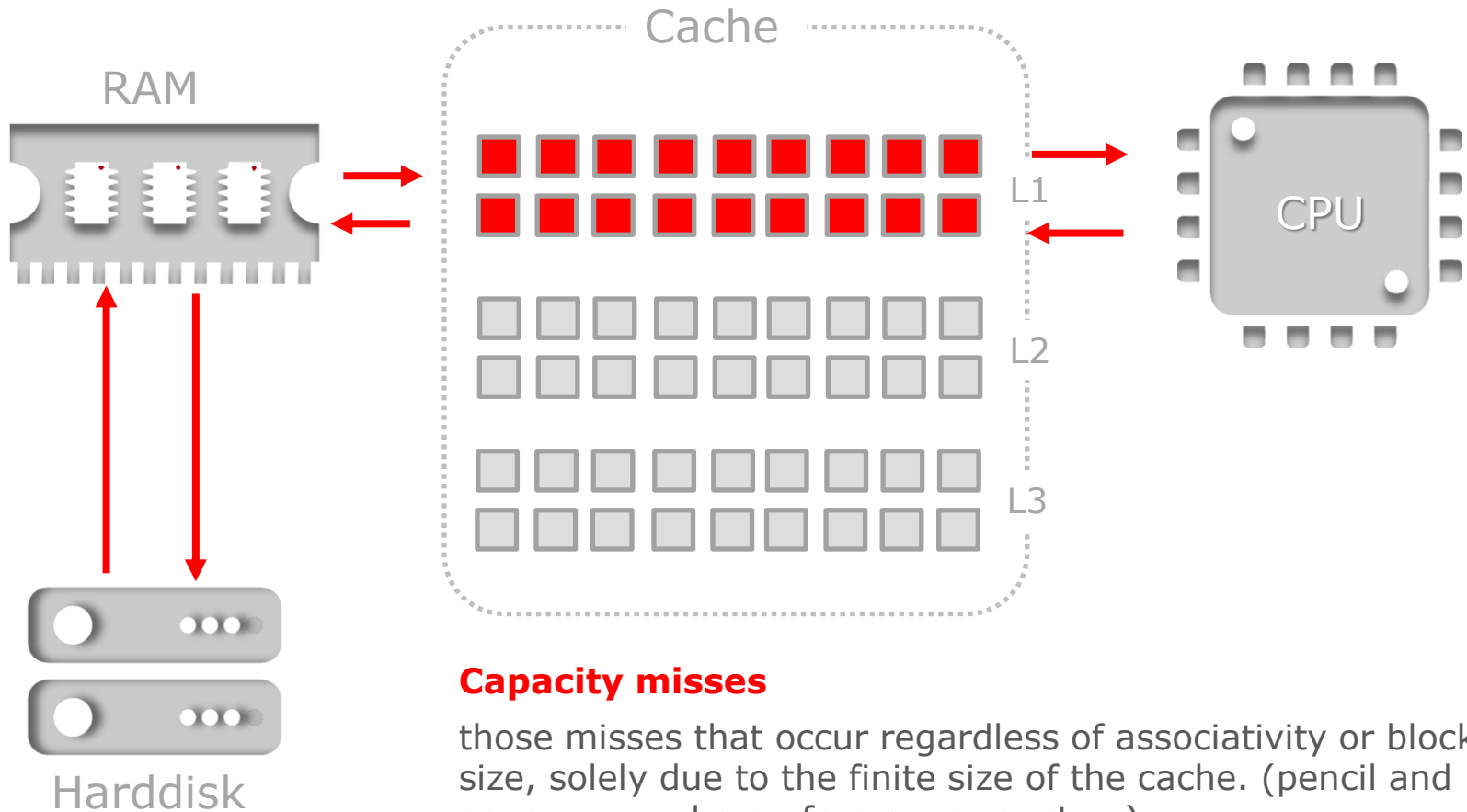
Integrating Performance Components



Compulsory misses

those misses caused by the first reference to a location in memory that a program has never requested before.

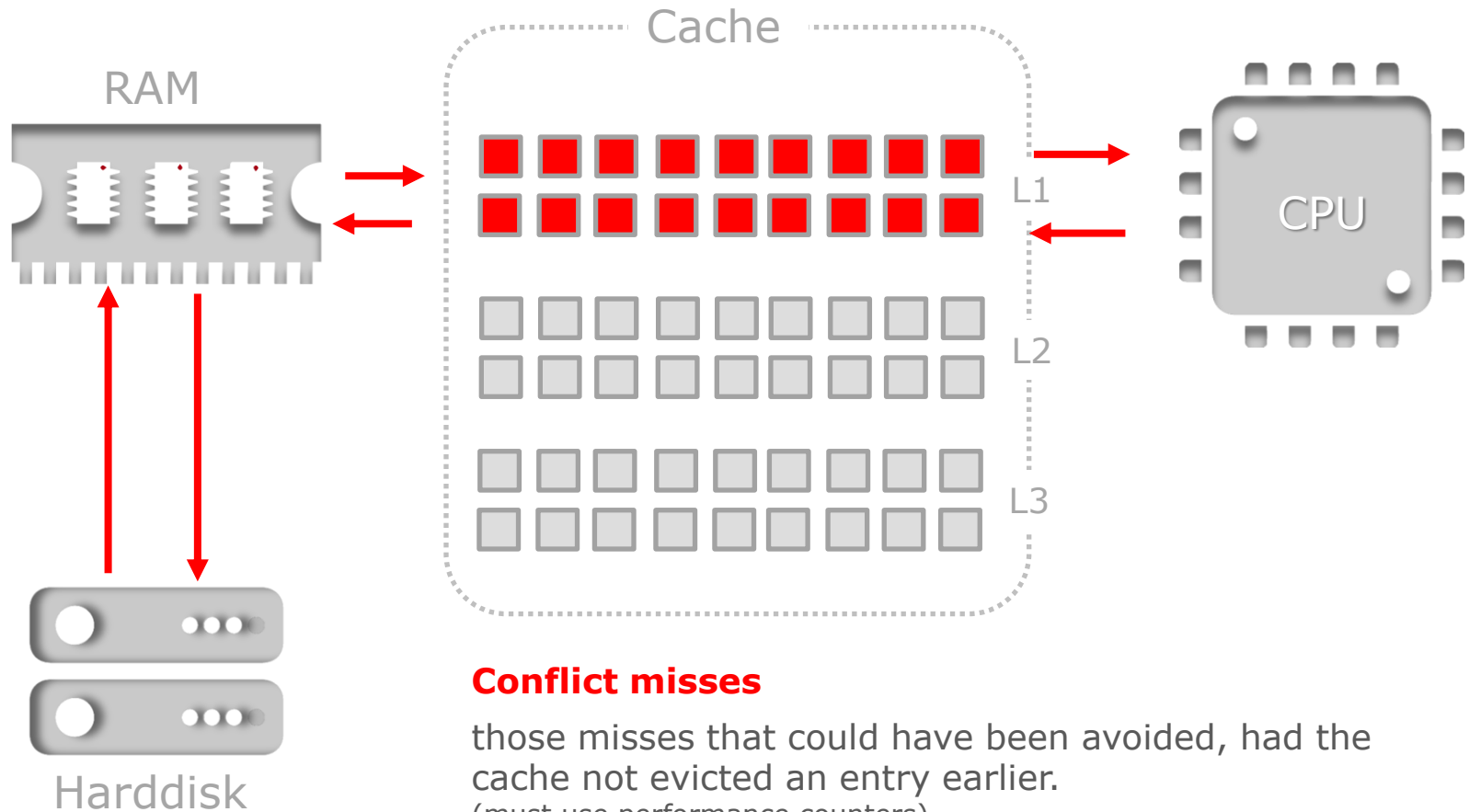
Integrating Performance Components



Capacity misses

those misses that occur regardless of associativity or block size, solely due to the finite size of the cache. (pencil and paper or maybe performance counters).

Integrating Performance Components



Different Performance Models

System/Architectural



Predict performance on multiprocessors

Bottleneck analysis

Provide insight into performance factors

Applicable to heterogeneous multicore computers

Stochastic analytical models Statistical performance models

Roofline Modeling

Easy-to-understand used nearly for

20^{Years}
by

Programmers
Compiler Writers
CPU architects



Hardly to be used by non-experts

Ignore block size, block allocation policy, and block replacement policy

Roofline Model

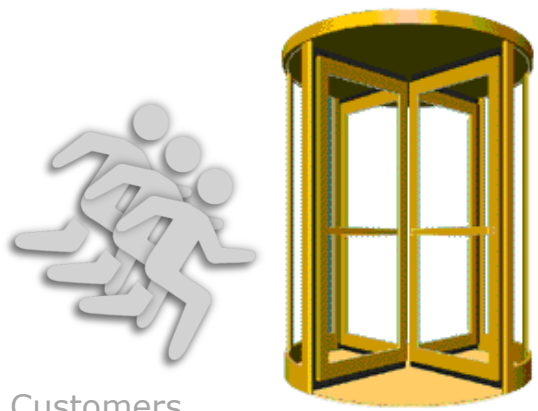
Modeling customer dispatch in a bank

[5]

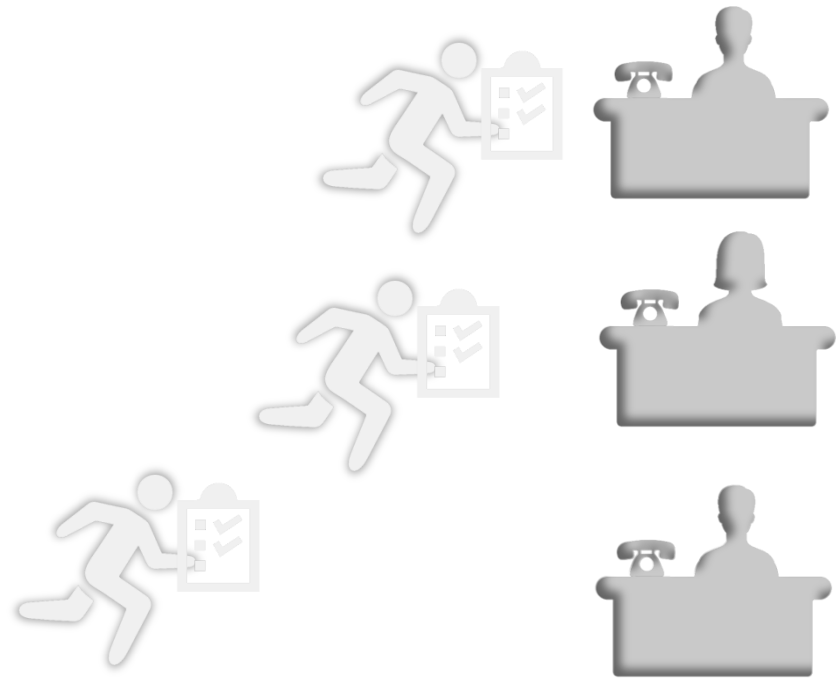
P_{max} (tasks/sec)

Revolving door throughput b_s (customers/sec)

Processing Capacity



Customers



Intensity (tasks/customer)

Bank Employee

[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

Roofline Model Modeling customer dispatch in a bank [5]

How fast can be processed? P (tasks/sec)

The bottleneck is either:

- The service desks (max. tasks/sec): P_{\max}
- The revolving door (max. customers/sec): $I \cdot b_s$

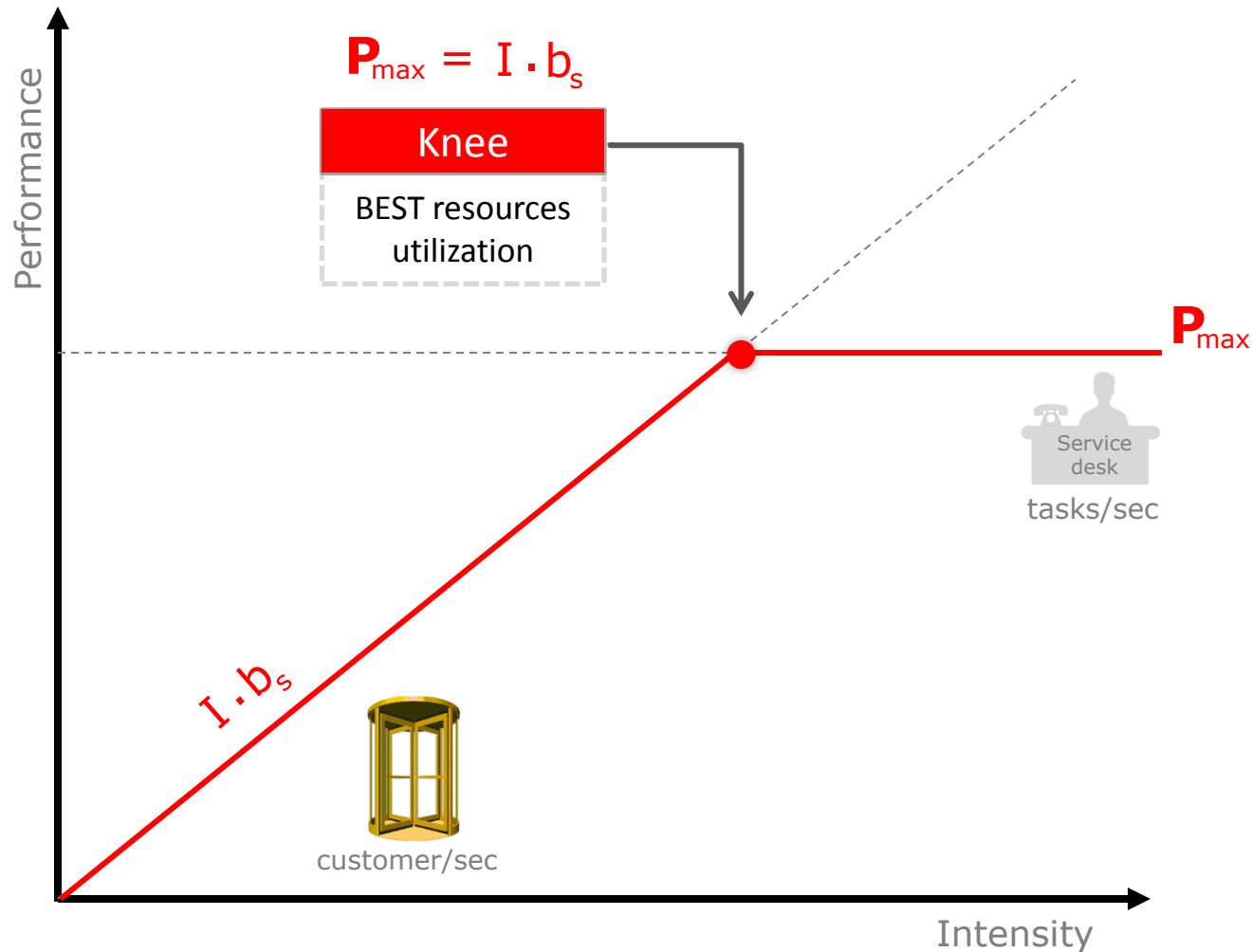
$$P = \min (P_{\max} , I \cdot b_s)$$

[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

Roofline Model

Modeling customer dispatch in a bank

[5] [6]



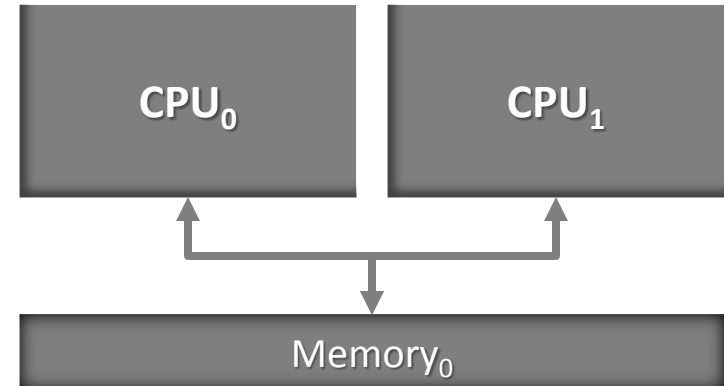
[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

[6] W. Schönauer (2000), "Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers"

Roofline Model^{[5][7]}

Consider a simple kernel that:

1. Transfer Bytes of data from Memory₀
2. Perform F/2 FLOPs on both CPUs
3. Memory can support PeakBandwidth Bytes/sec
4. The two CPUs combined can perform PeakPerformance FLOPs/sec



P_{max}: Loop peak performance taking in consideration the data transfer from L1 cache (not necessarily P_{peak})

Computational intensity (I): “work” per byte transferred through the slowest data path (“the bottleneck”) (measured in flops/bytes)^[6]

Code balance (B_c) = I⁻¹

b_s: Peak bandwidth of the slowest data path (byte/sec)

Expected Performance

$$P = \min (P_{\max}, I \cdot b_s)$$

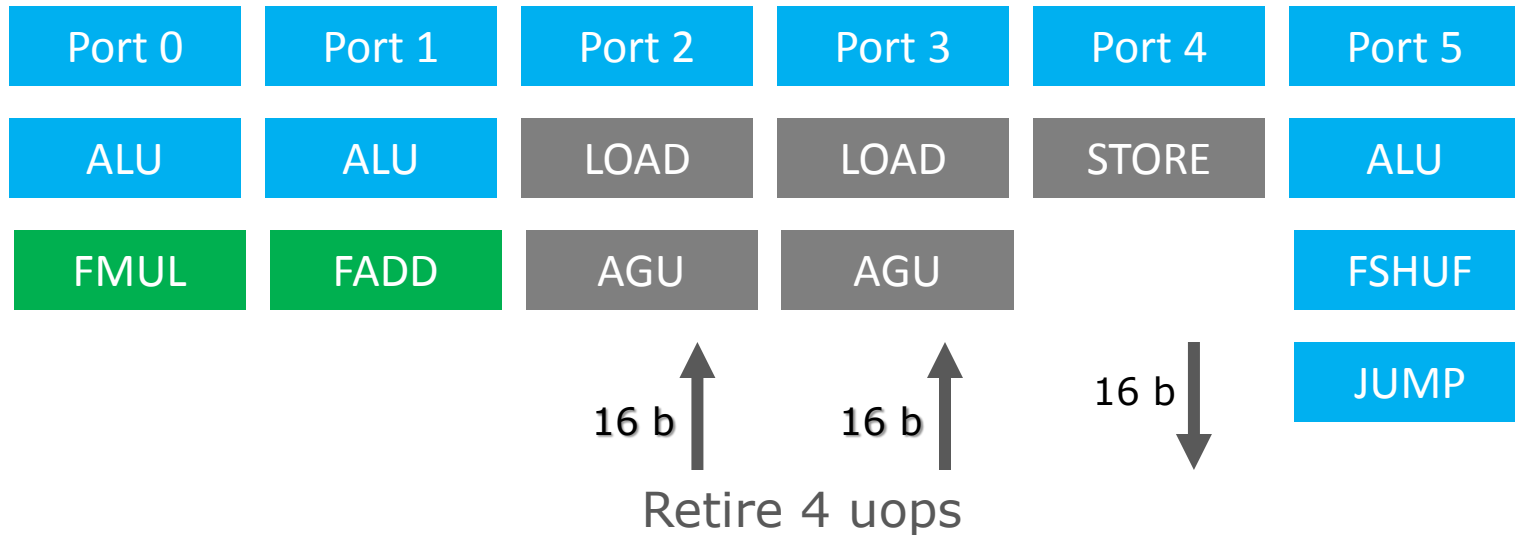
[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

[7] S. Williams (2008), "Auto-tuning Performance on Multicore Computers", UCB Technical Report No. UCB/Eecs-2008-164. PhD thesis

Roofline Model

- **Bandwidth** #'s collected via micro benchmarks
- **Computation** #'s derived from optimization manuals (pencil and paper)
- Assume complete overlap of either:
 - communicationor
 - computation

Roofline Model Analysis (P_{max}) [5], [8], [9]



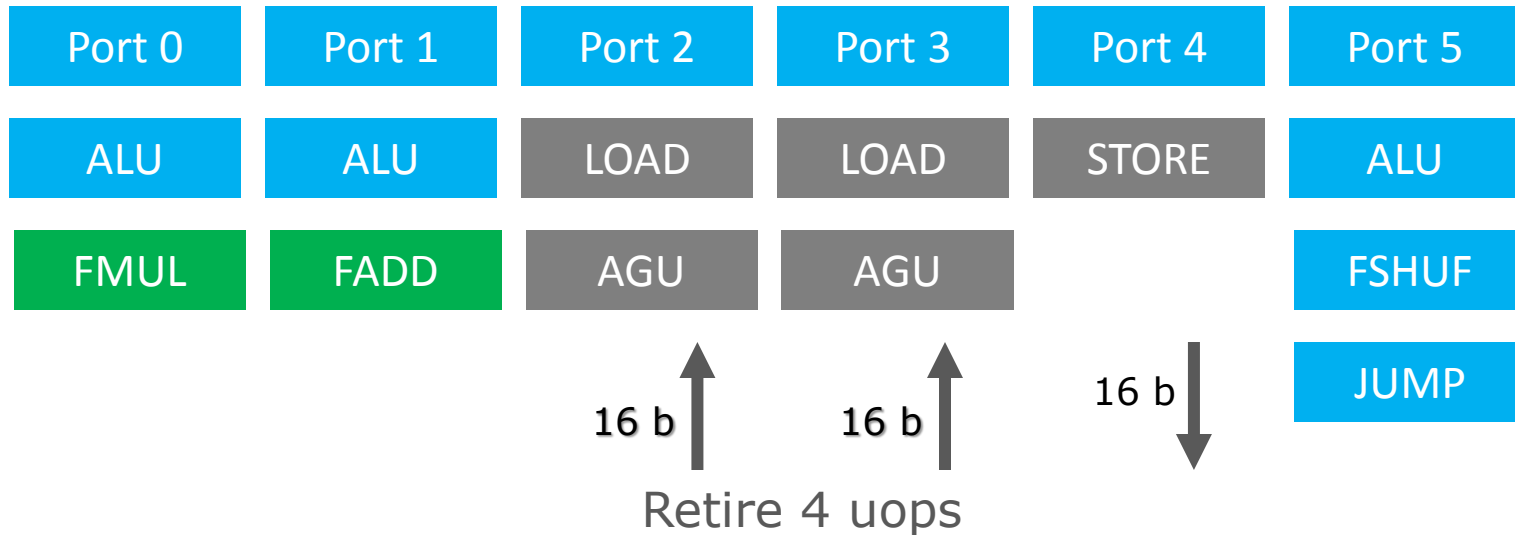
- **Assumption:** All instructions in a loop are maintained independently to different ports
- **Complex cases:** Sum number of penalty cycles for each cycle with AVX.

[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

[8] Datasheet Addendum, 2nd Generation Intel® Core™ Processor Family Mobile with ECC, May 2012

[9] Intel free Press, "Ron Friedman: The Man Behind Sandy Bridge", December 28, 2010. Retrieved November 11, 2011 <http://www.intelfreepress.com/news/the-man-behind-sandy-bridge/>

Roofline Model Analysis (P_{max}) [5], [8], [9]



- one load instruction + 1/2 store instruction
- one AVX MULT + one AVX ADD

Per cycle with SSE or scalar

- Two load instruction
- one MULT + one ADD instruction

Maximum of four micro-ops but three is more realistic

[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

[8] Datasheet Addendum, 2nd Generation Intel® Core™ Processor Family Mobile with ECC, May 2012

[9] Intel free Press, "Ron Friedman: The Man Behind Sandy Bridge", December 28, 2010. Retrieved November 11, 2011
<http://www.intelfreepress.com/news/the-man-behind-sandy-bridge/>

Example

Roofline Model

Analysis (P_{max}) [5], [8], [9]

```

Double *A, *B, *C, *D;
For (int i=0; i<N; i++) {
    A[i] = B[i] + C[i] * D[i]}

```

Number of cycles to process one AVX-vectorized iteration

- Cycle 1: LOAD + 1/2 STORE + MULT + ADD
- Cycle 2: LOAD + 1/2 STORE
- Cycle 3: LOAD

One AVX iteration (3 cycles) performs $4 \times 2 = 8$ Flops $\rightarrow 8$ Flops / 3 cy
 $3 \text{ Gcy/s} * 8 \text{ F} / 3 \text{ cy} = 8 \text{ GFlops/s}$

Bandwidth:

$8 \text{ GFlops/s} * 32 \text{ Byte} / 2 \text{ Flops} = 128 \text{ GBytes/s}$

Assume 3 GHz 8-core Sandy Bridge chip

$b_s = 40 \text{ GB/s}$

[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.
 [8] Datasheet Addendum, 2nd Generation Intel® Core™ Processor Family Mobile with ECC, May 2012
 [9] Intel free Press, "Ron Friedman: The Man Behind Sandy Bridge", December 28, 2010. Retrieved November 11, 2011
<http://www.intelfreepress.com/news/the-man-behind-sandy-bridge/>

Example

Roofline Model

Analysis (P_{max}) [5], [8], [9]

```

Double *A, *B, *C, *D;
For (int i=0; i<N; i++) {
    A[i] = B[i] + C[i] * D[i]}

```

Number of cycles to process one AVX-vectorized iteration

- Cycle 1: LOAD + 1/2 STORE + MULT + ADD
- Cycle 2: LOAD + 1/2 STORE
- Cycle 3: LOAD

$$B_c = (4+1) \text{ Words} / 2 \text{ Flops} = 2.5 \text{ W/F}$$

$$\rightarrow I = 0.4 \text{ F/W} = 0.05 \text{ F/B}$$

$$\rightarrow I \cdot b_s = 2.0 \text{ GF/s} \text{ (1.04 \% of peak performance)}$$

$$P_{peak} = 192 \text{ Gflop/s} \text{ (8 cores} \times \text{(4+4) Flops/cy} \times \text{3.0 GHz)}$$

$$P_{max} = 8 \times 8 \text{ Gflop/s} = 64 \text{ Gflop/s} \text{ (33\% peak)}$$

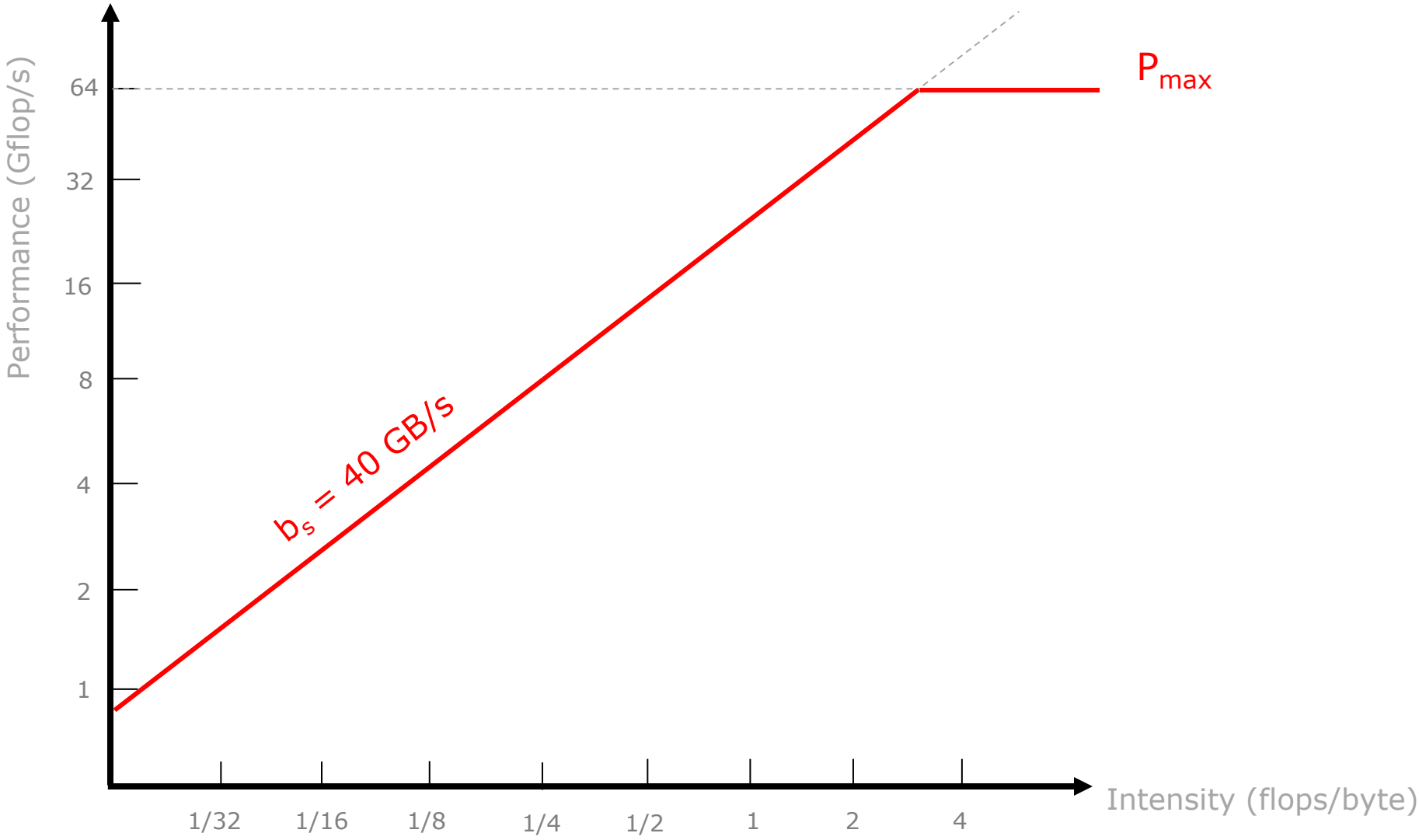
$$P = \min(P_{max}, I \cdot b_s) = \min(64, 2.0) \text{ GFlop/s} = 2.0 \text{ Gflop/s}$$

[5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.

[8] Datasheet Addendum, 2nd Generation Intel® Core™ Processor Family Mobile with ECC, May 2012


[9] Intel free Press, "Ron Friedman: The Man Behind Sandy Bridge", December 28, 2010. Retrieved November 11, 2011
<http://www.intelfreepress.com/news/the-man-behind-sandy-bridge/>

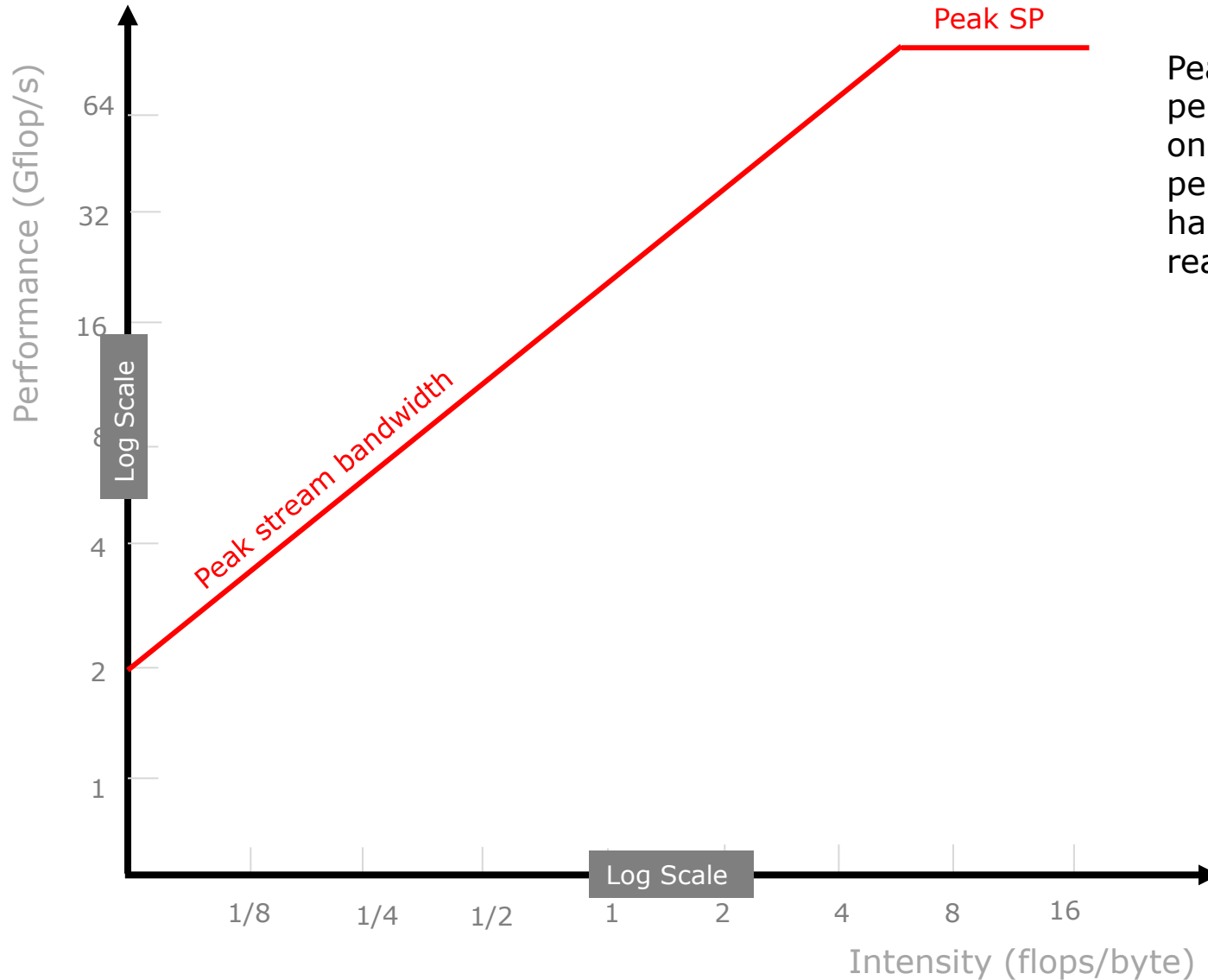
Roofline Model Graph [10]



[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Roofline Model Graph Analysis [10]


Opteron 2356 

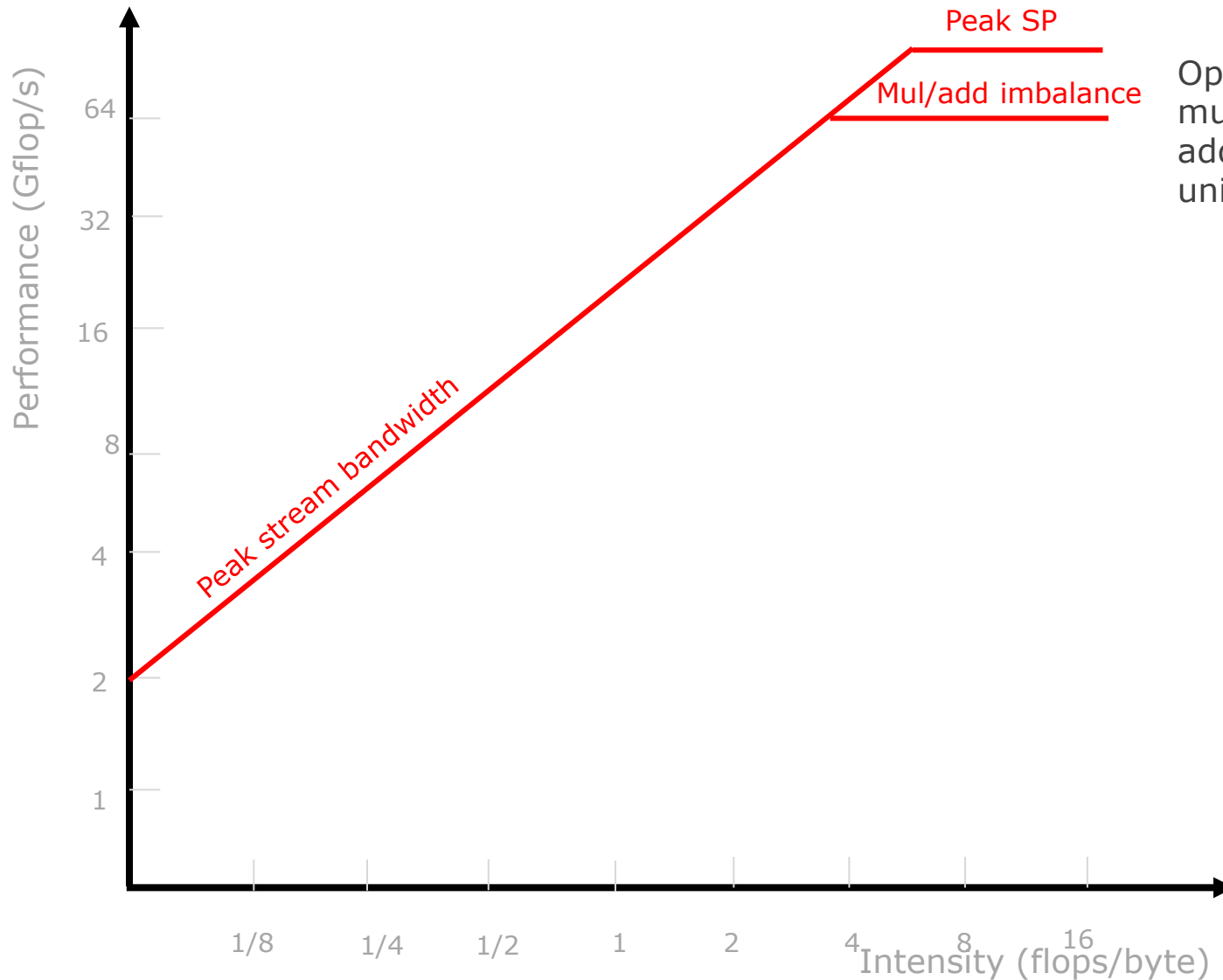


Peak roofline performance based on manual single precision peak and a hand tuned stream read for bandwidth

[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Roofline Model Graph Analysis [10]

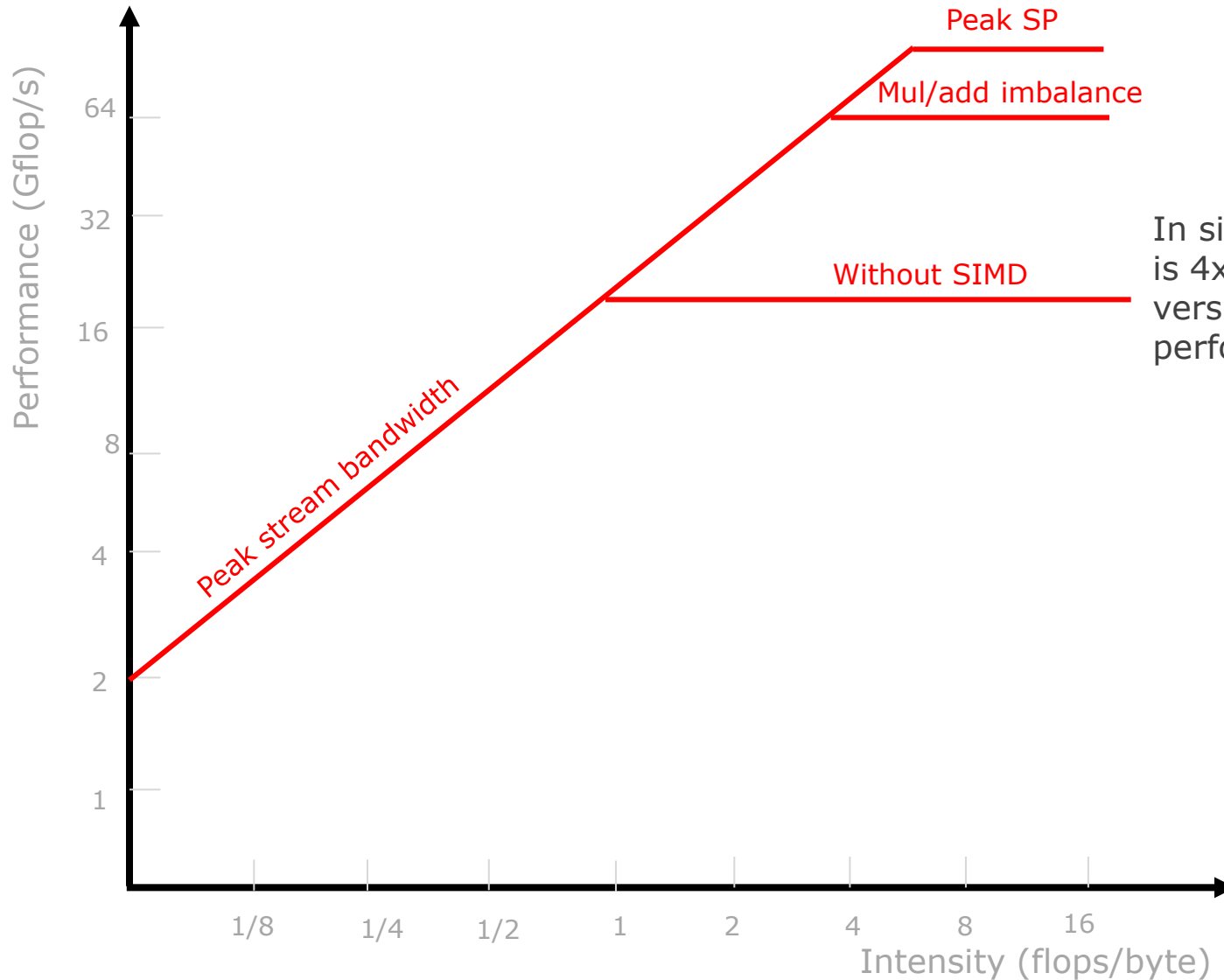
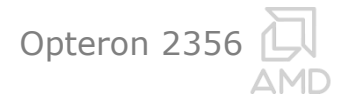
Opteron 2356 



Opteron has separate multipliers as well as adders and a functional unit parallelism

[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

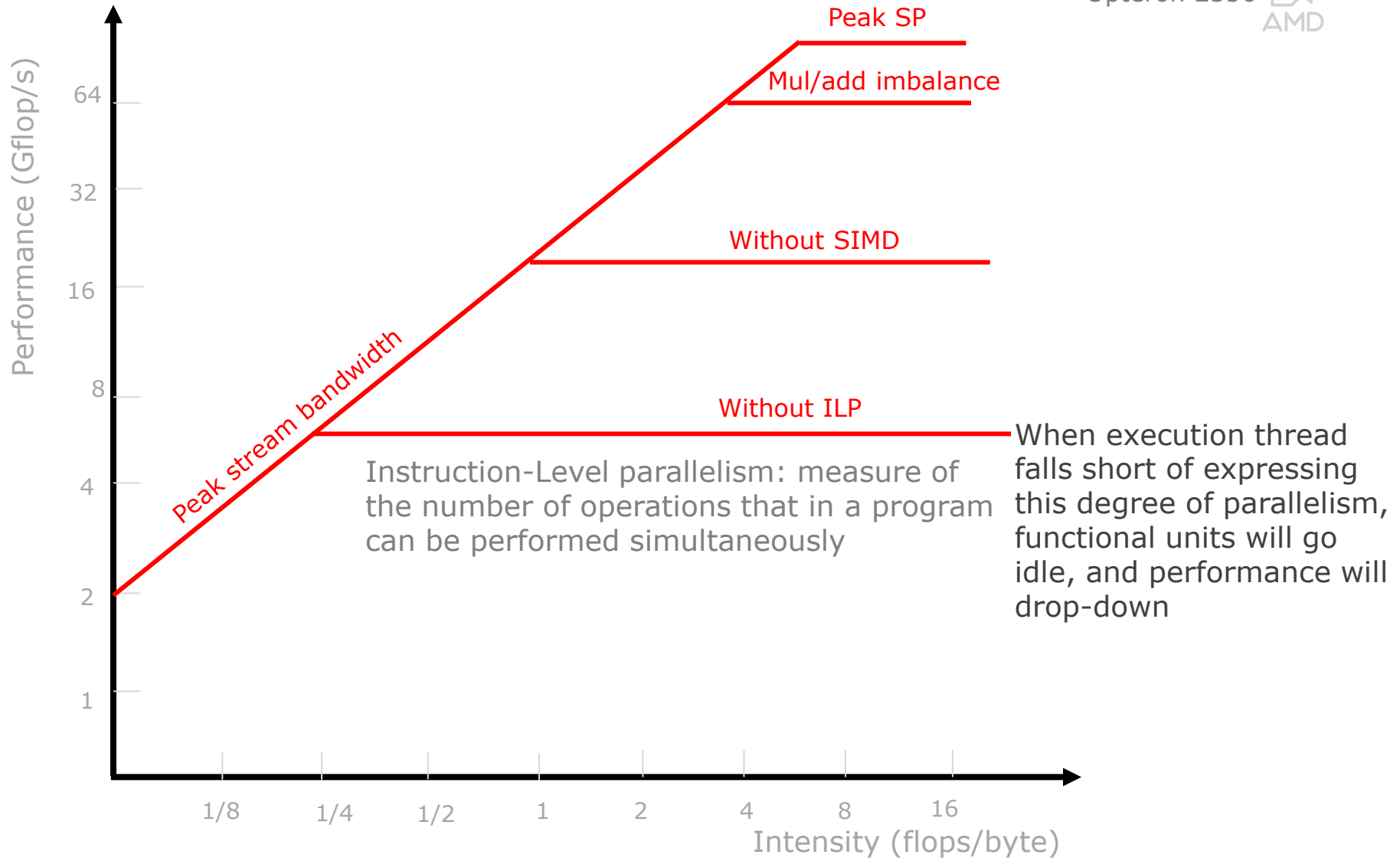
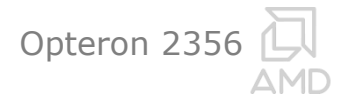
Roofline Model Graph Analysis [10]



In single precision, SIMD is 4x32b, if only the `_ss` versions are used, performance is 1/4

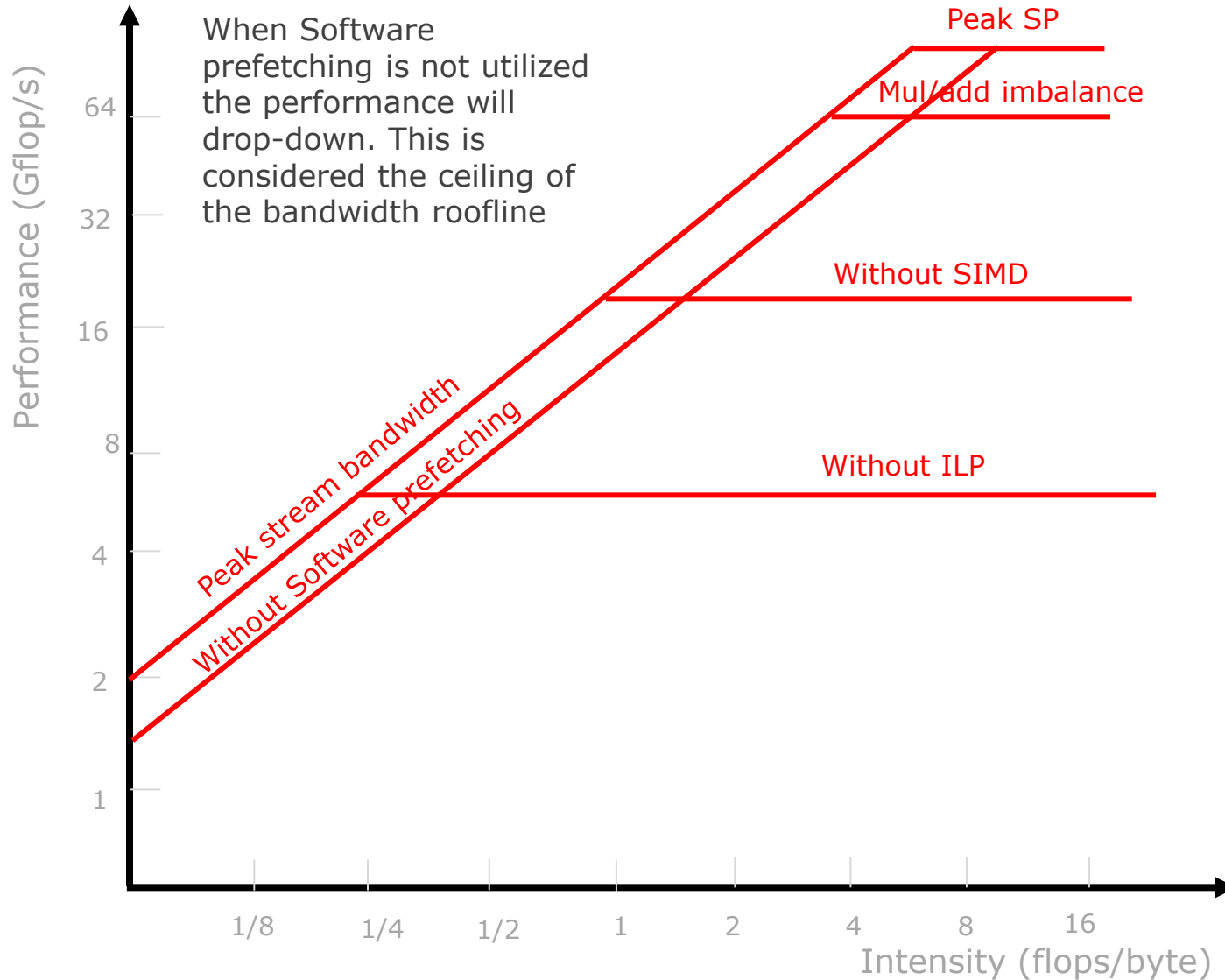
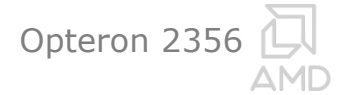
[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Roofline Model Graph Analysis [10]




[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

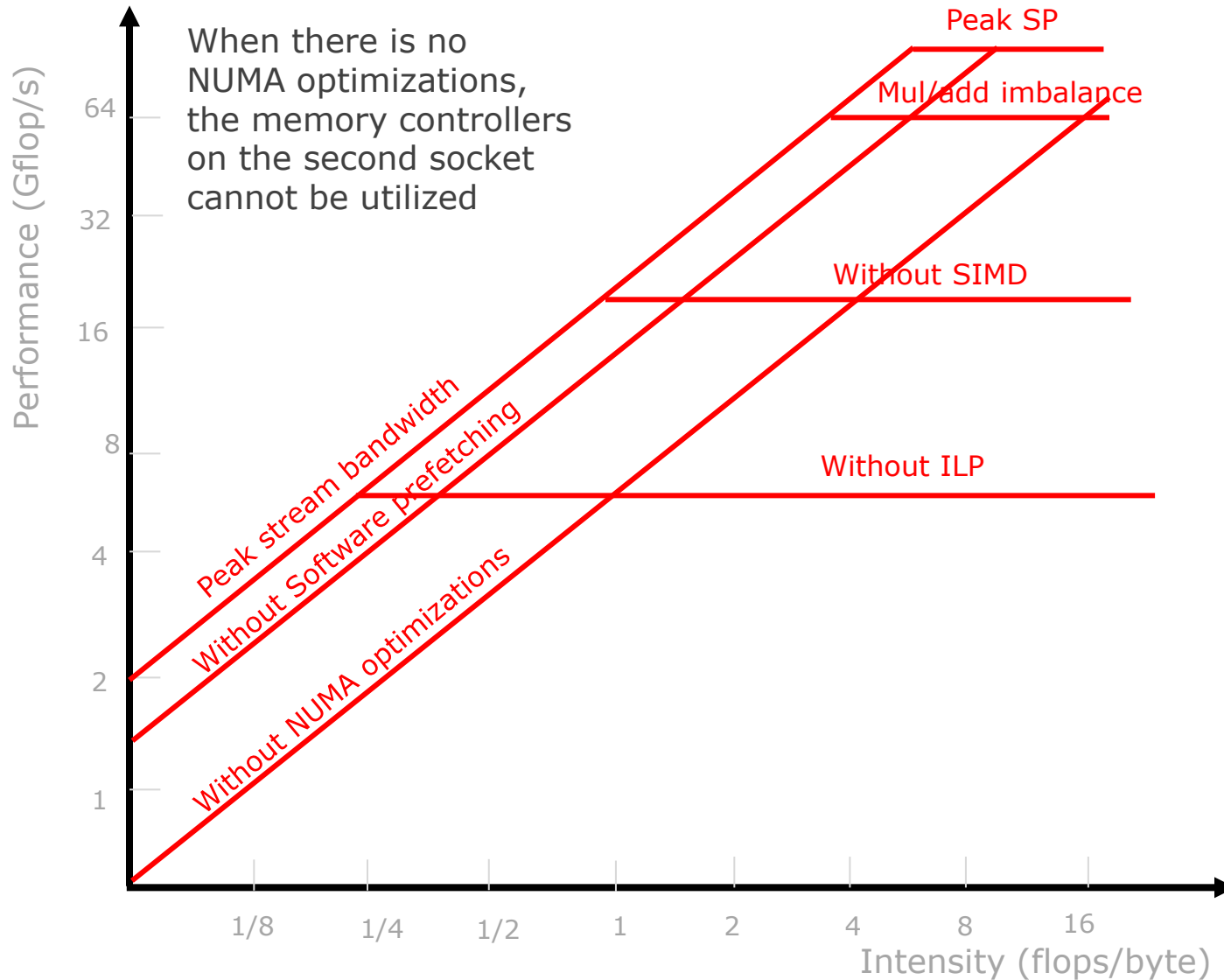
Roofline Model Graph Analysis [10]



[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab
Performance Modeling – Ahmed Hassan

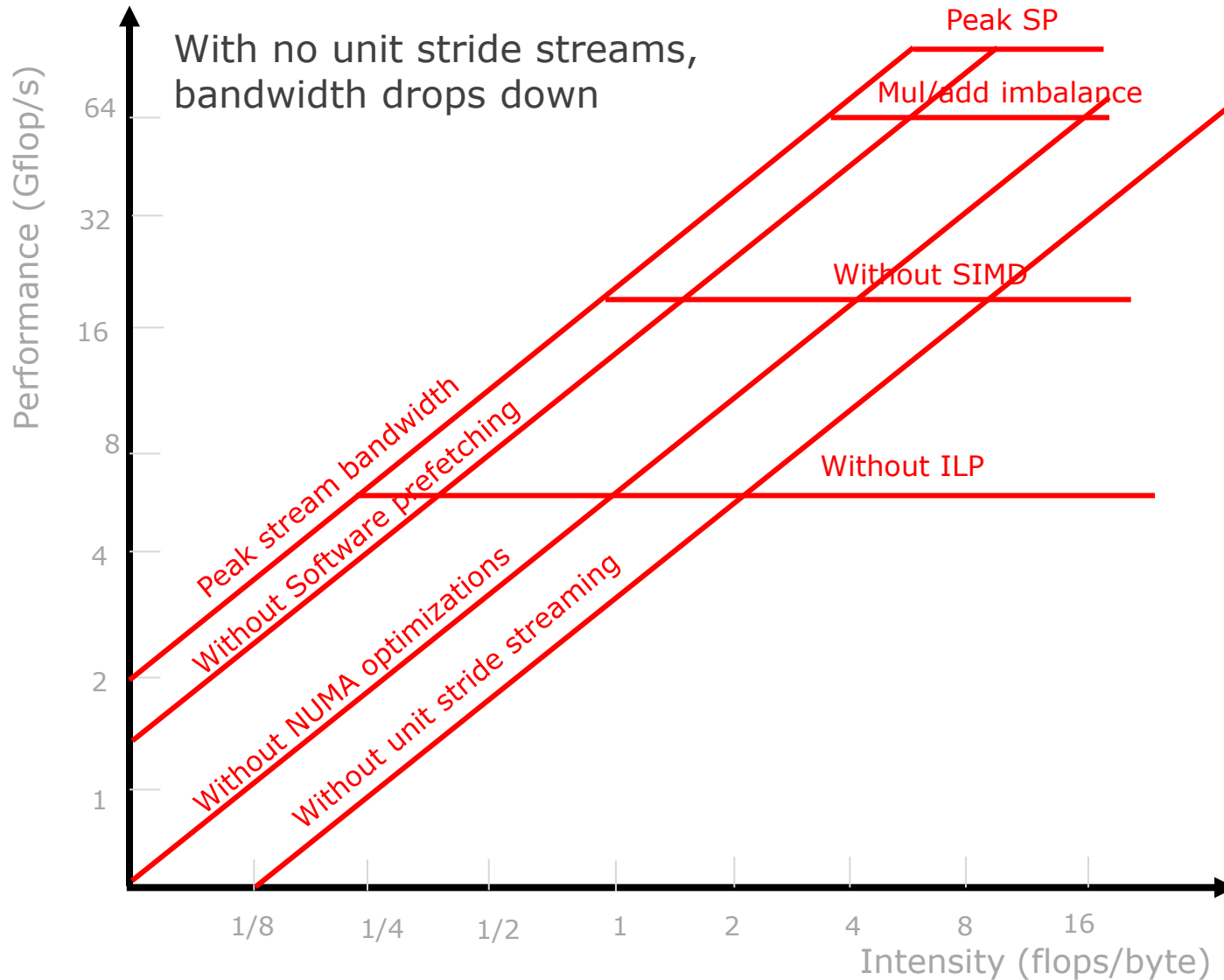
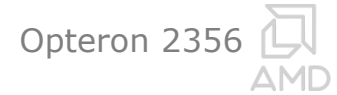
Roofline Model Graph Analysis [10]

Opteron 2356 



[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Roofline Model Graph Analysis [10]



[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Roofline Model

Graph Analysis

[10]

- There is no standard/single ordering or roofline model.
- The ceiling order is generally bottom up.
- Addition, Multiplication and FMA are balanced inherent in many linear algebra routines.
- Addition is the most dominant operation thus the multipliers and FMA go underutilized.

[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Execution-Cache-Memory (ECM) [11]

What is the optimal number of cores for minimum energy to solution?

Is it more energy-efficient to use more cores at lower clock speed than fewer cores at higher clock speed?

When exactly does the 'race to idle' rule apply?

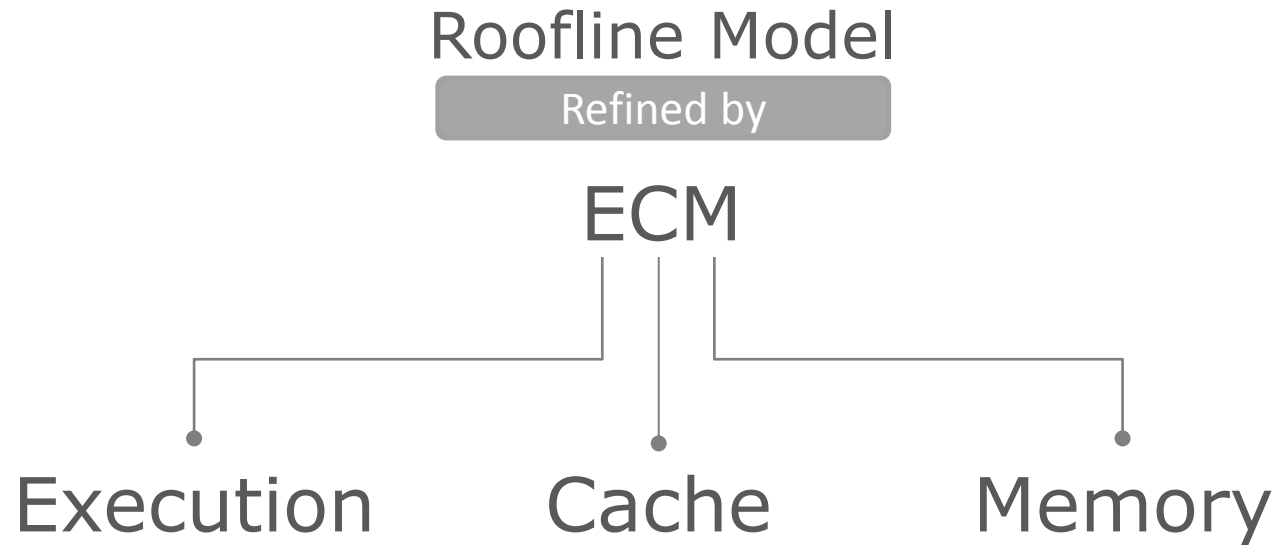


Is it necessary to sacrifice performance in favor of low energy consumption?

What is the influence of single-core optimization on energy efficiency?

[11] Treibig J, Hager G. Introducing a performance model for bandwidth-limited loop kernels. Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J (eds.). Springer Berlin / Heidelberg, 2010; 615–624, doi:10.1007/978-3-642-14390-8_64.

Execution-Cache-Memory (ECM) [11]



[11] Treibig J, Hager G. Introducing a performance model for bandwidth-limited loop kernels. Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J (eds.). Springer Berlin / Heidelberg, 2010; 615–624, doi:10.1007/978-3-642-14390-8_64.

Execution-Cache-Memory (ECM) [11] [12]

- Refines the roofline model to predict the behavior of saturation and scaling of bandwidth-limit.
- Provides a clear understand of the single and multi-core performance of streaming kernels.

Intel Sandy Bridge processor exposes part of its power characteristics to the programmer through “Running Average Power Limit” feature.

[11] Treibig J, Hager G. Introducing a performance model for bandwidth-limited loop kernels. Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J (eds.). Springer Berlin / Heidelberg, 2010; 615–624, doi:10.1007/978-3-642-14390-8_64.

[12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.

Execution-Cache-Memory (ECM) Hardware [12] [13]

- The majority of numerical codes are based on streaming loop kernels.
- Kernels are always limited by the bandwidth of memory that leads to a distinct scaling behavior across the cores of a multicore chip.

Roofline model is used to anticipate the performance

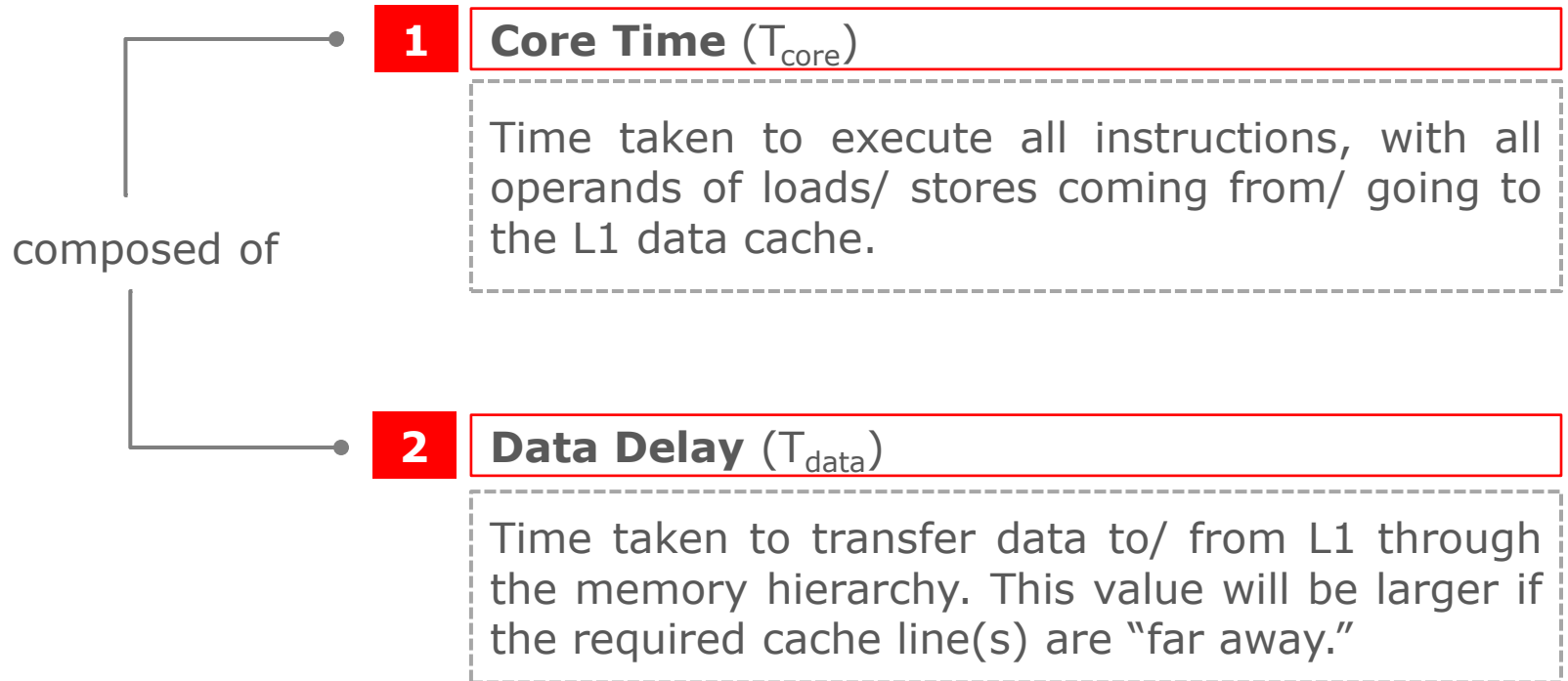
BUT

ECM model provides essential insight about the cache bandwidths and organization on the multicore chip to show up a more authentic characterization on the single-core level.

[12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.

[13] Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro 2012; 32:20–27, doi:10.1109/MM.2012.12.

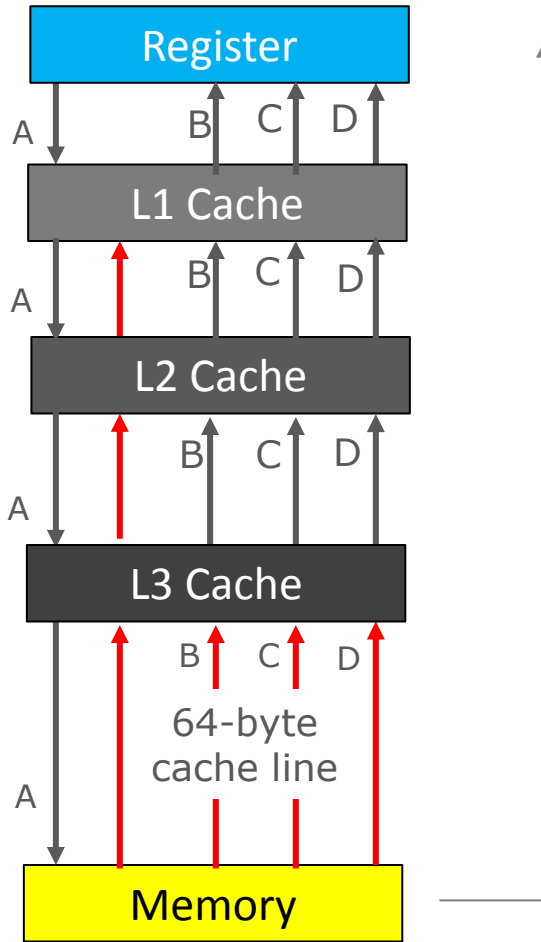
Execution-Cache-Memory (ECM) Single core [12] [13]




[12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.

[13] Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro 2012; 32:20–27, doi:10.1109/MM.2012.12.

Execution-Cache-Memory (ECM) Single core [12] [13]



Maximum: 4 cycles
 Minimum: 2 cycles
 depending on whether the transfers can overlap or not
 32-byte wide buses between the cache levels

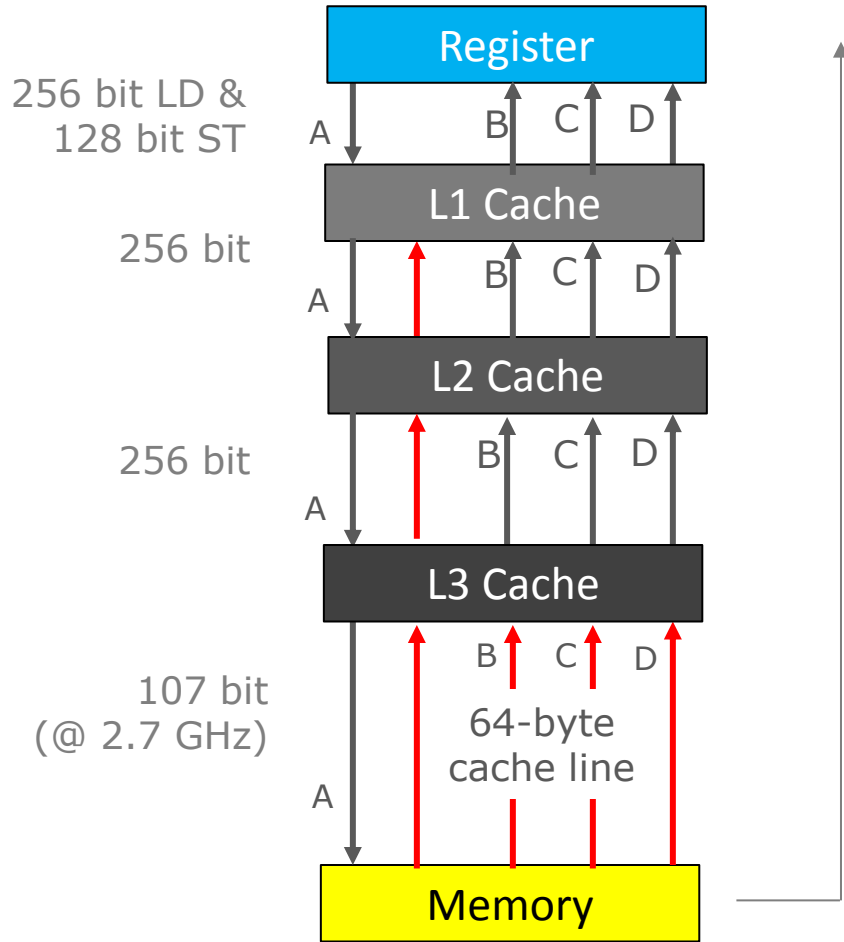
 Intel Architecture Code Analyzer (IACA)
 Tool that can derive more accurate predictions by taking dependencies into account.

[12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.
 [13] Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro 2012; 32:20–27, doi:10.1109/MM.2012.12.

Execution-Cache-Memory (ECM)

Multicore scaling

[12] [13]



$$\max(2(B)+2(C)+2(D), 4(A))cy = 6 cy$$

$$(2(B)+2(C)+2(D), 4(A))cy = 10 cy$$

$$(2(B)+2(C)+2(D), 4(A))cy = 10 cy$$

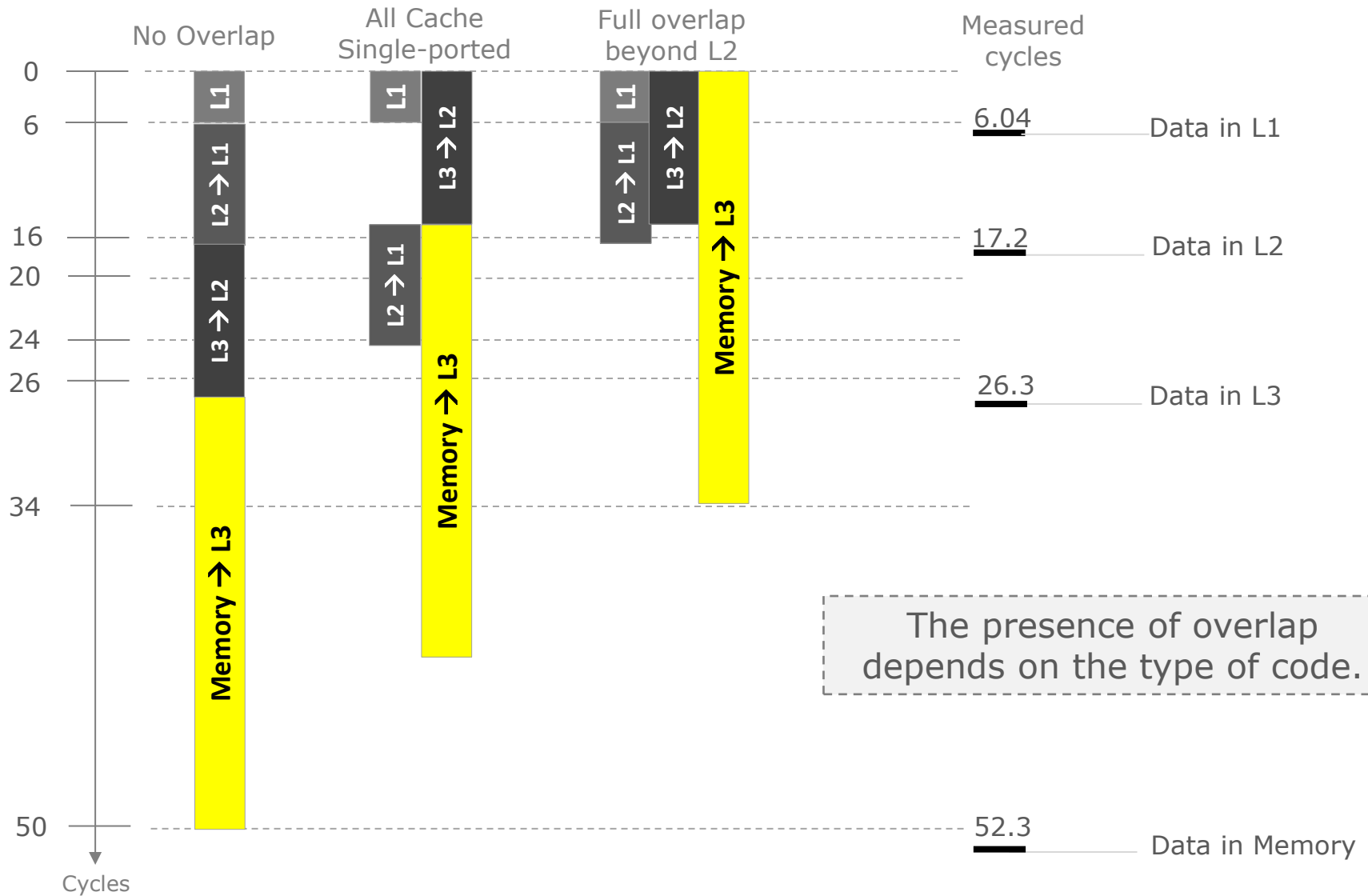
$$(5.64 B. 2.7Gcy/s)/(36 GB/s) = 24 cy$$

Single-core ECM model anticipate lower and upper limits of the bandwidth pressure on all memory hierarchy levels. When the bandwidth capacity of one level is drained, performance starts to saturate.

[12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.
 [13] Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro 2012; 32:20–27, doi:10.1109/MM.2012.12.

Execution-Cache-Memory (ECM)

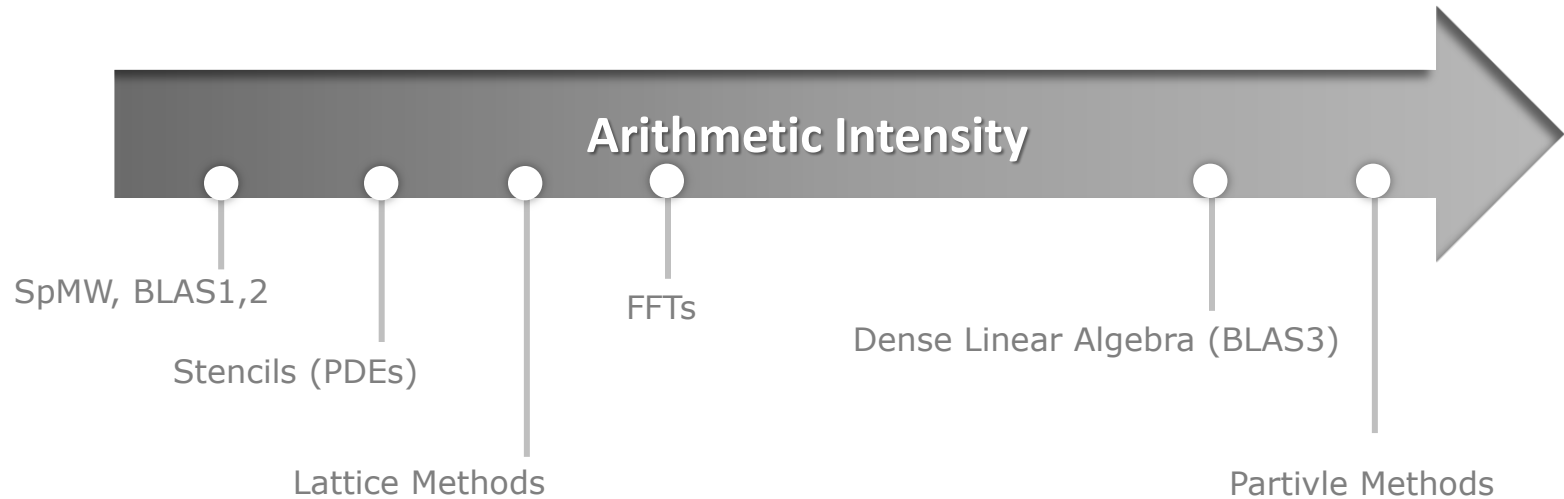
Multicore scaling [12] [13]



The presence of overlap depends on the type of code.

[12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.
 [13] Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro 2012; 32:20-27, doi:10.1109/MM.2012.12.

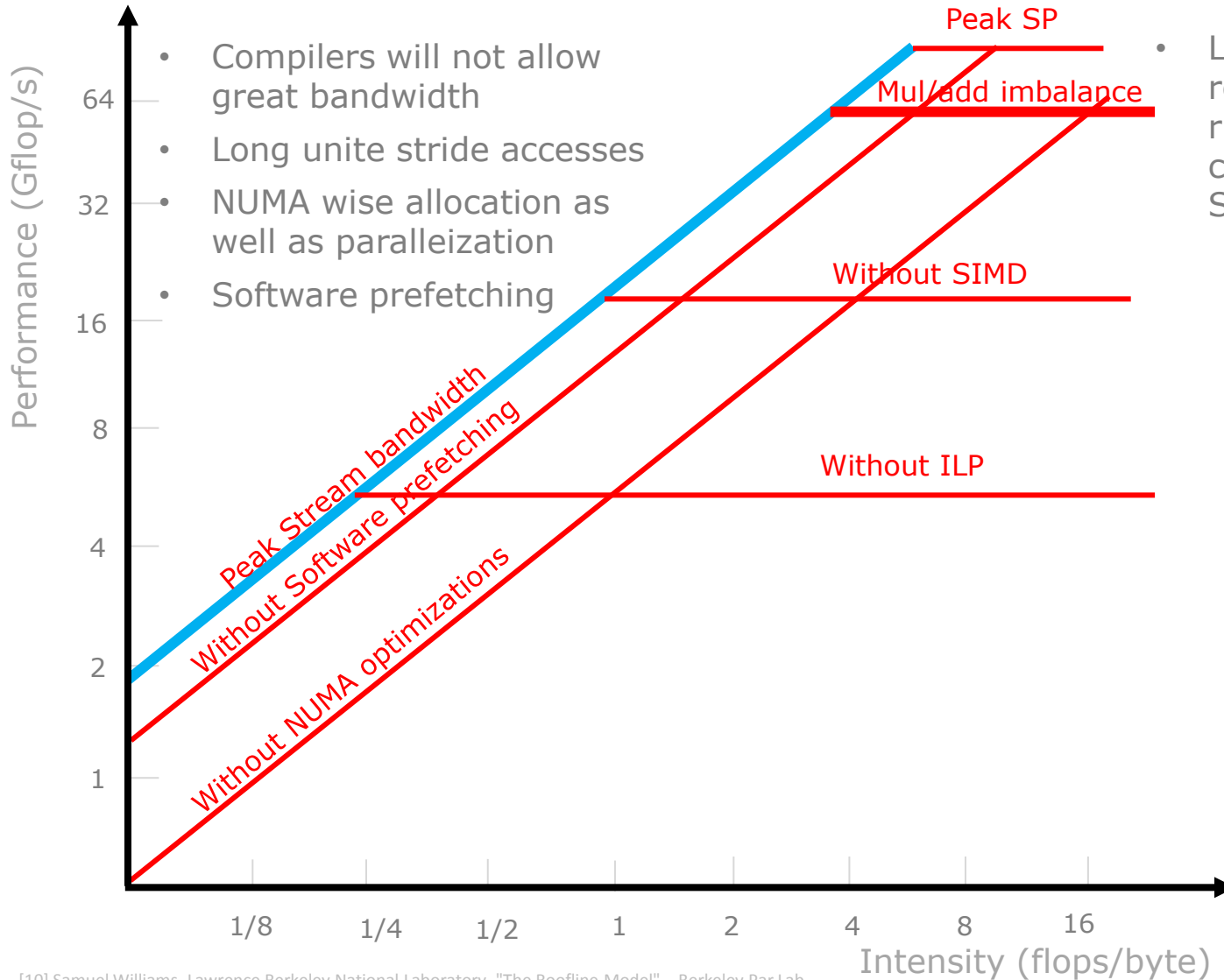
Roofline Model in HPC **Overview** [10]



- Certain arithmetic intensity is exceed by local store space.
- Arithmetic Intensity (AI) \sim Total Flops / Total DRAM Bytes
- Some HPC kernels have a constant arithmetic intensity.

[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Software Optimization [10]



- Compilers will not allow great bandwidth
- Long unite stride accesses
- NUMA wise allocation as well as paralleization
- Software prefetching

- Loop unrolling, reordering, and long running loops are considered as a type of Software optimization.

[10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab

Application/Service Modeling [14]

Other way of performance modeling is to model a communication between application/services.

Inputs

- Scenarios and design documentation about critical and significant use cases.
- Application design and target infrastructure and any constraints imposed by the infrastructure.
- QoS requirements and infrastructure constraints, including service level agreements (SLAs).
- Workload requirements derived from marketing data on prospective customers.

[14] HP LoadRunner User Manual, v11.00, URL: http://community.hpe.com/hpeb/attachments/hpeb/sws-LoadRunner_SF/11042/1/hp_man_LoadRunner11.00_AnalysisUser_pdf.pdf

Application/Service Modeling [14]

Outputs

- A performance model document.
- Test cases with goals.

For example if we are developing an online booking system then we measure the performance of the system with respect to our pre-defined SLA.

e.g.

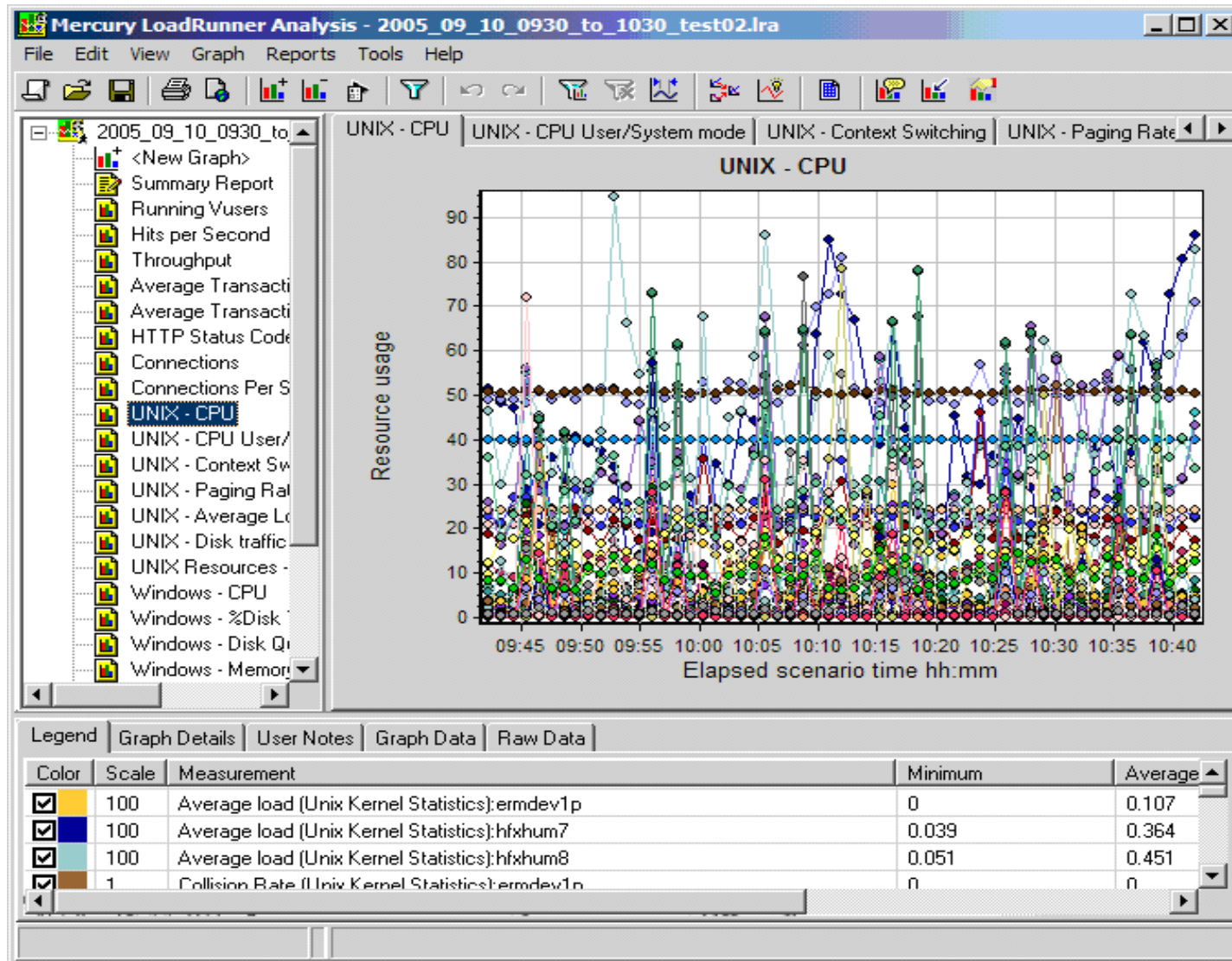
- Number of Co-current booking requests
- Number of Running Vusers
- Number of Hit per Second
- CPU etc

Tools:

Commercial Tools like HP LoadRunner or Open-source Tool like JMeter

[14] HP LoadRunner User Manual, v11.00, URL: http://community.hpe.com/hpeb/attachments/hpeb/sws-LoadRunner_SF/11042/1/hp_man_LoadRunner11.00_AnalysisUser_pdf.pdf

Application/Service Modeling



Source: <http://bish.co.uk> - Performance Testing - Page 17

Conclusion

Roofline Performance Modeling focuses on rates and efficiencies (Gflop/s, % of peak)

Afford a visual assistance that provides:

- Realistic forecast of performance as well as productivity
- Show hardware constraint for a given kernel
- Show potential assistance and priority of optimizations

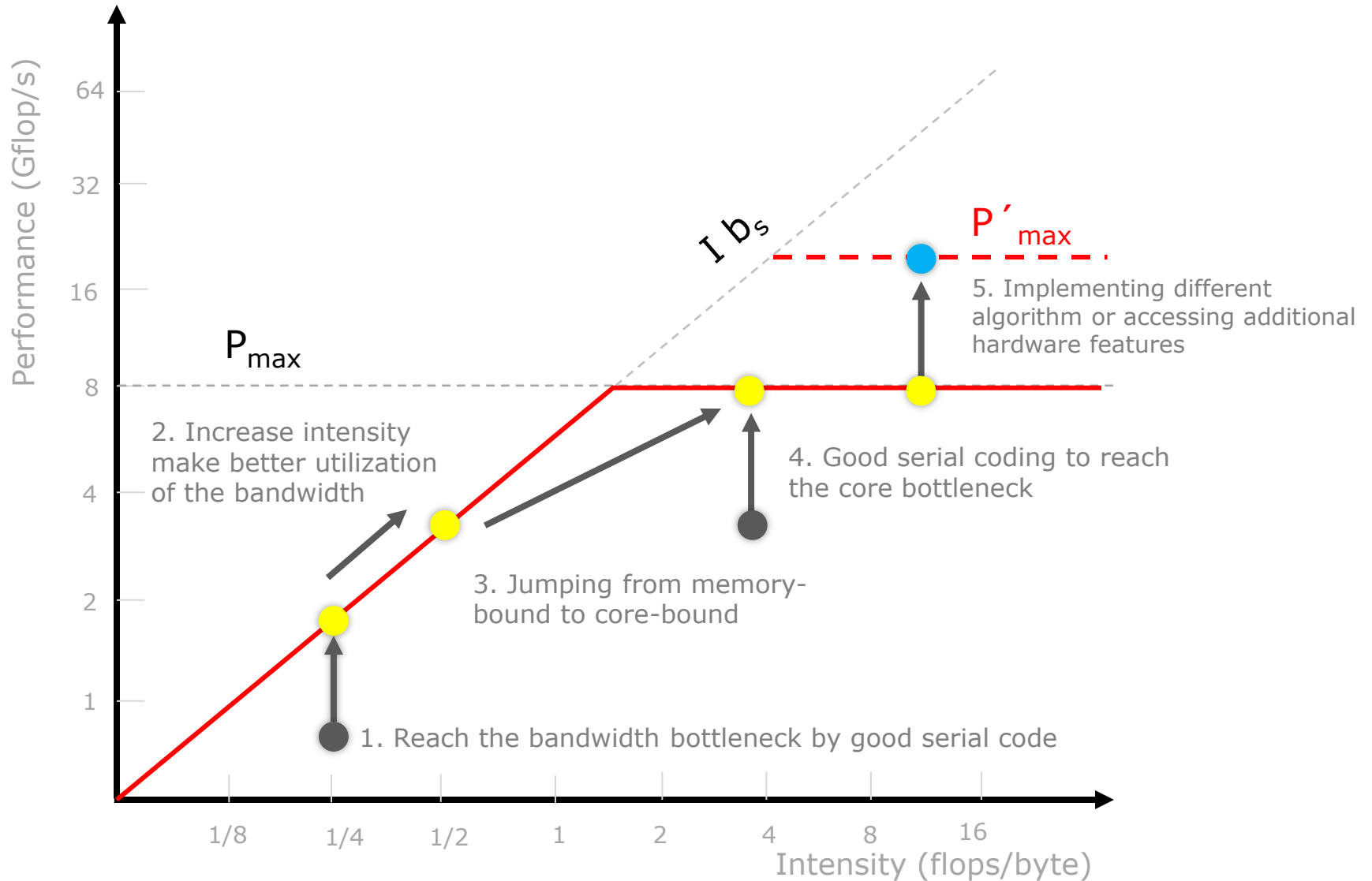
Easily extendable to other architectural paradigms as well as other communication/computation metrics.

Who's not the audience for the Roofline:

- Not for those interested in fine tuning (+5%)
- Not for those challenged by parallel kernel correctness

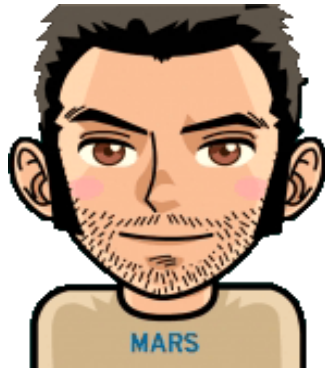
Execution-Cache-Memory (ECM) describes the scaling characteristics of bandwidth bound codes on a multicore chip better than a simple bottleneck analysis.

Conclusion [15]



[15] Performance Modeling: The Roofline Model, "Loop-based performance modeling: Execution vs. data transfer", Friedrich-Alexander Universität, Erlangen-Nürnberg, pp 2-16

Questions?



Thank you

Ahmed Hassan

Sources

- [1] National Center for Supercomputing Applications (NCSA), University of Illinois, USA, "Performance Modeling for Systematic Performance Tuning", ACM 978-1-4503-0771-0/11/11, pp. 1 - 5.
- [2] S. Jarvis, S.Wright, Simon Hammond, High Performance Computing Systems: Performance Modeling, Benchmarking and Simulation, Springer, ISBN: 978-3-319-10213-9, pp. 19 - 26.
- [3] Heike McCraw, Innovative Computing Laboratory, Department of Electrical Engineering and Computer Science, "Performance Modeling", University of Tennessee, 2013, pp. 7 - 9.
- [4] Samuel Williams, David Patterson, ParLab Summer Retreat, "The Roofline Model: A pedagogical tool for program analysis and optimization", Berkeley Par Lab
- [5] Samuel Williams, Andrew Waterman, David Patterson, "Simple performance modeling: The Roofline Model" Communications of the ACM, Vol. 52 No. 4, 10.1145/1498765.1498785, pp. 65 - 76.
- [6] W. Schönauer (2000), "Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers"
- [7] S. Williams (2008), "Auto-tuning Performance on Multicore Computers", UCB Technical Report No. UCB/EECS-2008-164. PhD thesis
- [8] Datasheet Addendum, 2nd Generation Intel® Core™ Processor Family Mobile with ECC, May 2012
- [9] Intel free Press, "Ron Friedman: The Man Behind Sandy Bridge", December 28, 2010. Retrieved November 11, 2011, URL: <http://www.intelfreepress.com/news/the-man-behind-sandy-bridge/>

Sources

- [10] Samuel Williams, Lawrence Berkeley National Laboratory, "The Roofline Model", , Berkeley Par Lab
- [11] Treibig J, Hager G. Introducing a performance model for bandwidth-limited loop kernels. Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J (eds.). Springer Berlin / Heidelberg, 2010; 615–624, doi:10.1007/978-3-642-14390-8 64.
- [12] G. Hager, J. Treibig, J. Habich, and G. Wellein, Erlangen Regional Computing Center (RRZE), "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen, Germany, pp. 1 - 21.
- [13] Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. IEEE Micro 2012; 32:20–27, doi:10.1109/MM.2012.12.
- [14] HP LoadRunner User Manual, v11.00, URL: http://community.hpe.com/hpeb/attachments/hpeb/sws-LoadRunner_SF/11042/1/hp_man_LoadRunner11.00_AnalysisUser_pdf.pdf
- [15] Performance Modeling: The Roofline Model, "Loop-based performance modeling: Execution vs. data transfer", Friedrich-Alexander Universität, Erlangen-Nürnberg, pp 2-16

Sources

- Schönauer W. Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers. Self-edition, 2000. URL <http://www.rz.uni-karlsruhe.de/~rx03/book>.
- Treibig J, Hager G. Introducing a performance model for bandwidth-limited loop kernels. Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J (eds.). Springer Berlin / Heidelberg, 2010; 615–624, doi: 10.1007/978-3-642-14390-8 64.
- Suleman MA, Qureshi MK, Patt YN. Feedback-driven threading: power-efficient and high performance execution of multi-threaded workloads on CMPs. SIGARCH Comput. Archit. News Mar 2008; 36(1):277– 286, doi:10.1145/1353534.1346317.
- Hoisie A, Lubeck O, Wasserman HJ. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. Int. J. High Perform. Comp. Appl. 2000; 14:330–346, doi:10.1177/109434200001400405.
- Nudd GR, Kerbyson DJ, Papaefstathiou E, Perry SC, Harper JS, Wilcox DV. Pace — A toolset for the performance prediction of parallel and distributed systems. Int. J. High Perform. Comp. Appl. 2000; 14(3):228–251, doi:10.1177/109434200001400306.
- Kerbyson DJ, Jones PW. A performance model of the Parallel Ocean Program. Int. J. High Perform. Comp. Appl. 2005; 19:261–276, doi:10.1177/1094342005056114.
- G. Hager, J. Treibig, J. Habich, and G. Wellein, "Exploring performance and power properties of modern multicore chips via simple machine models", Erlangen Regional Computing Center (RRZE) Martensstr. 1, 91058 Erlangen, Germany

Backup Slides

FMA

The FMA instruction set is an extension to the 128 and 256-bit Streaming SIMD Extensions instructions in the x86 microprocessor instruction set to perform fused multiply-add (FMA) operations.

There are two variants:

- FMA4 is supported in AMD processors starting with the Bulldozer architecture. FMA4 was realized in hardware before FMA3.
- FMA3 is supported in AMD processors starting with the Piledriver architecture and Intel starting with Haswell processors and Broadwell processors since 2014.

FLOPS

In computing, FLOPS or flops (an acronym for floating-point operations per second) is a measure of computer performance, useful in fields of scientific calculations that make heavy use of floating-point calculations. For such cases it is a more accurate measure than the generic instructions per second.

$\text{GFLOPS} = (\text{CPU Clock in GHz}) \times (\text{Number of CPU Kernels})$

FLOPS can be calculated using this equation:

$$\text{FLOPS} = \text{sockets} \times \frac{\text{cores}}{\text{socket}} \times \text{clock} \times \frac{\text{FLOPs}}{\text{cycle}}$$

Most microprocessors today can carry out 4 FLOPs per clock cycle; thus a single-core 2.5 GHz processor has a theoretical performance of 10 billion FLOPS = 10 GFLOPS.

Sockets is referring to processor chip sockets on a motherboard, in other words, how many processor chips are in use, with each chip having one or more cores on it. This equation only applies to one very specific (but common) hardware architecture and it ignores limits imposed by memory bandwidth and other constraints. In general, gigaFLOPS are not determined by theoretical calculations such as this one; instead, they are measured by benchmarks of actual performance/throughput. Because this equation ignores all sources of overhead, in the real world, one will never get actual performance that is anywhere near to what this equation predicts.

Cache entries

Data is transferred between memory and cache in blocks of fixed size, called cache lines. When a cache line is copied from memory into the cache, a cache entry is created. The cache entry will include the copied data as well as the requested memory location (now called a tag).

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache. The cache checks for the contents of the requested memory location in any cache lines that might contain that address. If the processor finds that the memory location is in the cache, a cache hit has occurred. However, if the processor does not find the memory location in the cache, a cache miss has occurred. In the case of a cache hit, the processor immediately reads or writes the data in the cache line. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

Cache performance

The proportion of accesses that result in a cache hit is known as the hit rate, and can be a measure of the effectiveness of the cache for a given program or algorithm.

Read misses delay execution because of requiring data to be transferred from memory, which is much slower than reading from the cache. Write misses may occur without such penalty, since the processor can continue execution while data is copied to main memory in the background.

Replacement policies

In order to make room for the new entry on a cache miss, the cache may have to evict one of the existing entries. The heuristic that it uses to choose the entry to evict is called the replacement policy. The fundamental problem with any replacement policy is that it must predict which existing cache entry is least likely to be used in the future. Predicting the future is difficult, so there is no perfect way to choose among the variety of replacement policies available.

One popular replacement policy, least-recently used (LRU), replaces the least recently accessed entry.

Marking some memory ranges as non-cacheable can improve performance, by avoiding caching of memory regions that are rarely re-accessed. This avoids the overhead of loading something into the cache without having any reuse. Cache entries may also be disabled or locked depending on the context.

Write policies

If data is written to the cache, at some point it must also be written to main memory; the timing of this write is known as the write policy. In a write-through cache, every write to the cache causes a write to main memory. Alternatively, in a write-back or copy-back cache, writes are not immediately mirrored to the main memory, and the cache instead tracks which locations have been written over, marking them as dirty. The data in these locations is written back to the main memory only when that data is evicted from the cache. For this reason, a read miss in a write-back cache may sometimes require two memory accesses to service: one to first write the dirty location to main memory, and then another to read the new location from memory. Also, a write to a main memory location that is not yet mapped in a write-back cache may evict an already dirty location, thereby freeing that cache space for the new memory location.

There are intermediate policies as well. The cache may be write-through, but the writes may be held in a store data queue temporarily, usually so that multiple stores can be processed together (which can reduce bus turnarounds and improve bus utilization).

Cached data from the main memory may be changed by other entities (e.g. peripherals using direct memory access (DMA) or another core in a multi-core processor), in which case the copy in the cache may become out-of-date or stale. Alternatively, when a CPU in a multiprocessor system updates data in the cache, copies of data in caches associated with other CPUs will become stale. Communication protocols between the cache managers that keep the data consistent are known as cache coherence protocols.

CPU stalls

The time taken to fetch one cache line from memory (read latency) matters because the CPU will run out of things to do while waiting for the cache line. When a CPU reaches this state, it is called a stall. As CPUs become faster compared to main memory, stalls due to cache misses displace more potential computation; modern CPUs can execute hundreds of instructions in the time taken to fetch a single cache line from main memory.

Various techniques have been employed to keep the CPU busy during this time, including out-of-order execution in which the CPU (Pentium Pro and later Intel designs, for example) attempts to execute independent instructions after the instruction that is waiting for the cache miss data. Another technology, used by many processors, is simultaneous multithreading (SMT), or—in Intel's terminology—hyper-threading (HT), which allows an alternate thread to use the CPU core while the first thread waits for required CPU resources to become available.

Instruction-level parallelism (ILP) is a measure of how many of the operations in a computer program can be performed simultaneously. The potential overlap among instructions is called instruction level parallelism.

There are two approaches to instruction level parallelism:

- Hardware
- Software

Hardware level works upon dynamic parallelism whereas, the software level works on static parallelism.

Instruction Prefetch

In computer architecture, instruction prefetch is a technique used in microprocessors to speed up the execution of a program by reducing wait states.

Modern microprocessors are much faster than the memory where the program is kept, meaning that the program's instructions cannot be read fast enough to keep the microprocessor busy. Adding a cache can provide faster access to needed instructions.

Prefetching occurs when a processor requests an instruction from main memory before it is actually needed. Once the instruction comes back from memory, it is placed in a cache. When an instruction is actually needed, the instruction can be accessed much more quickly from the cache than if it had to make a request from memory.

ECM Measurement Tools

We have used the Intel compiler Version 12.1 update 9 for compiling source codes. Hardware counter measurements were performed with the likwid-perfctr tool from the LIKWID tool suite, which, in its latest development release, can access the power information (via the RAPL interface) and the “uncore” events (i.e., L3 cache and memory/QuickPath interface) on Sandy Bridge processors.

The LIKWID suite also contains likwid-bench, a micro-benchmarking framework that makes it easy to build and run assembly language loop kernels from scratch, without the uncertainties of compiler code generation. likwid-bench was used to validate the results for some of the streaming micro-benchmarks in this work.