Applying the Roofline Model

Georg Ofenbeck

Ruedi Steinman

Victoria Caparrós Cabezas

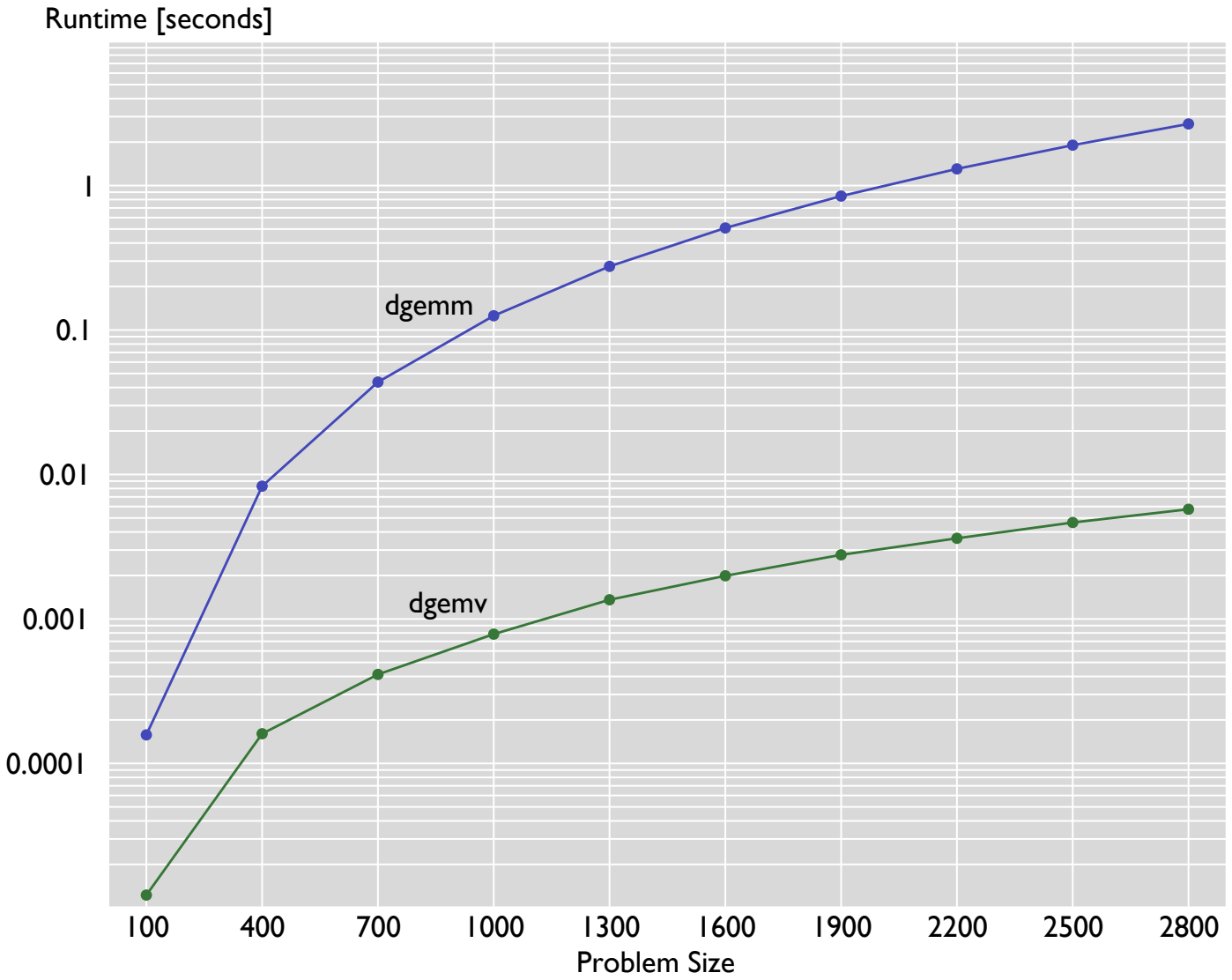
Daniele Spampinato

Markus Püschel

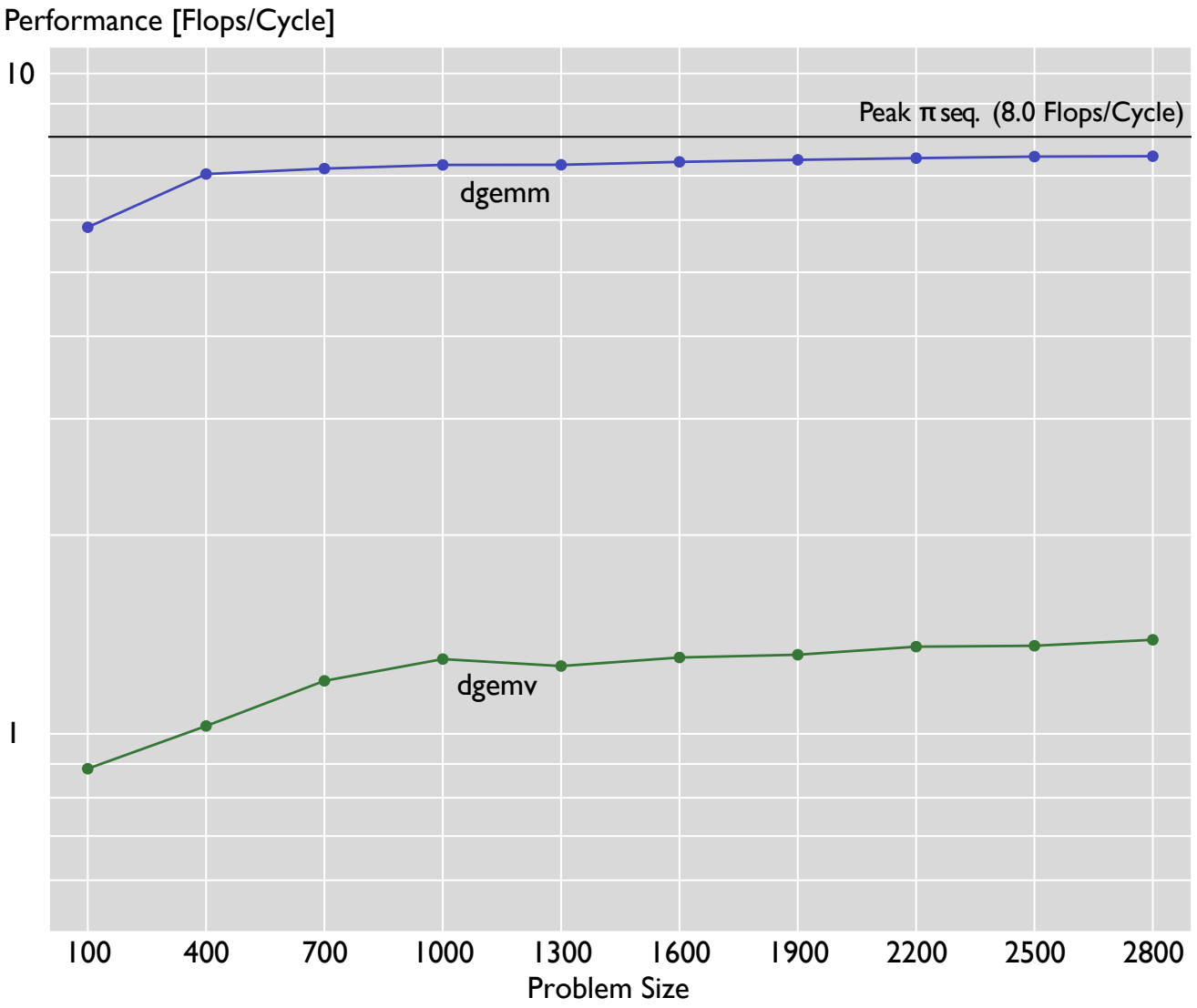
ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

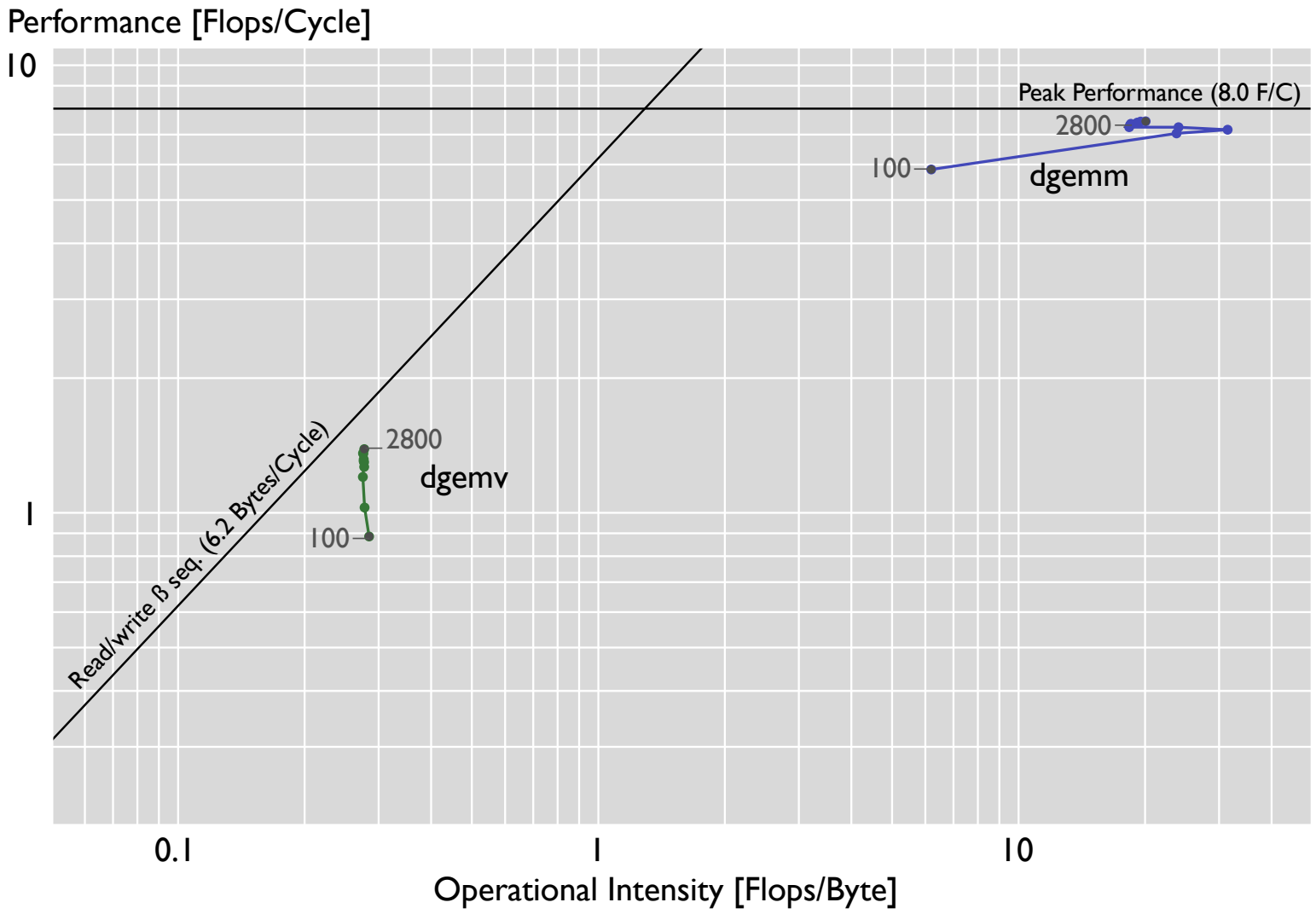
Measuring Performance — Runtime



Measuring Performance — Flops/Cycle



Measuring Performance — Roofline Plot



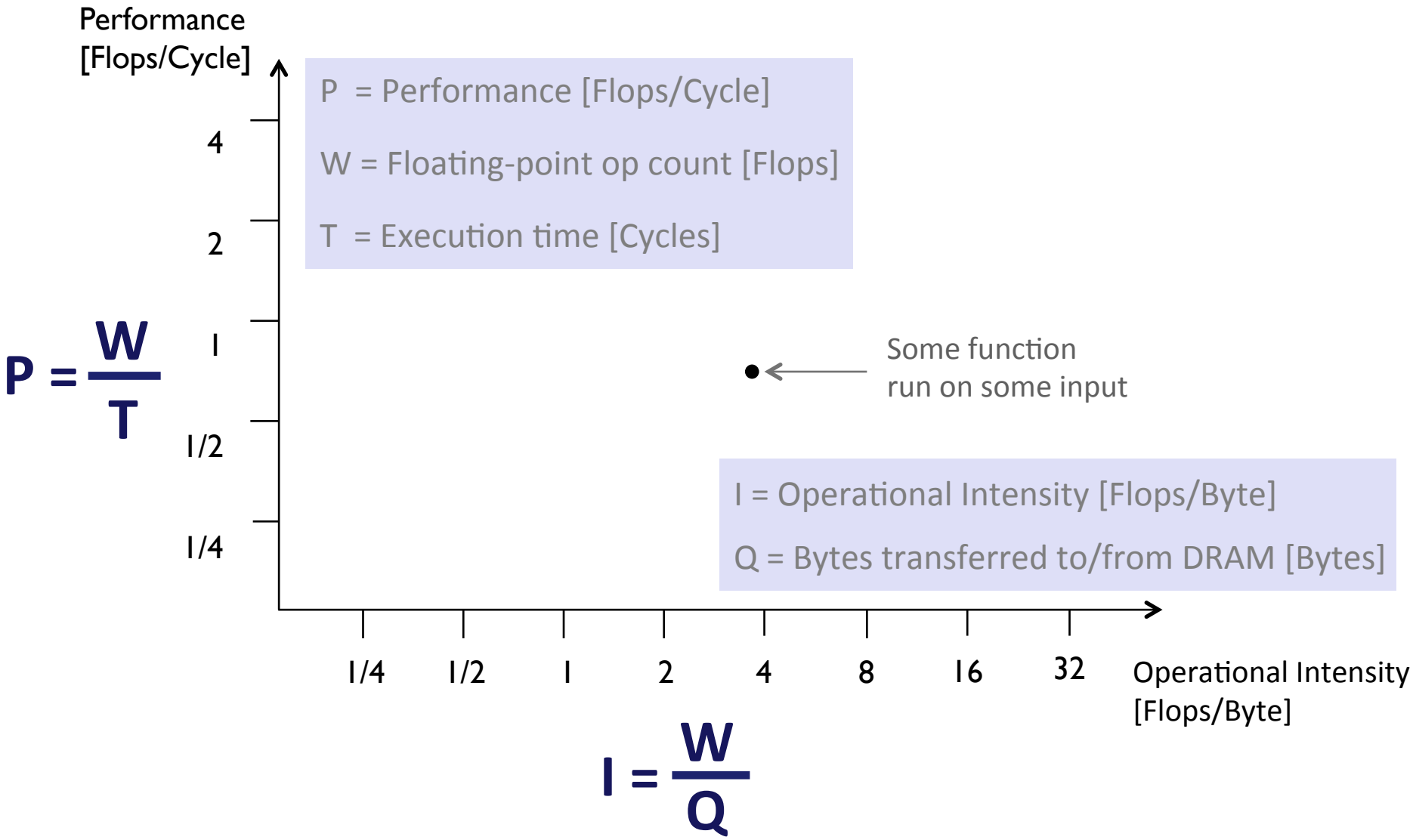
Goals:

- **Build** roofline plots with accurate measurements using hardware performance counters
- **Analyze** roofline plots to understand performance bottlenecks and guide the optimization process

Outline

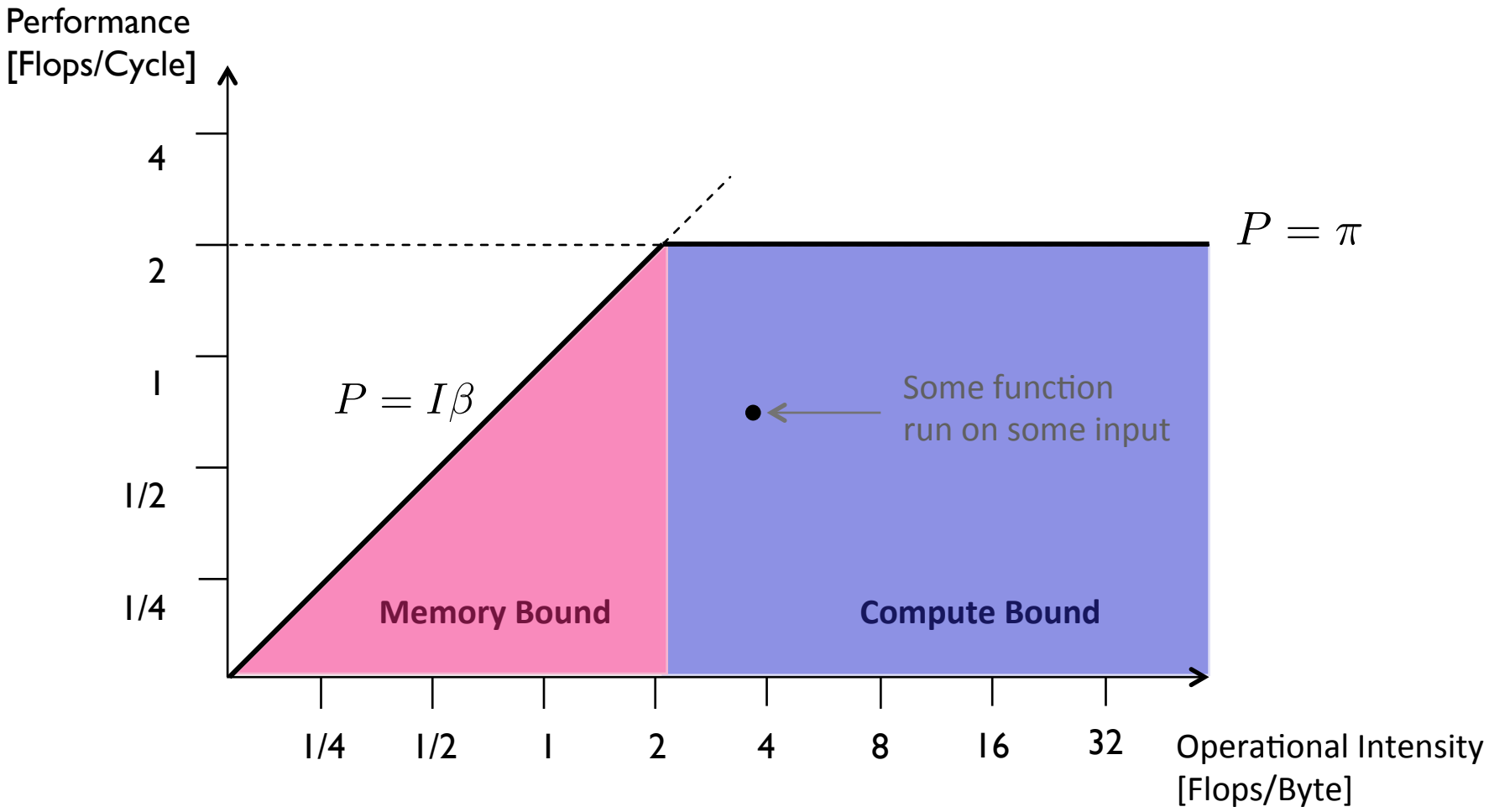
- Motivation
- Introduction to the roofline model
- How to measure P and I using hardware performance counters
- Measuring strategy
- Validation and results

Roofline Model — Application's Performance



[Williams, 2009] "Roofline: An Insightful Visual Performance Model for Multicore", S. Williams *et al.* Communications of the ACM, 2009

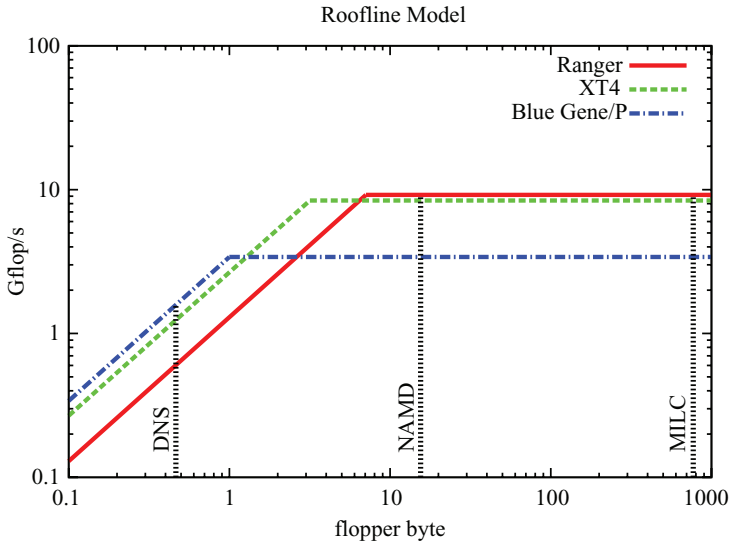
Roofline Model — Performance Bounds



[Williams, 2009] "Roofline: An Insightful Visual Performance Model for Multicore", S. Williams *et al.* Communications of the ACM, 2009

Users of the Roofline Model

[Bhatele, 2010]



[Rossinelli, 2011]

$$F(c) = \underbrace{2\cdot 4}_{\text{localize grid points}} + \underbrace{2\cdot 4}_{\text{set-up } \alpha_0,\alpha_1} + \underbrace{2\cdot 4\cdot 3}_{\text{Horner eval.}} + \underbrace{4^2\cdot 3\cdot c}_{\text{filtering}} = 48\cdot c + 40 \text{ [FLOP]}.$$

$$M^{\text{CPU}}(c) = (\underbrace{2}_{\text{particle location}} + \underbrace{4\cdot c}_{\text{not in-cache}} + \underbrace{2\cdot c}_{\text{write-allocate}}) \cdot 4 = 24\cdot c + 8 \text{ [bytes]}.$$

From these considerations, we estimate the bounds of the operational intensity for grid–particle interpolation (with $c=2$) to be $0.9 < r_{\text{CPU}} < 2.5$ on the CPU and $0.9 < r_{\text{GPU}} < 2.9$ on the GPU. Because the average operational intensity could lie anywhere between the lowest and the highest possible values, we assume $r_{\text{CPU}} = 1.5$ ($r_{\text{CPU/NT}} = 1.88$) and $r_{\text{GPU}} = 1.9$.

Roofline model traditionally used with back-of-the-envelop calculations

[Bhatele , 2010] "Understanding Application Performance via Micro-benchmarks on Three Large Supercomputers: Intrepid, Ranger and Jaguar", Abhinav Bhatele *et al.* Internaltional Journal of High Performance Computing Applications, 2010

[Rossinelli , 2011] "Mesh-particle interpolations on graphics processing units and multicore central processing units", D. Rossinelli *et al.* Phil. Trans. R. Soc, 2011

Measuring T, W and Q

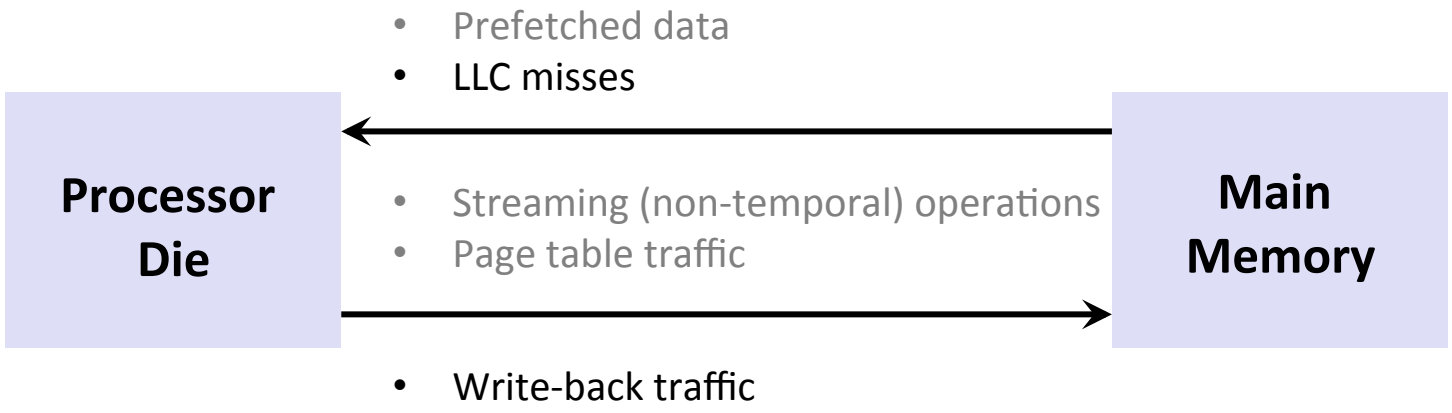
$$P = \frac{W}{T} \qquad I = \frac{W}{Q}$$

- **Measuring runtime T**
  - Time Stamp Counter
- **Measuring work W**
  - Composed from scalar and SIMD operations

$$W = \text{Scalar\_single} + \text{SSE\_single} \times 4 + \text{AVX\_single} \times 8$$

$$W = \text{Scalar\_double} + \text{SSE\_double} \times 2 + \text{AVX\_double} \times 4$$

- **Measuring memory traffic Q**



… But Measuring is Hard

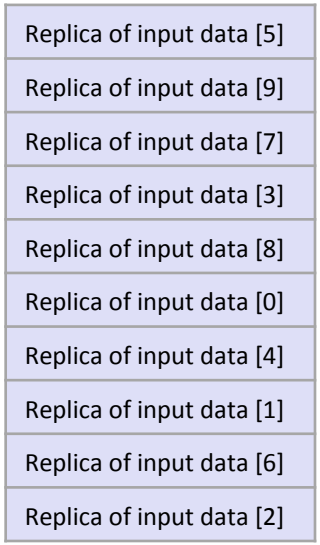
- Dynamic Frequency Scaling
- Dead Code Elimination
- Initialization
- Alignment
- Asynchronous calls
- Hardware Prefetcher

If any of these factors is not controlled, measurements become meaningless

Measurement Strategy

```
for (nr_of_repeats){
    start_measure()
    for (nr_of_runs){
      target_code(replica_of_data)
    }
    stop_measure()
}
```

Allocated memory (e.g., nr_runs = 10)



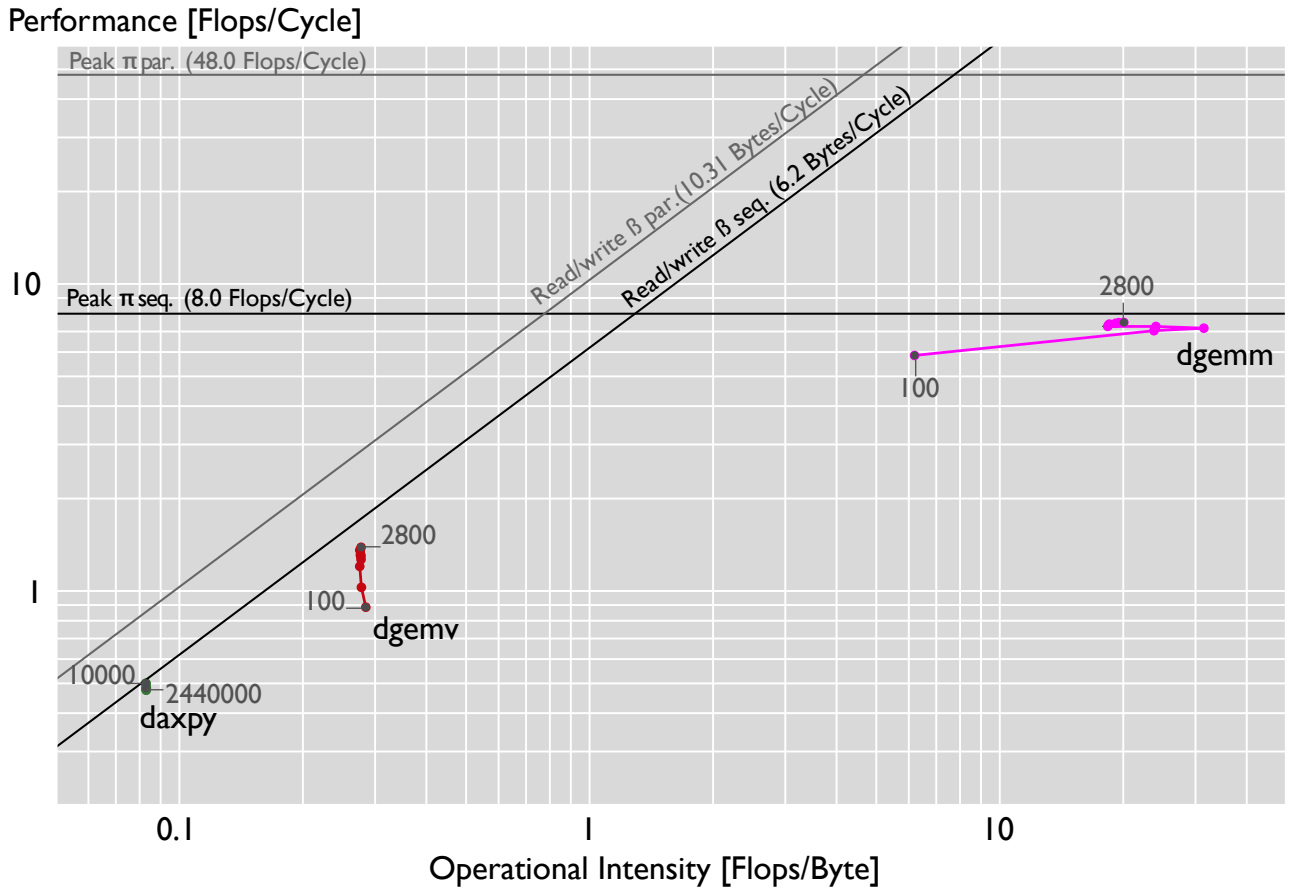
- **Repeat the execution (nr_of_repeats)**
  - Median, 25% and 75% percentile
- **Run the code several times until execution gets long enough (nr_runs)**
  - Cold cache measurements require special treatment

Experimental Setup

- **Intel PCM for accessing hardware performance counters**
- **Different microarchitectures and operating systems**

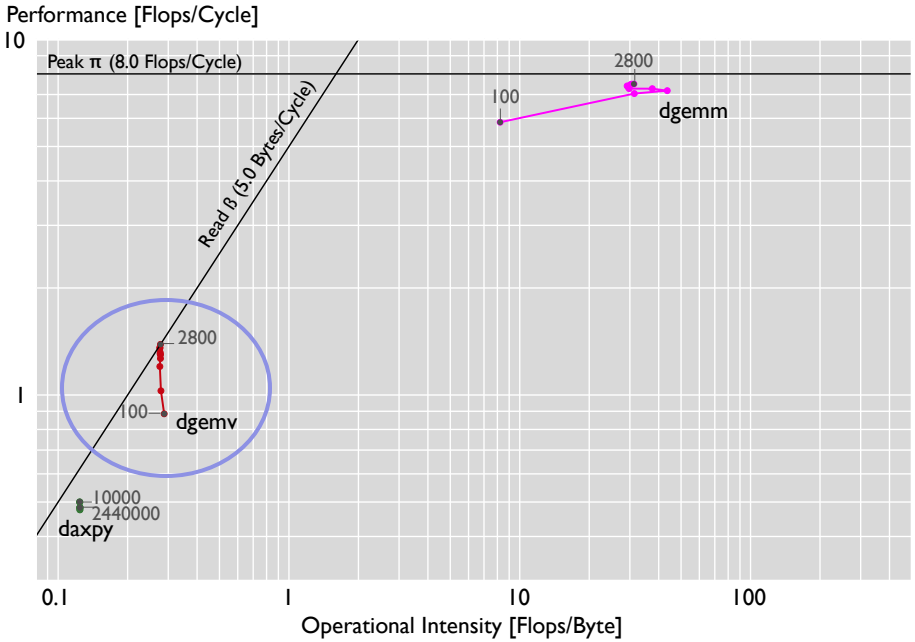
CPU Model	Xeon E5-2660	Xeon X5680	Core i7-3930K
Microarch.	Sandy Bridge EP	Westmere EP	Sandy Bridge E
ISA	AVX	SSE 4.2	AVX
Cores	8	6	6
Sockets	2	2	1
Frequency [GHz]	2.2	3.3	3.2
π **per core [Flops/cycle]**	8	4	8
β **one/all cores [Bytes/cycle]**	6.7/14.1	6.7/13.9	6.2/10.4
Operating System	RHEL Server 6	RHEL Server 6	Ubuntu 12.10 Windows 7

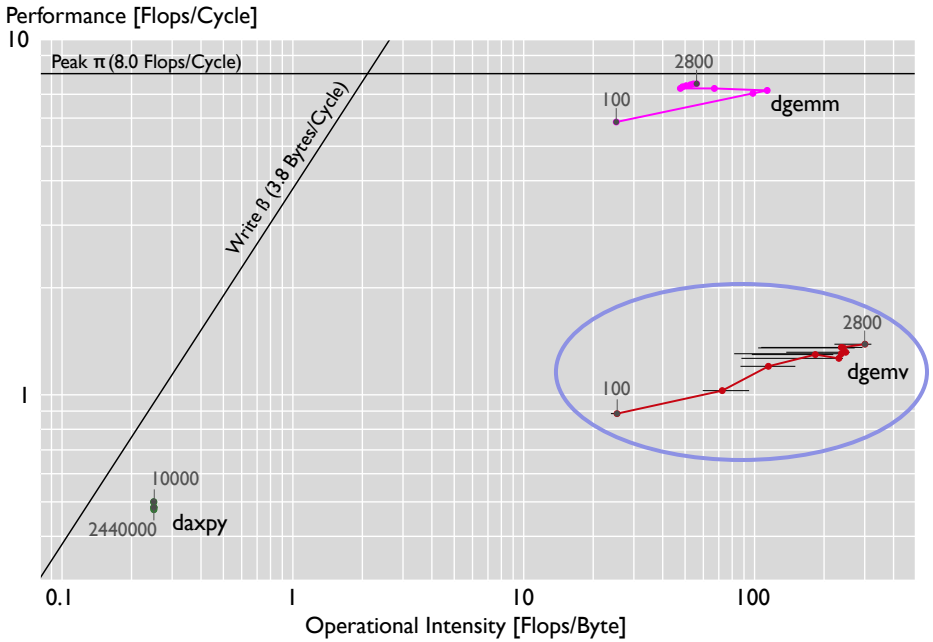
BLAS Overview



BLAS function		$W(n)$	$Q_r(n)$	$Q_w(n)$	$Q(n) = Q_{r+w}(n)$	$I_r(n)$	$I(n) = I_{r+w}(n)$
daxpy	$\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$	$= 2n$	$\geq 16n$	$\geq 8n$	$\geq 24n$	$\leq \frac{1}{8}$	$\leq \frac{1}{12}$
dgemv	$\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$	$= 2n^2 + 2n$	$\geq 8n^2 + 16n$	$\geq 8n$	$\geq 8n^2 + 24n$	$\leq \frac{n+1}{4n+8} \approx \frac{1}{4}$	$\leq \frac{n+1}{4n+12} \approx \frac{1}{4}$
dgemm	$C \leftarrow \alpha AB + \beta C$	$= 2n^3 + 2n^2$	$\geq 24n^2$	$\geq 8n^2$	$\geq 32n^2$	$\leq \frac{n+1}{12}$	$\leq \frac{n+1}{16}$

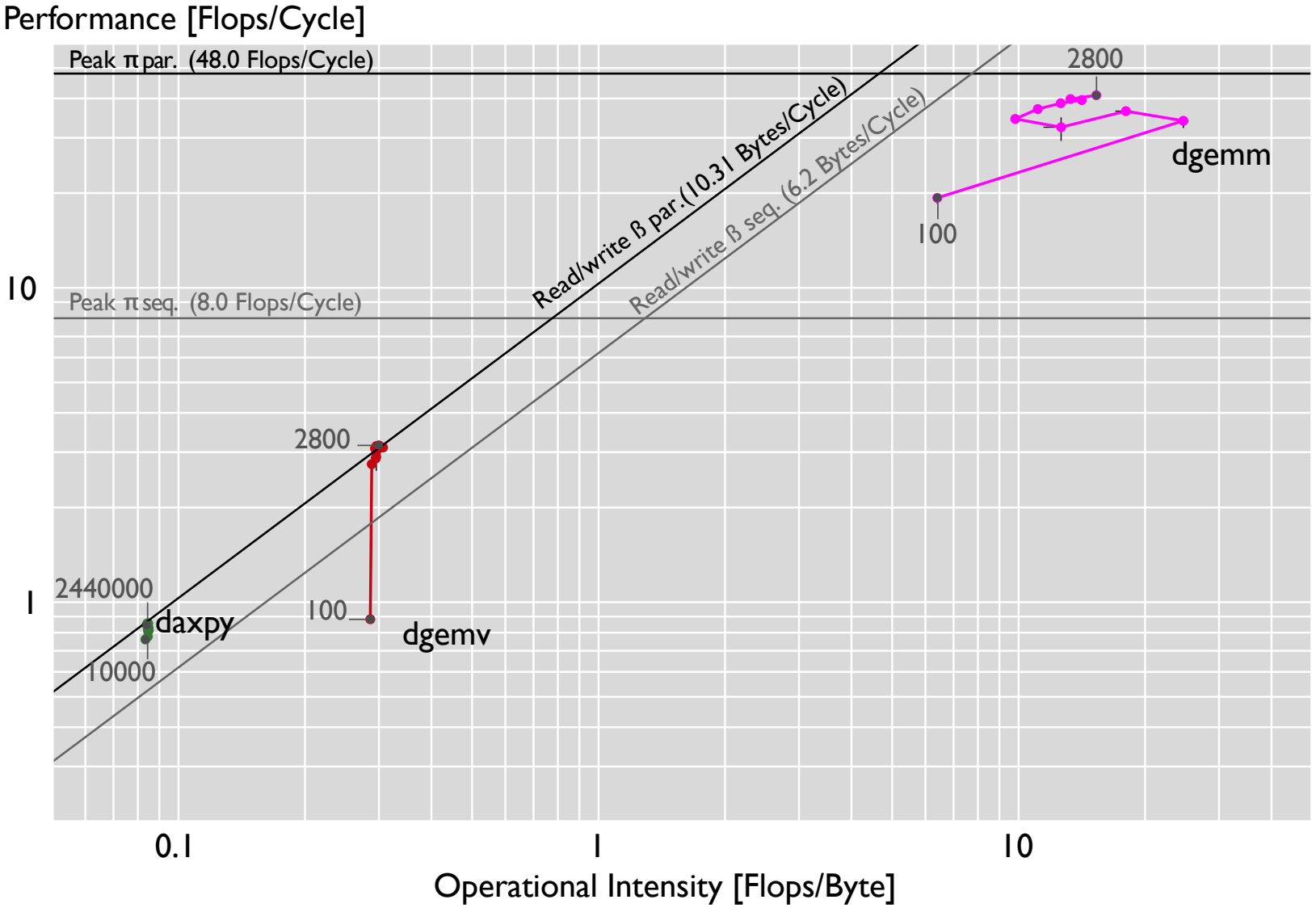
BLAS Overview — Read/Write-Only BW



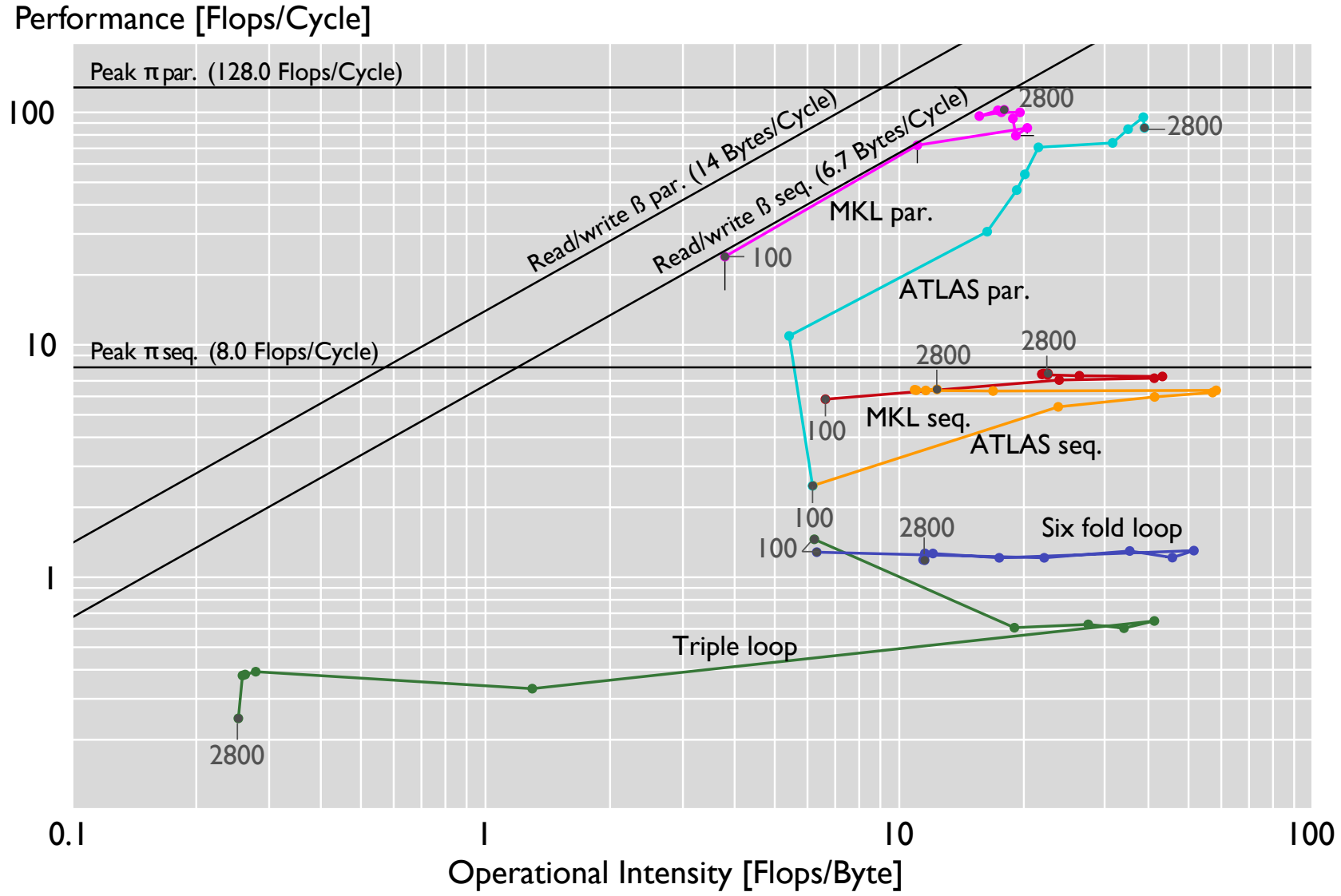


BLAS function		$W(n)$	$Q_r(n)$	$Q_w(n)$	$Q(n) = Q_{r+w}(n)$	$I_r(n)$	$I(n) = I_{r+w}(n)$
daxpy	$\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$	$= 2n$	$\geq 16n$	$\geq 8n$	$\geq 24n$	$\leq \frac{1}{8}$	$\leq \frac{1}{12}$
dgemv	$\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$	$= 2n^2 + 2n$	$\geq 8n^2 + 16n$	$\geq 8n$	$\geq 8n^2 + 24n$	$\leq \frac{n+1}{4n+8} \approx \frac{1}{4}$	$\leq \frac{n+1}{4n+12} \approx \frac{1}{4}$
dgemm	$C \leftarrow \alpha AB + \beta C$	$= 2n^3 + 2n^2$	$\geq 24n^2$	$\geq 8n^2$	$\geq 32n^2$	$\leq \frac{n+1}{12}$	$\leq \frac{n+1}{16}$

BLAS Overview — Parallel

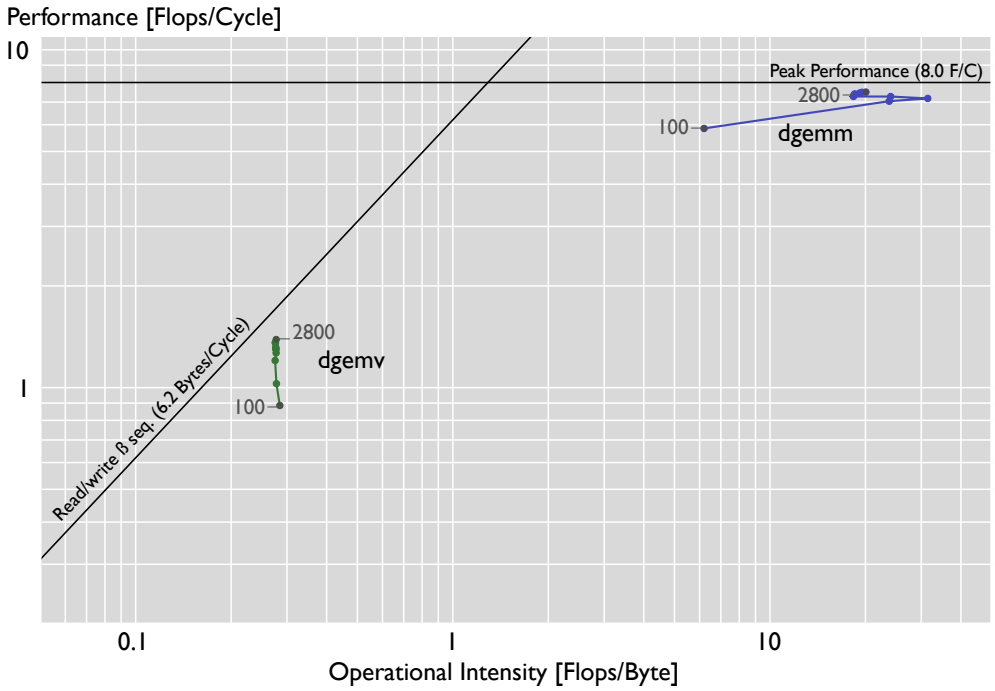


MMM Overview — Optimization Study



Conclusion

- New insights into performance bottlenecks
- Robust measurement strategy required



- Yet, it is a model and has some limitations
  - Assumes complete overlap of computation and communication
  - Does not consider additional bottlenecks

Backup Slides

Performance Counters Table

Event	Event Mask Mnemonic
Flops	
Sandy / Ivy Bridge	
Scalar single	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE
SSE single	FP_COMP_OPS_EXE.SSE_PACKED_SINGLE
AVX single	SIMD_FP_256.PACKED_SINGLE
Scalar double	FP_COMP_OPS_EXE.SSE_FP_SCALAR_DOUBLE
SSE double	FP_COMP_OPS_EXE.SSE_PACKED_DOUBLE
AVX double	SIMD_FP_256.PACKED_DOUBLE
Westmere	
Scalar	FP_COMP_OPS_EXE.SSE_FP_SCALAR
SSE	FP_COMP_OPS_EXE.SSE_FP_PACKED
Memory ops	
Sandy / Ivy Bridge	
Cache lines reads	UNC_CBO_CACHE_LOOKUP.I
	UNC_CBO_CACHE_LOOKUP.ANY_REQUEST_FILTER
Cache lines writes	UNC_ARB_TRK_REQUEST.EVICTIONS
Westmere-EP	
Cache lines reads	UNC_QMC_NORMAL_READS.ANY
Cache lines writes	UNC_QMC_WRITES.FULL.ANY
Sandy Bridge-EP	
Cache lines reads	UNC_IMC_NORMAL_READS.ANY
Cache lines writes	UNC_IMC_WRITES.FULL.ANY
Timers	
Core Cycles	UnHalted Core Cycles
Reference Cycles	UnHalted Reference Cycles
Time stamp counter	IA32_TIME_STAMP_COUNTER