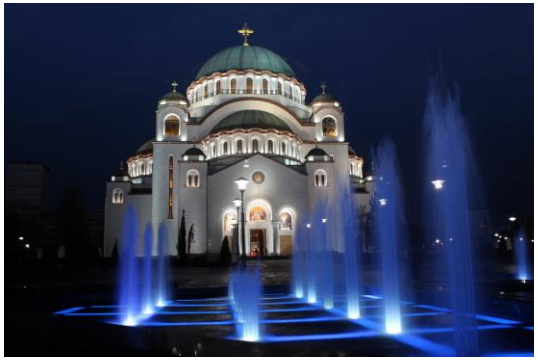
Cache-Aware Roofline Model:
Performance, Power and Energy-Efficiency

Aleksandar Ilić,

Frederico Pratas and Leonel Sousa

INESC-ID/IST,
Universidade de Lisboa, Portugal

Introduction



THIS IS ALL ABOUT BUILDING THE ROOFS

Motivation

HITTING THE ROOF

GETTING THE MAXIMUM

: ARCHITECTURES VS. APPLICATIONS

- Multi-core CPUs, accelerators, diverse compute capabilities, complex memory hierarchy…
- Complex applications, different compute and memory requirements
- Optimize and characterize applications for specific architectures

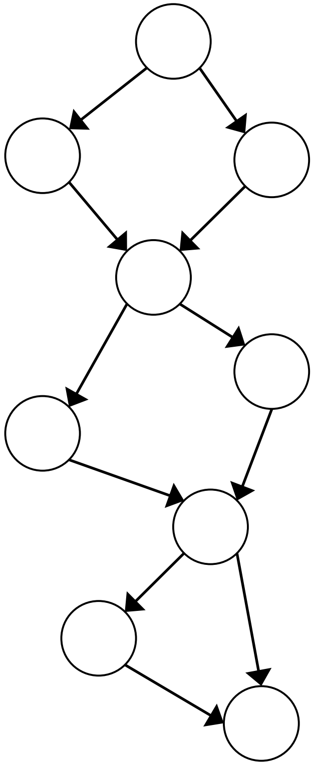
MAXIMUM ATTAINABLE PERFORMANCE VS. POWER/ENERGY CONSUMPTION

⇒ HOW FAR CAN WE GO?

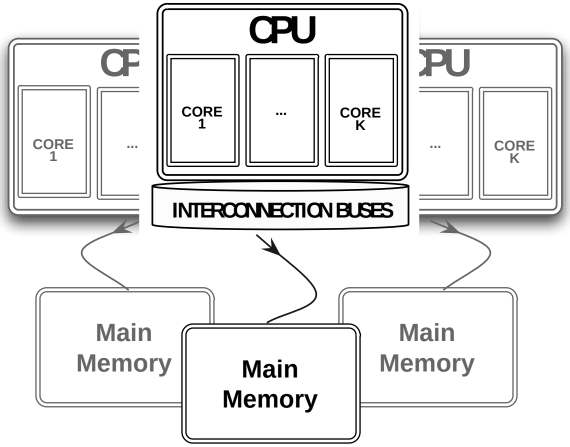
⇒ WHAT ARE THOSE MAXIMUMS FOR PERFORMANCE, POWER, ENERGY, EFFICIENCY…?

Reaching the Maximum via Scheduling

Static DAG-based Scheduling



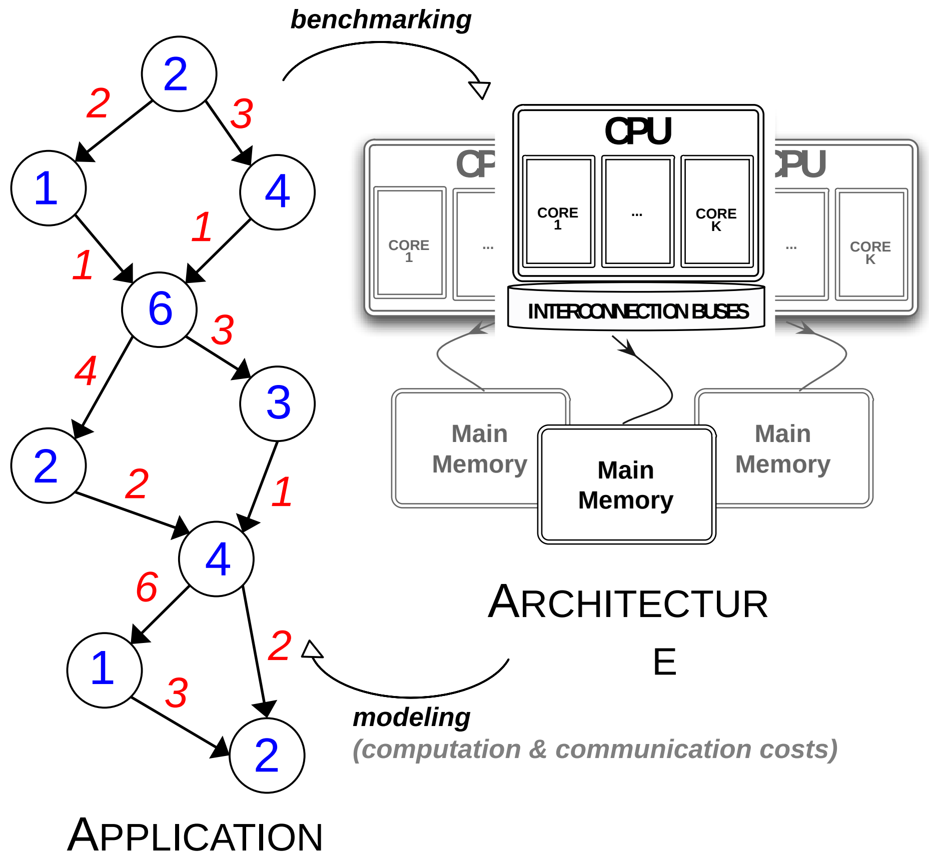
Application



Architectur
E

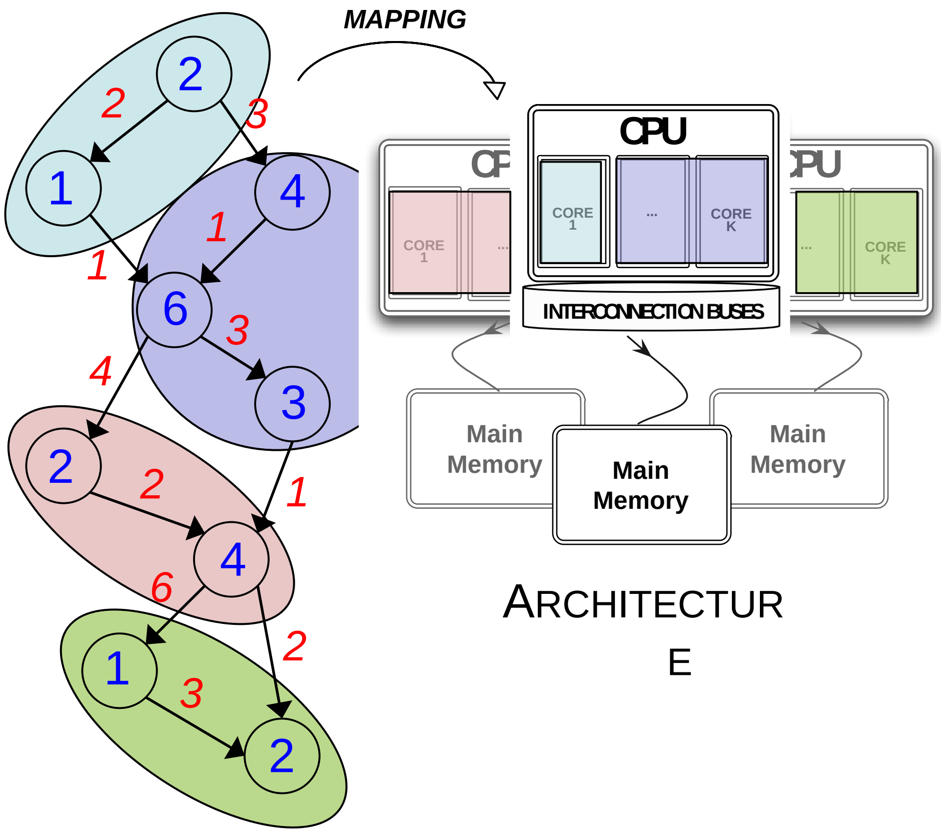
Reaching the Maximum via Scheduling

STATIC DAG-BASED SCHEDULING



Reaching the Maximum via Scheduling

STATIC DAG-BASED SCHEDULING

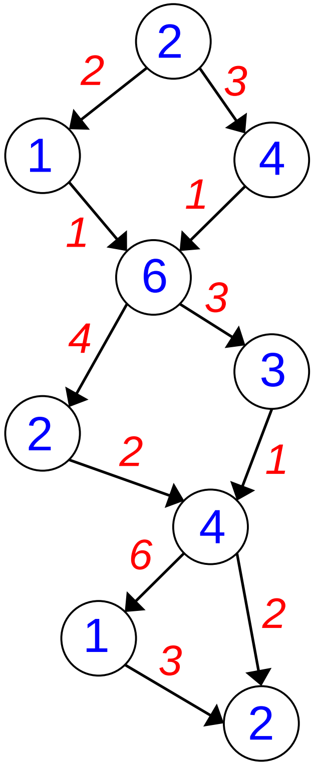


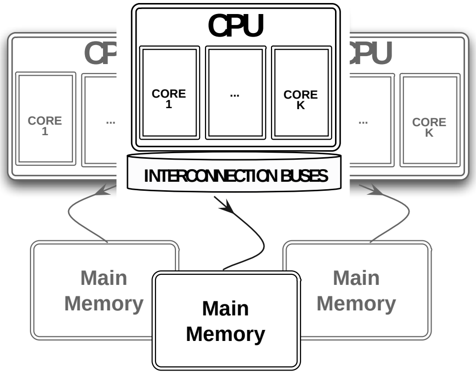
APPLICATION

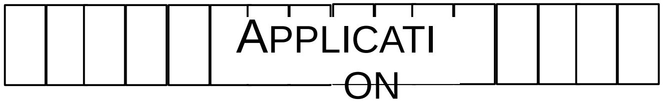
Reaching the Maximum via Scheduling

Static DAG-based Scheduling

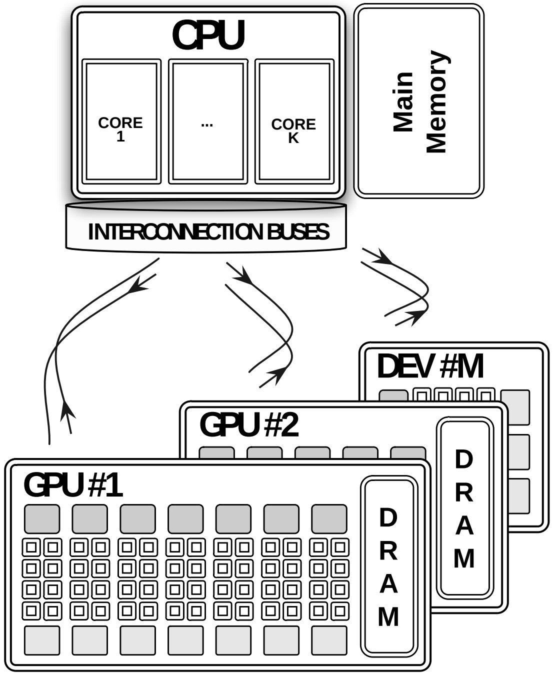
Dynamic DLT-based Scheduling







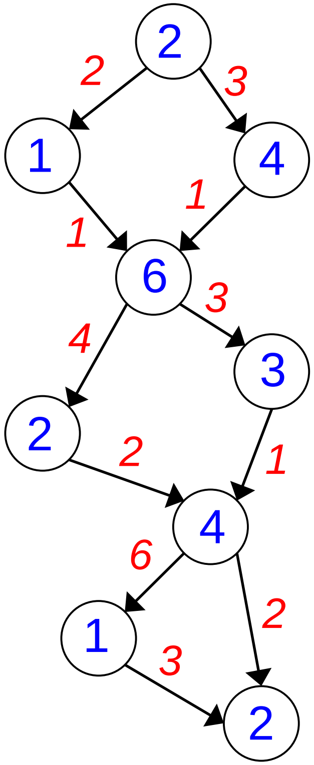
APPLICATI
ON

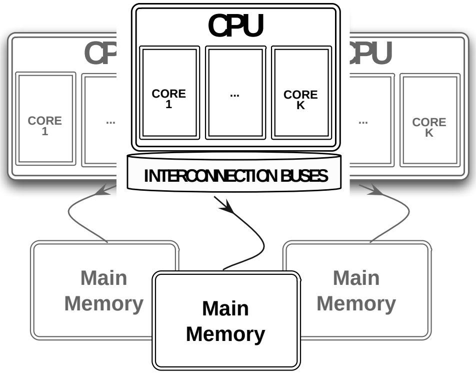


ARCHITECTUR
E

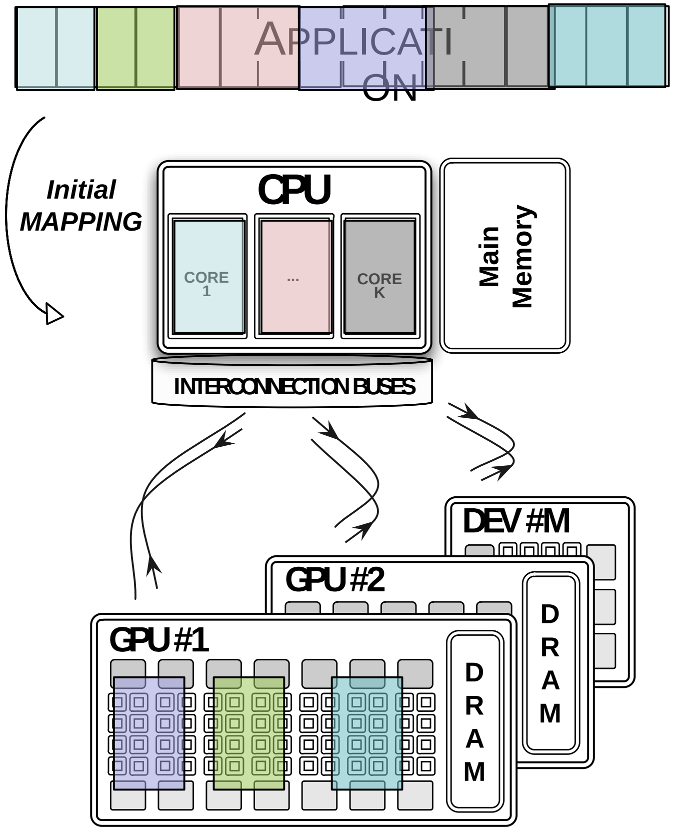
Reaching the Maximum via Scheduling

STATIC DAG-BASED SCHEDULING





DYNAMIC DLT-BASED SCHEDULING

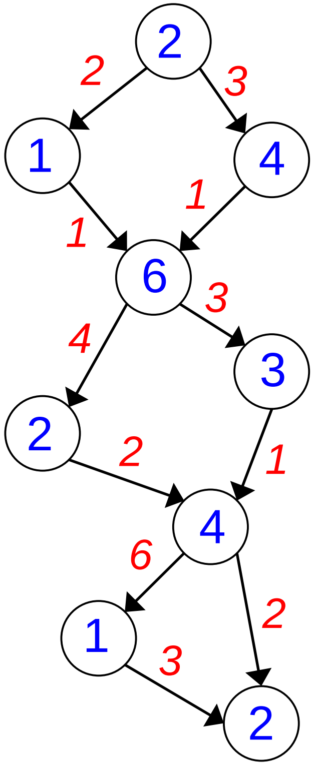


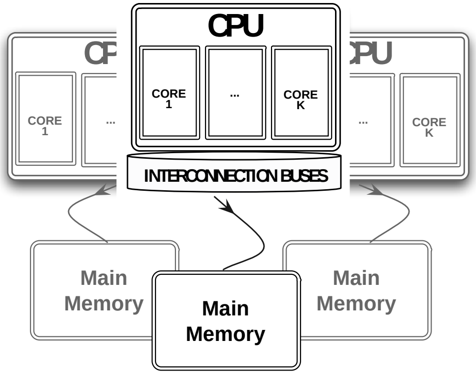
ARCHITECTUR
E

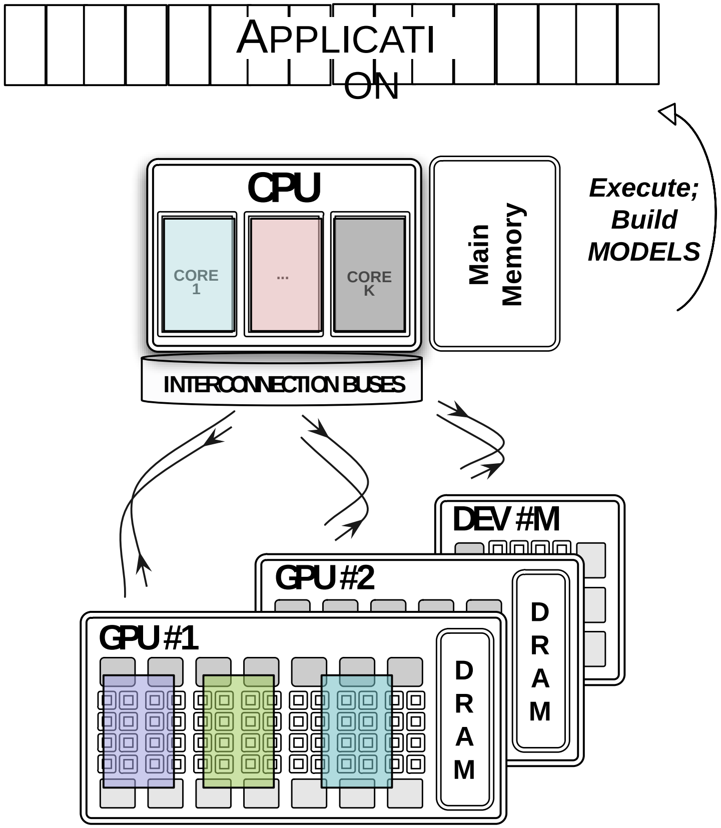
Reaching the Maximum via Scheduling

Static DAG-based Scheduling

Dynamic DLT-based Scheduling





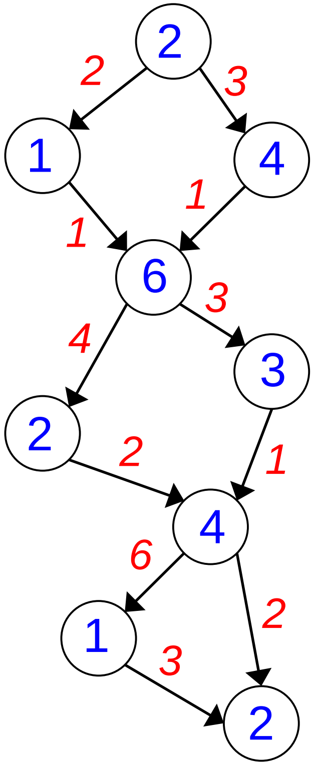


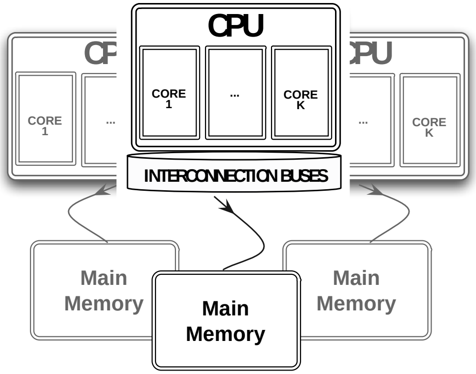
ARCHITECTUR

E

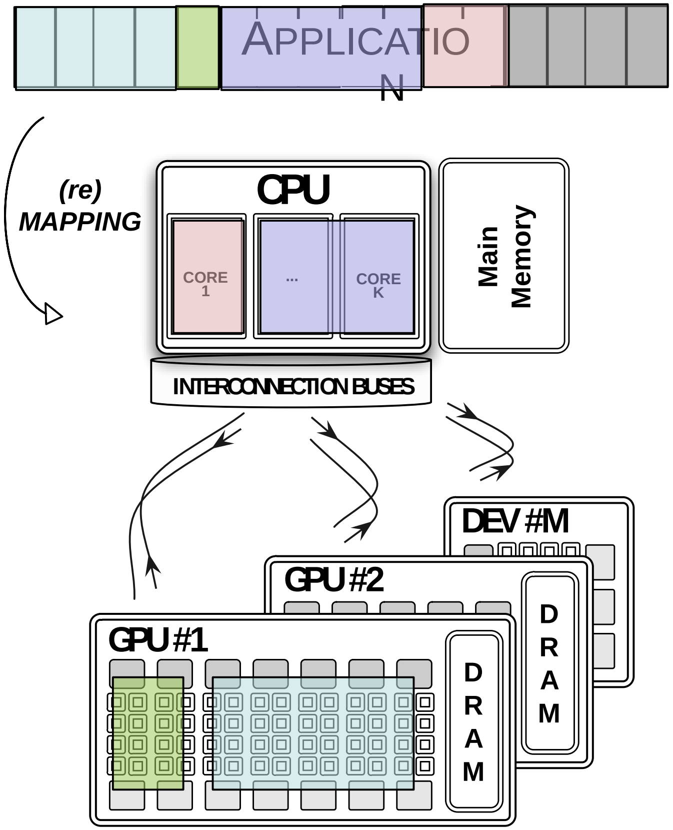
Reaching the Maximum via Scheduling

Static DAG-based Scheduling





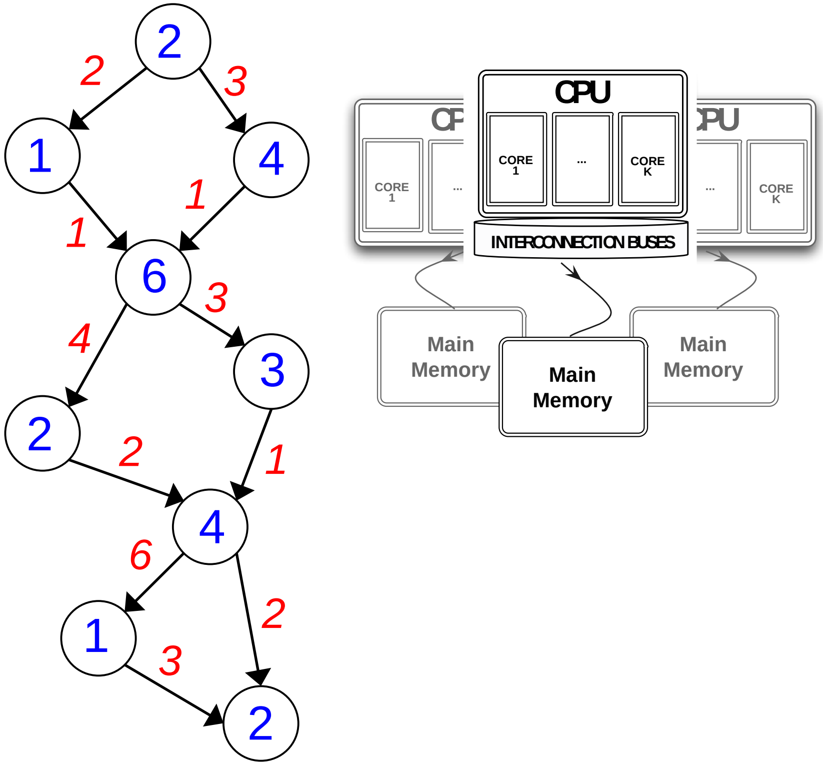
Dynamic DLT-based Scheduling



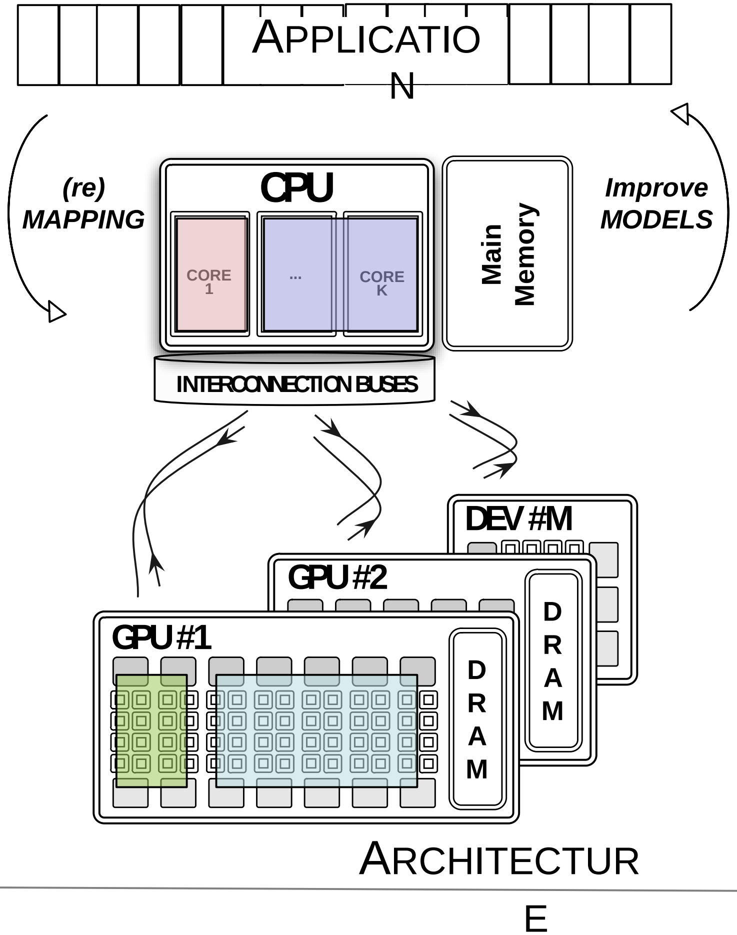
Architectur
e

Reaching the Maximum via Scheduling

Static DAG-based Scheduling

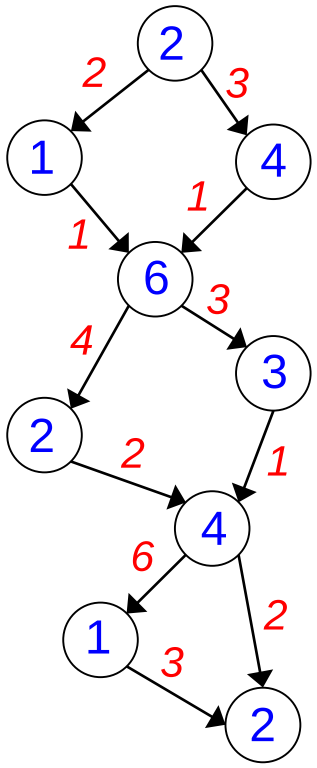


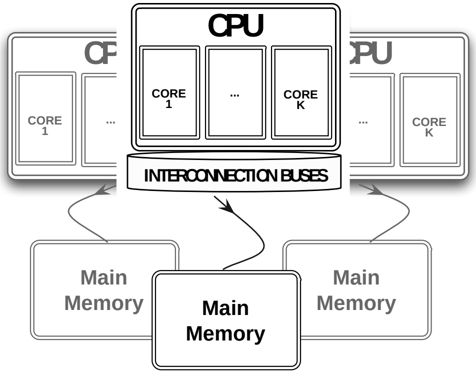
Dynamic DLT-based Scheduling



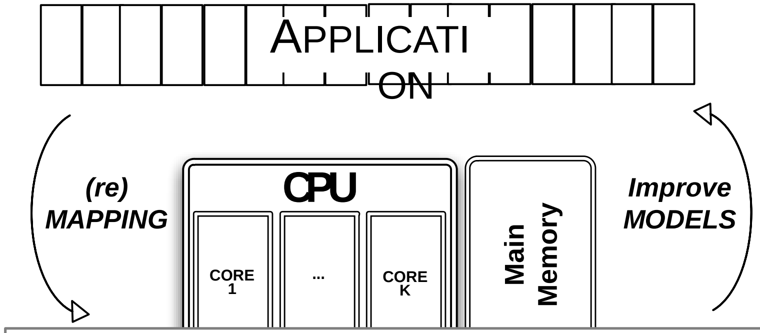
Reaching the Maximum via Scheduling

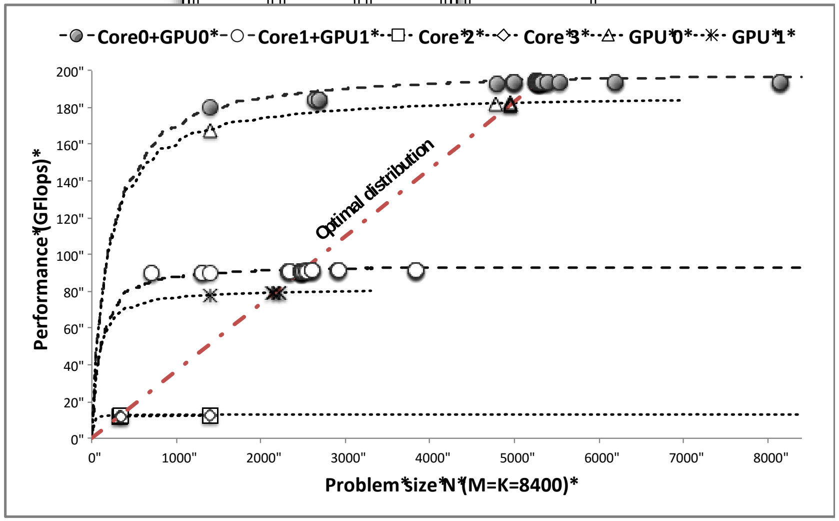
Static DAG-based Scheduling





Dynamic DLT-based Scheduling



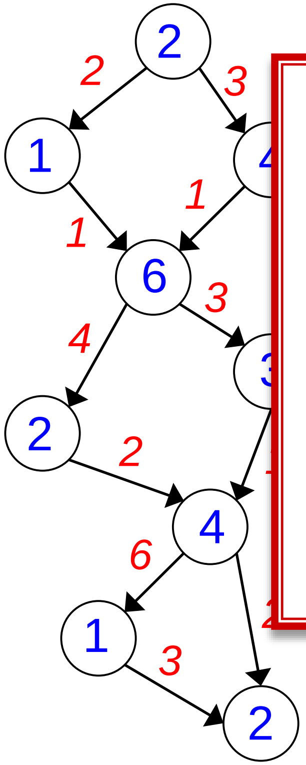


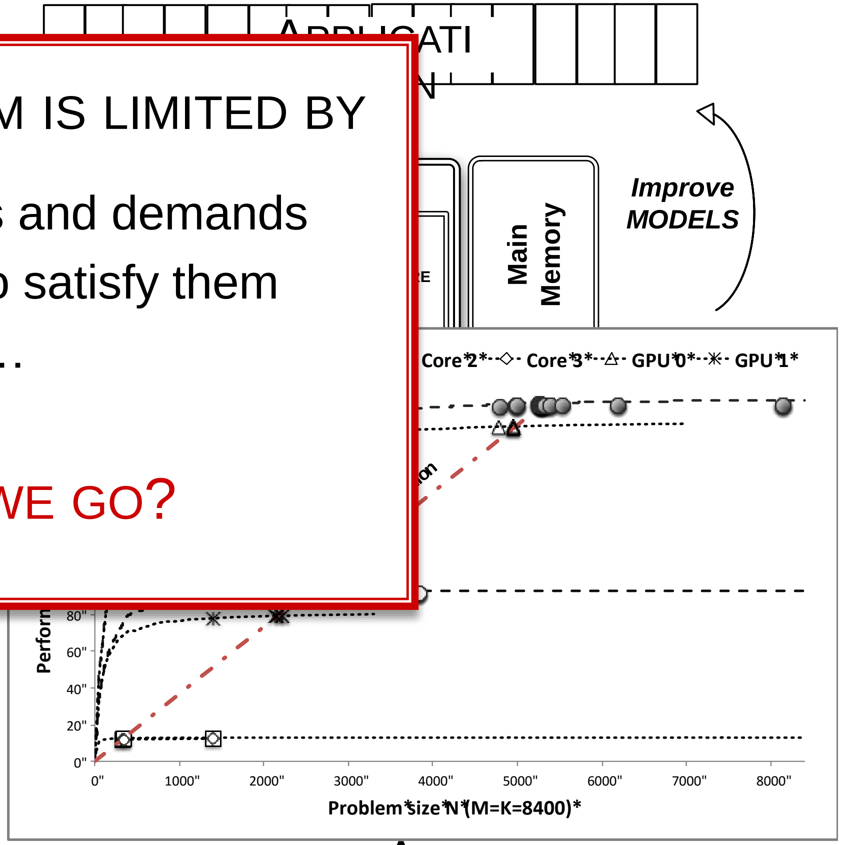
ARCHITECTUR

E

Reaching the Maximum via Scheduling

Static DAG-based Scheduling Dynamic DLT-based Scheduling





Architectur
E

Reaching the Maximum is limited by

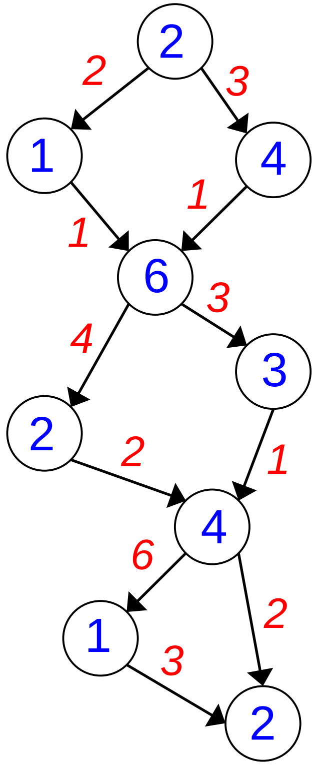
- Application characteristics and demands
- Architecture capabilities to satisfy them
- Trade-offs, assumptions …

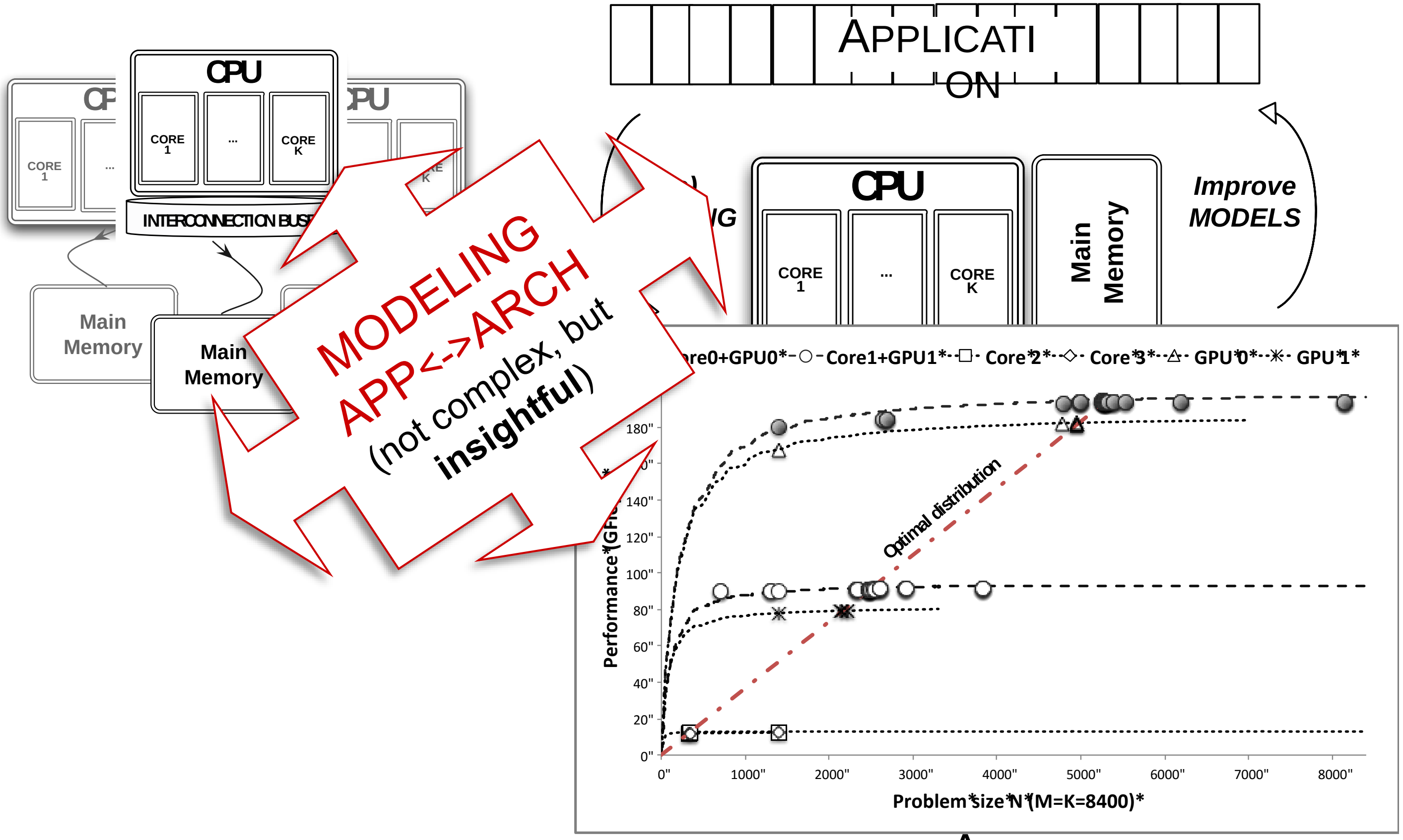
How far can we go?

Reaching the Maximum via Scheduling

Static DAG-based Scheduling

Dynamic DLT-based Scheduling





ARCHITECTURE

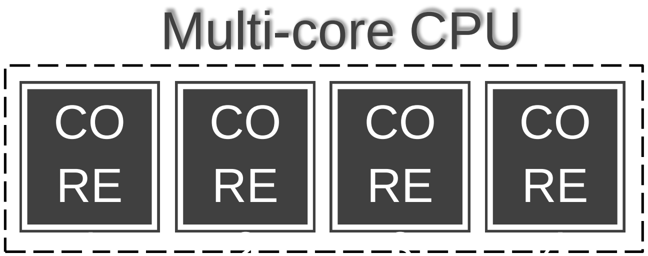
Outline

THE CACHE-AWARE ROOFLINE MODEL:

- PERFORMANCE*
- POWER
- EFFICIENCY

**A. Ilic, F. Pratas and L. Sousa “Cache-ware Roofline Model: Upgrading the Loft”, IEEE Computer Architecture Letters (2013)*

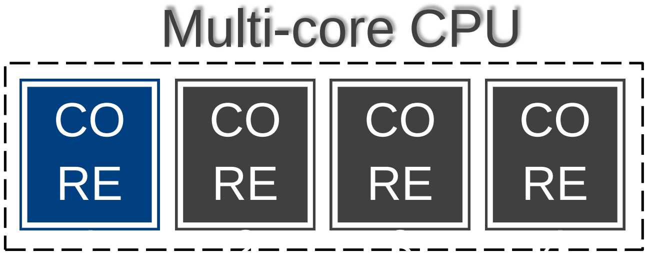
Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

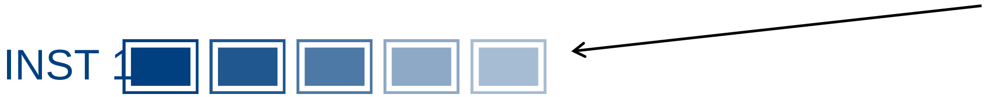
- PARALLELISM ACROSS THE PROCESSING CORES

Multi-core Architectures

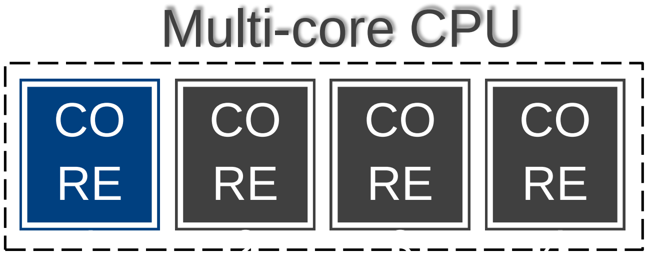


SEVERAL (IDENTICAL) PROCESSING CORES

- Parallelism across the processing cores
- INSTRUCTION LEVEL PARALLELISM (PIPELINING)

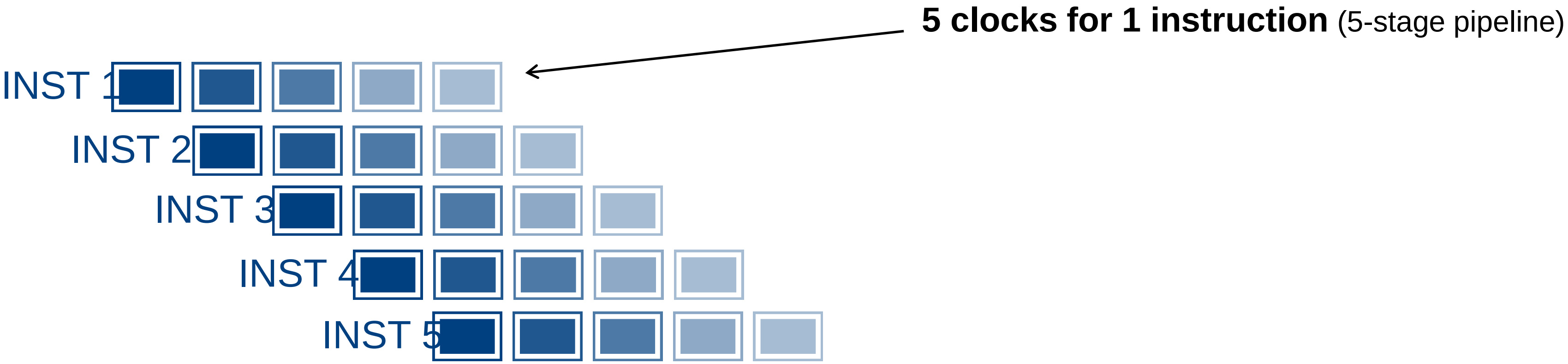


5 clocks for 1 instruction (5-stage pipeline)

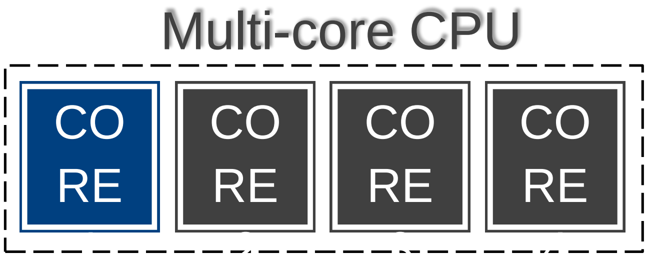


Several (Identical) Processing Cores

- Parallelism across the processing cores
- INSTRUCTION LEVEL PARALLELISM (PIPELINING)

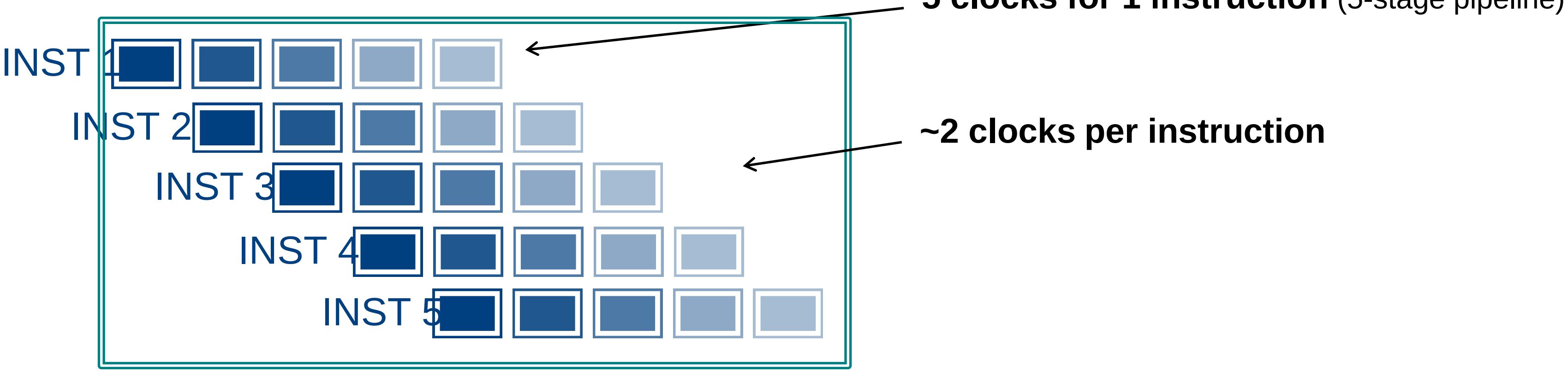


Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

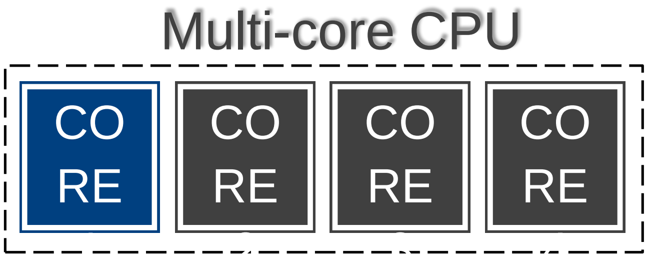
- Parallelism across the processing cores
- INSTRUCTION LEVEL PARALLELISM (PIPELINING)



5 clocks for 1 instruction (5-stage pipeline)

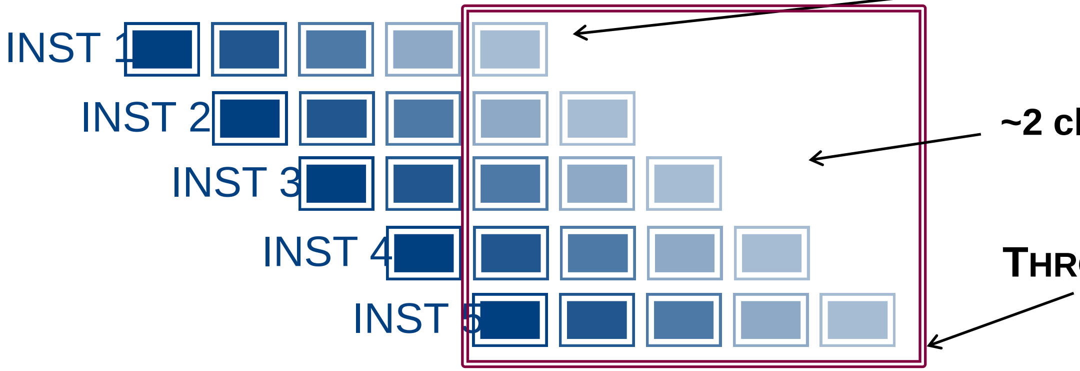
~2 clocks per instruction

Multi-core Architectures



Several (Identical) Processing Cores

- Parallelism across the processing cores
- INSTRUCTION LEVEL PARALLELISM (PIPELINING)

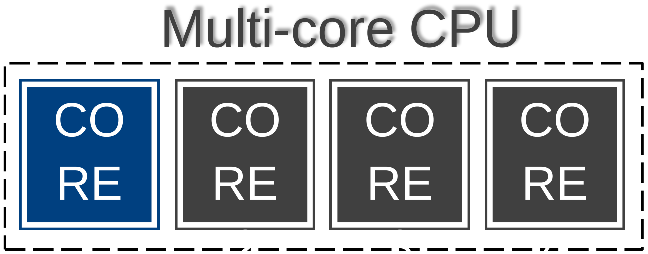


5 clocks for 1 instruction (5-stage pipeline)

~2 clocks per instruction

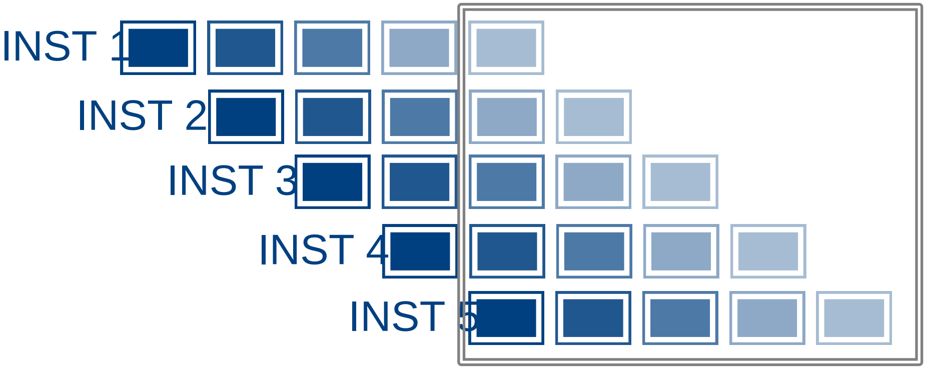
Throughput: 1 instruction per 1 clock

Multi-core Architectures



Several (identical) Processing Cores

- Parallelism across the processing cores
- INSTRUCTION LEVEL PARALLELISM (PIPELINING)

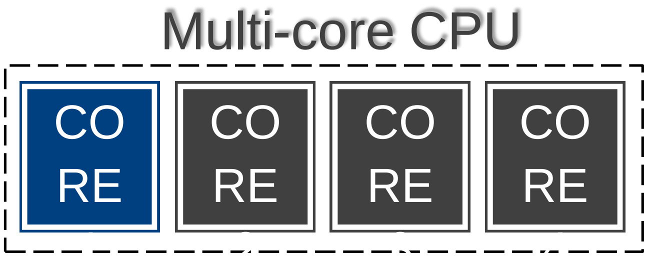


Hands-on: Ivy Bridge at 3.5 GHz (i7 3770K)

- Double-precision Floating-point operation (FLOP)
  - Multiplication(MUL) or addition (ADD)
- **Throughput**: 1 instruction/clock

Instruction Type	FLOPS per Instr.	Performance (GFLOPS/s)	
		1 Core	4 Cores
64 bits	1	**3.5** (1flop x 3.5GHz)	**14**
128 bits (SSE)	2	**7**	**28**
256 bits (AVX)	4	**14**	**56**

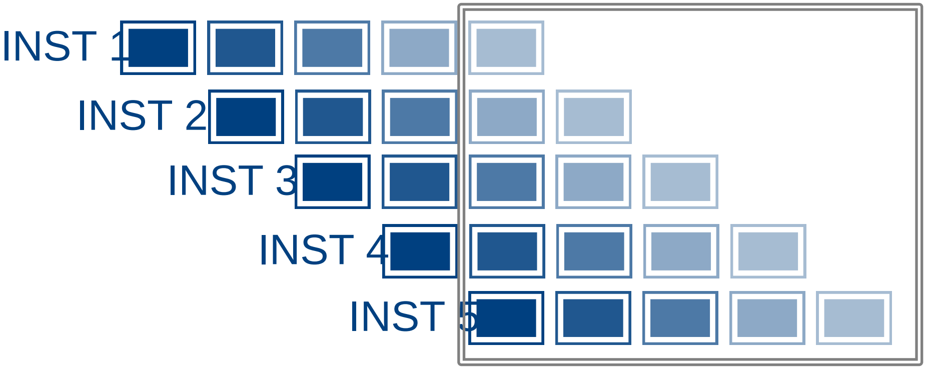
Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

– Parallelism across the processing cores

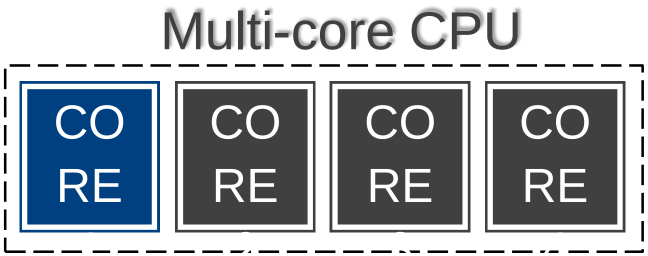
– INSTRUCTION LEVEL PARALLELISM (PIPELINING)



HANDS-ON: IVY BRIDGE AT 3.5 GHZ (I7 3770K)

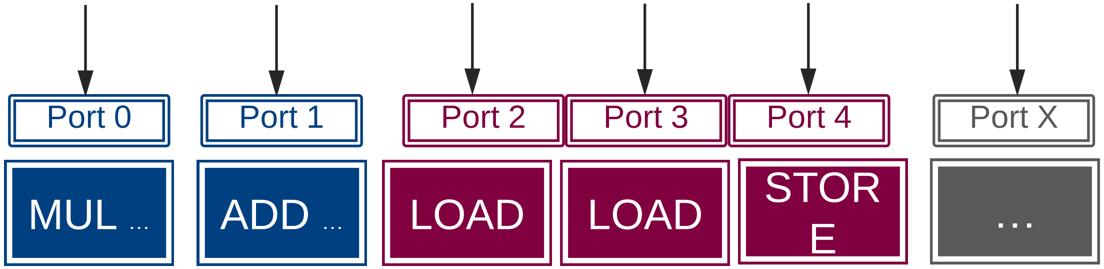
- Double-precision Floating-point operation (FLOP)
  - Multiplication(MUL) or addition (ADD)
- **Throughput**: 1 instruction/clock

Instruction Type	FLOPS per Instr.	Performance (GFLOPS/s)	
		1 Core	4 Cores
64 bits	1	**3.5** (1flop x 3.5GHz)	**14**
128 bits (SSE)	2	**7**	**28**
256 bits (AVX)	4	**14**	56



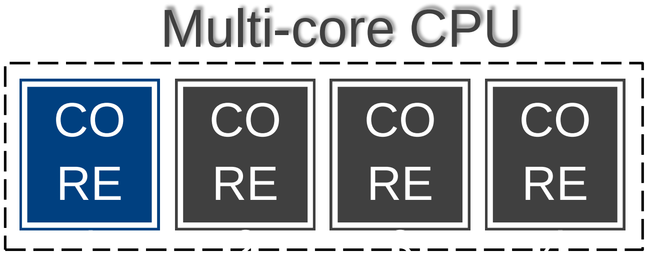
Several (Identical) Processing Cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**



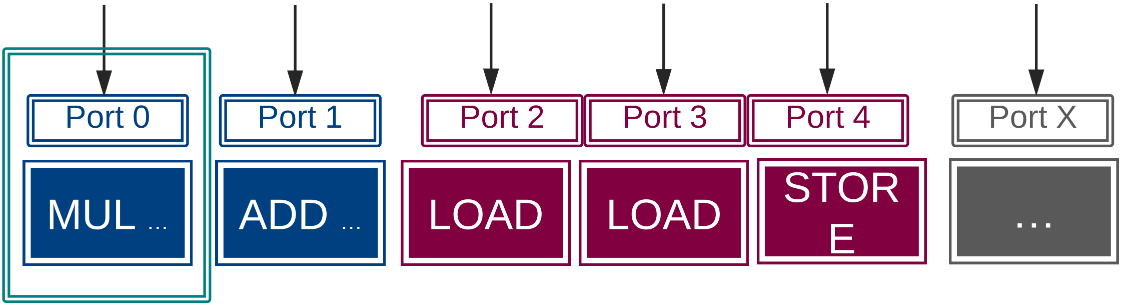
Independent instructions can run in parallel

Multi-core Architectures



Several (identical) processing cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**

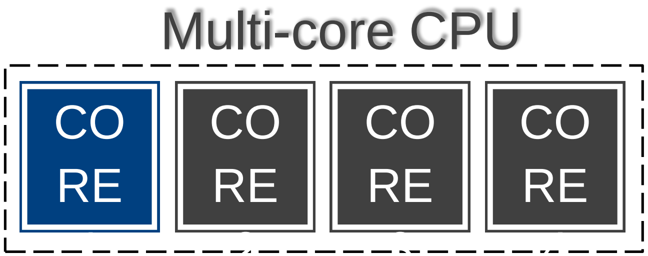


Hands-on: Ivy Bridge at 3.5 GHz (i7 3770K)

- Double-precision FLOPs
- **Throughput**: 1 instruction/clock

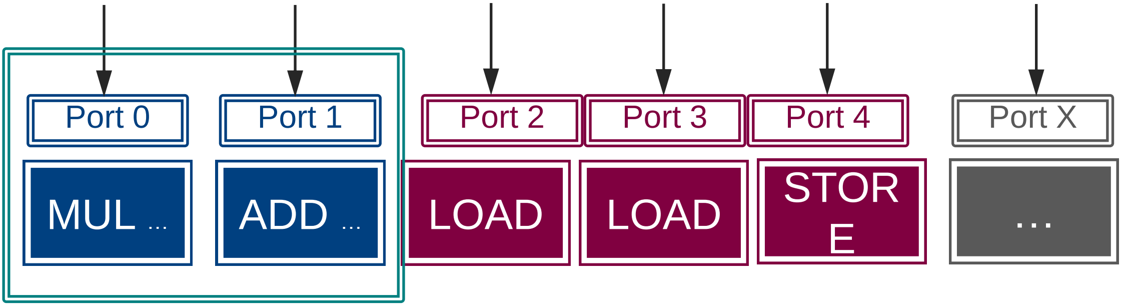
Instruction Type	FLOPS per Instr.	**Performance (GFLOPS/s)**	
		1 Core	4 Cores
64 bits	1	**3.5** (2flops x 3.5GHz)	**14**
128 bits (SSE)	2	**7**	**28**
256 bits (AVX)	4	**14**	56

Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**

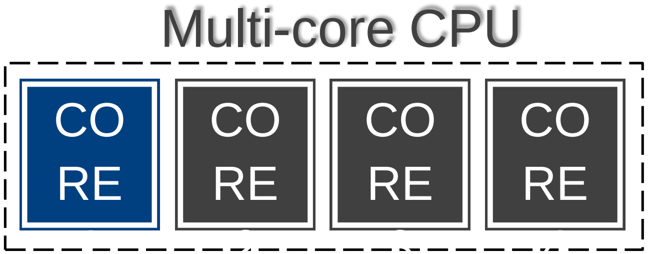


HANDS-ON: IVY BRIDGE AT 3.5 GHZ (I7 3770K)

- Double-precision FLOPs
- **Throughput**: **2 FP instructions/clock**

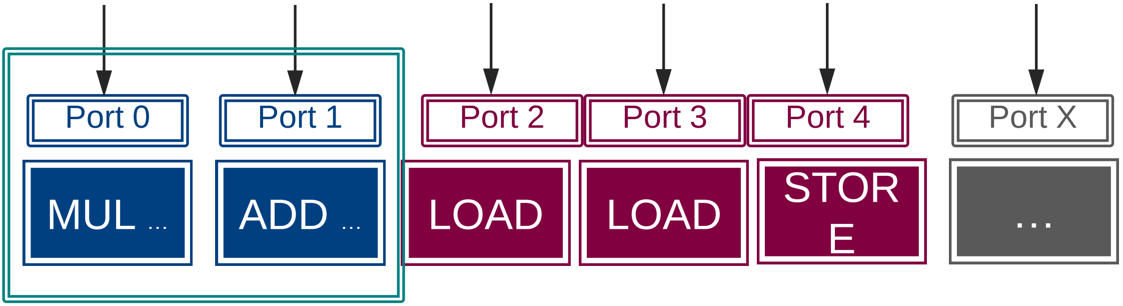
Instruction Type	FLOPS per Instr.	Performance (GFLOPS/s)	
		1 Core	4 Cores
64 bits	1	**3.5 → 7** (2flops x 3.5GHz)	**14→ 28**
128 bits (SSE)	2	**7→ 14**	**28→ 56**
256 bits (AVX)	4	**14→ 28**	56 → **112**

Multi-core Architectures



Several (Identical) Processing Cores

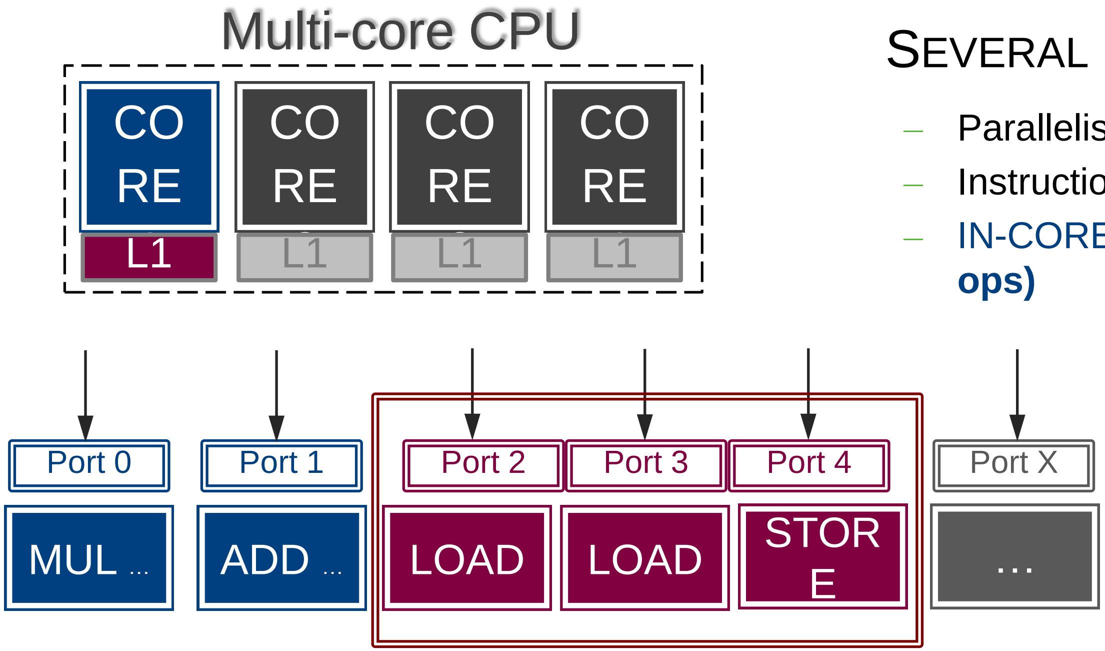
- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**



i7 3770K Ivy Bridge	Performance (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	?
4 Cores	112	?

*256-bit AVX double-precision floating-point instructions

Multi-core Architectures



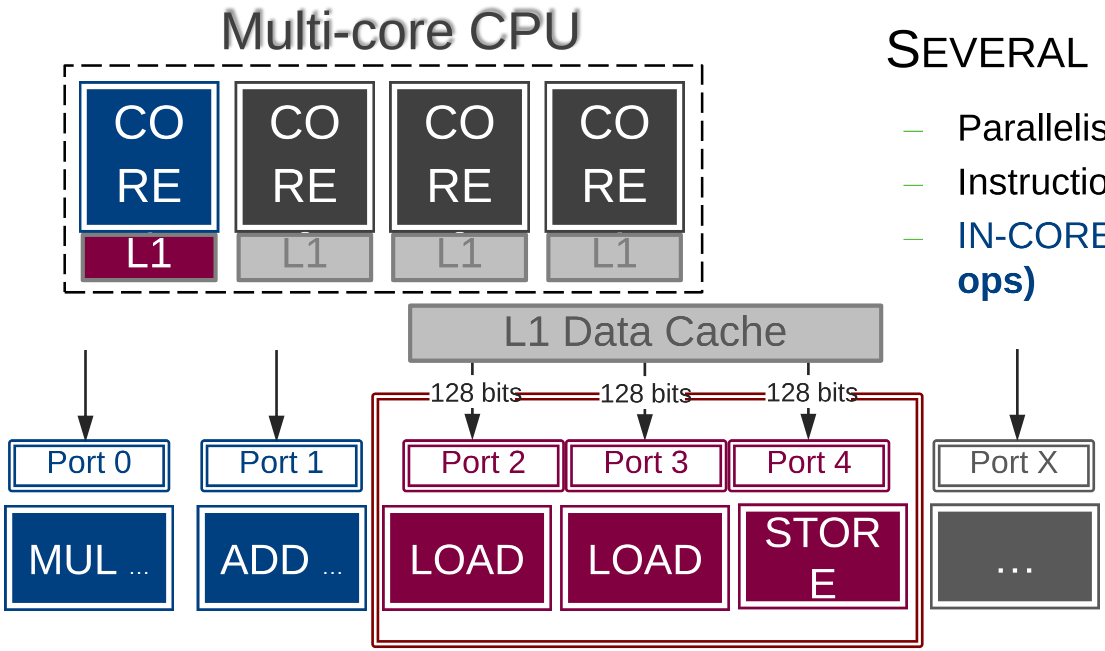
SEVERAL (IDENTICAL) PROCESSING CORES

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**

i7 3770K Ivy Bridge	Performance (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	?
4 Cores	112	?

*256-bit AVX double-precision floating-point instructions

Multi-core Architectures



Several (Identical) Processing Cores

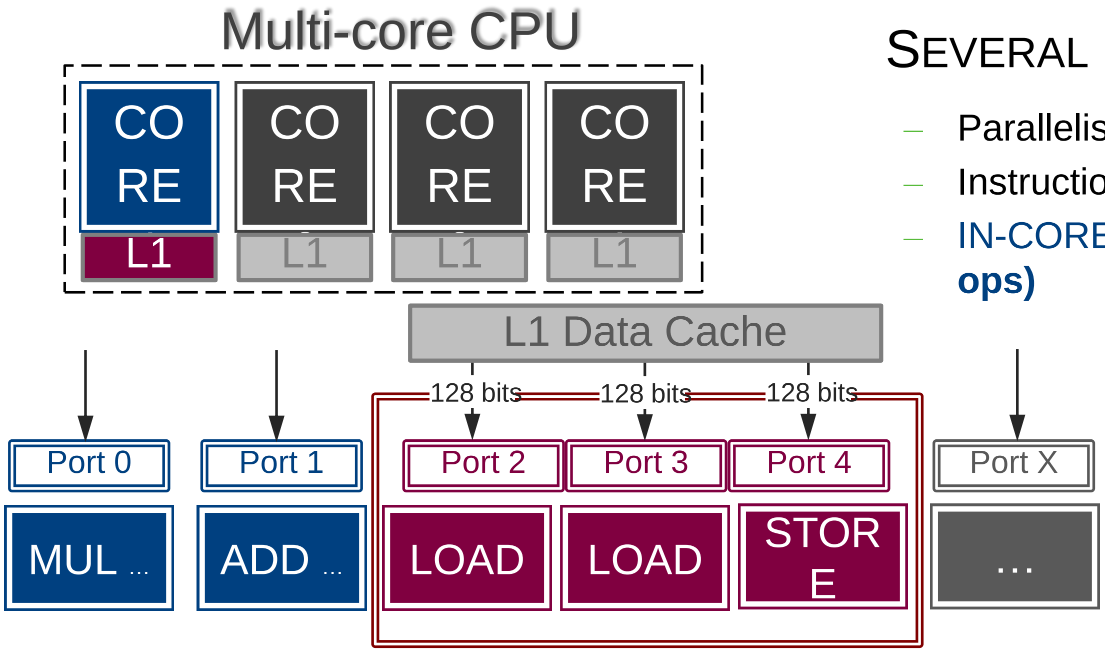
- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**

i7 3770K Ivy Bridge	Performanc e (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	?
4 Cores	112	?

*256-bit AVX double-precision floating-point instructions

What is the peak (theoretical) bandwidth of the L1→Core communication bus?

Multi-core Architectures



Several (identical) Processing Cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**

i7 3770K Ivy Bridge	Performance (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	168
4 Cores	112	?

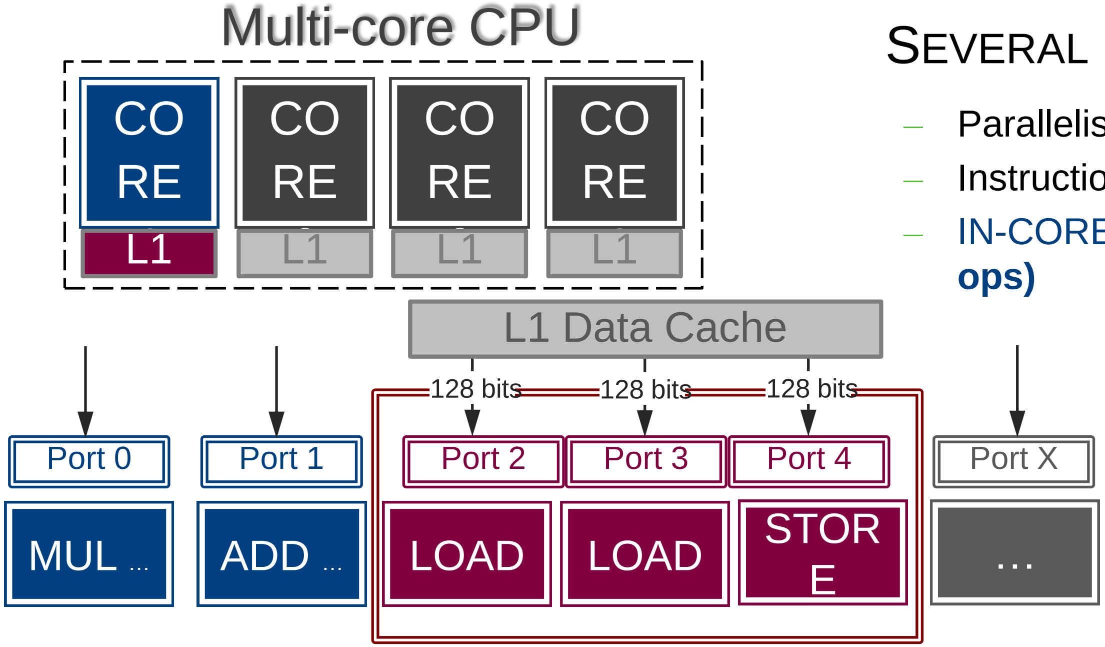
*256-bit AVX double-precision floating-point instructions

What is the peak (theoretical) bandwidth of the L1→Core communication bus?

- *3x128bits* = *48 bytes* can be transferred at the same time (per core)
- bus operates at the frequency of the processor → *3.5 GHz*

⇒ 48 bytes x 3.5 GHz = **168 GB/s (1 Core)**

Multi-core Architectures



Several (Identical) Processing Cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- IN-CORE PARALLELISM **(several ports for different ops)**

i7 3770K Ivy Bridge	Performance (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

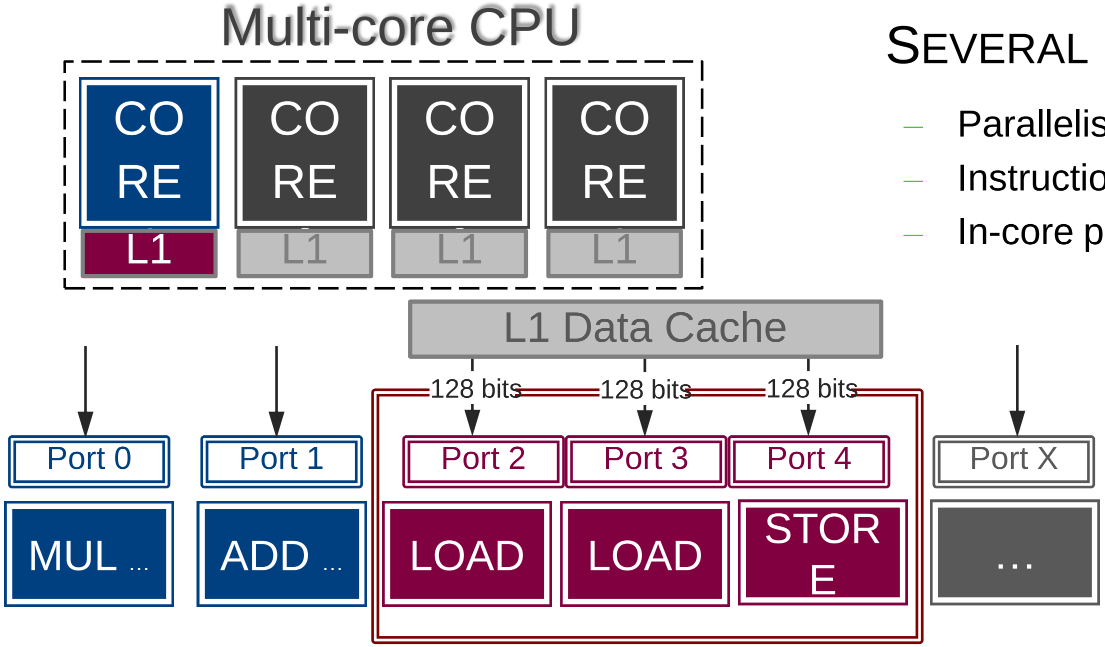
What is the peak (theoretical) bandwidth of the L1→Core communication bus?

- *3x128bits* = *48 bytes* can be transferred at the same time (per core)
- bus operates at the frequency of the processor → *3.5 GHz*

⇒ 48 bytes x 3.5 GHz = **168 GB/s (1 Core)**

⇒ 4 x 168 GB/s = **672 GB/s (4 cores)**

Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

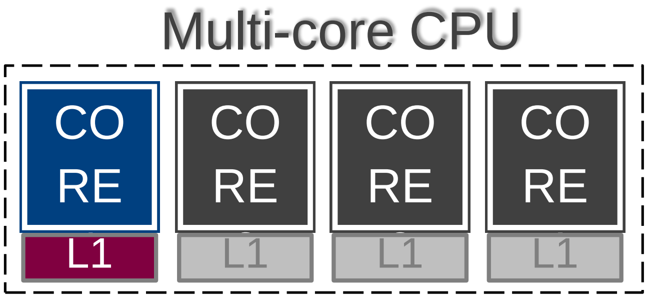
i7 3770K Ivy Bridge	Performance (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

We can't get higher than this!

What does it tell about the attainable performance?

Multi-core Architectures



Several (Identical) Processing Cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

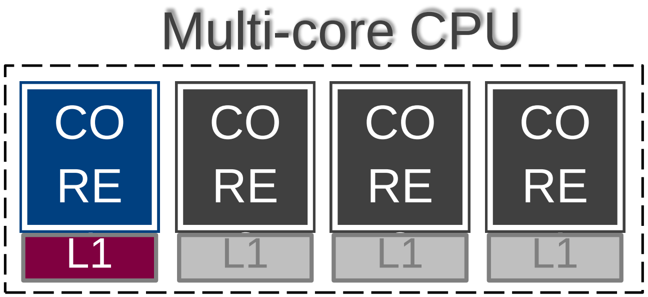
i7 3770K Ivy Bridge	Performance (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

We can't get higher than this!

What does it tell about the attainable performance?

Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

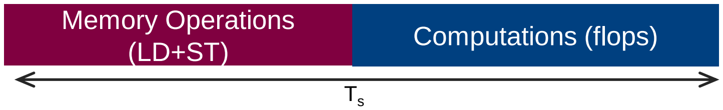
In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

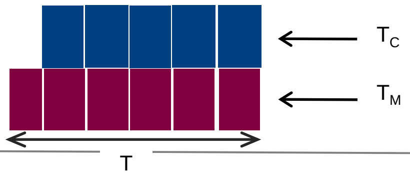
i7 3770K Ivy Bridge	Performance (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

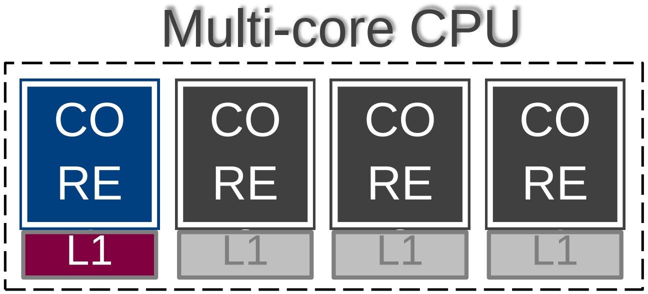
IF memory operations and computations are serially performed:



BUT they are actually executed in parallel (interleaved):



Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

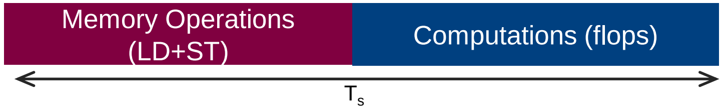
In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

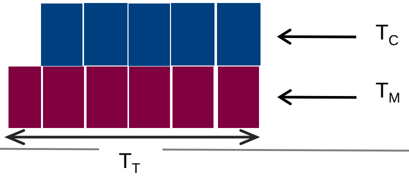
i7 3770K Ivy Bridge	Performanc e (GFlops/s)*	Bandwidth L1→C (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

IF memory operations and computations are serially performed:

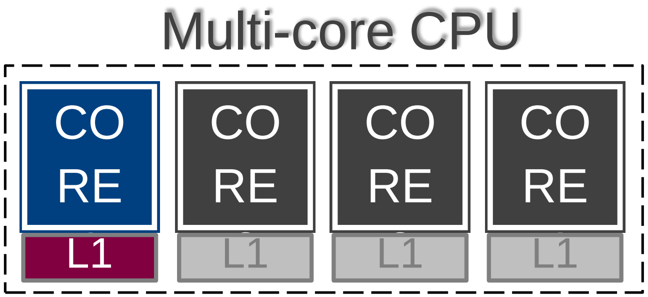


BUT they are actually executed in parallel (interleaved):



=> $T_T = \max\{T_C, T_M\}$

Multi-core Architectures



Several (Identical) Processing Cores

- – Parallelism across the processing cores
- – Instruction level parallelism (pipelining)
- – In-core parallelism (several ports for different ops)

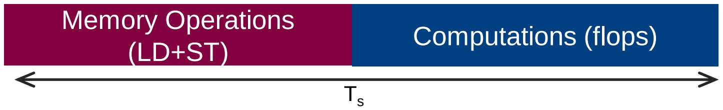
In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	168
4 Cores	**112**	672

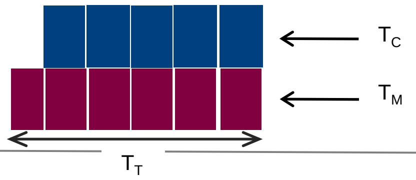
*256-bit AVX double-precision floating-point instructions

IF memory operations and computations are serially performed:



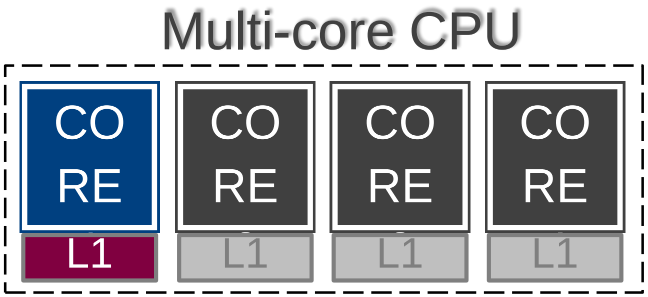
$T_C = \#flops/F_P$

BUT they are actually executed in parallel (interleaved):



=> $T_T=\max\{T_C,T_M\}=\max\{\#flops/F_P,T_M\}$

Multi-core Architectures



SEVERAL (IDENTICAL) PROCESSING CORES

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

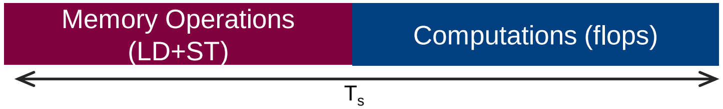
In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

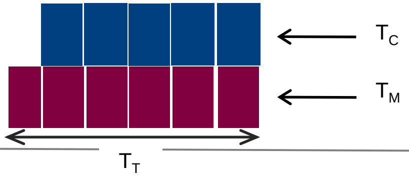
IF memory operations and computations are serially performed:



T_M = #bytes/B_P

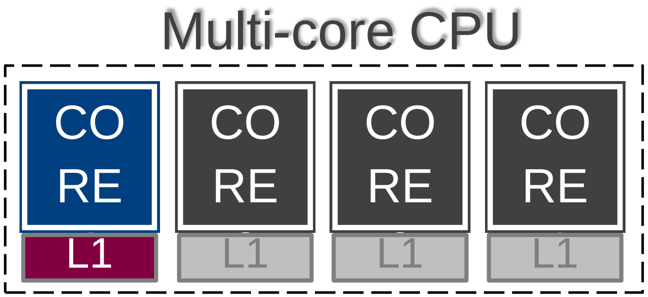
T_C = #flops/F_P

BUT they are actually executed in parallel (interleaved):



=> T_T=max{T_C,T_M}=max{#flops/F_P, #bytes/B_P}

Multi-core Architectures



Several (Identical) Processing Cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

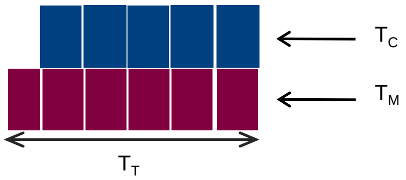
In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

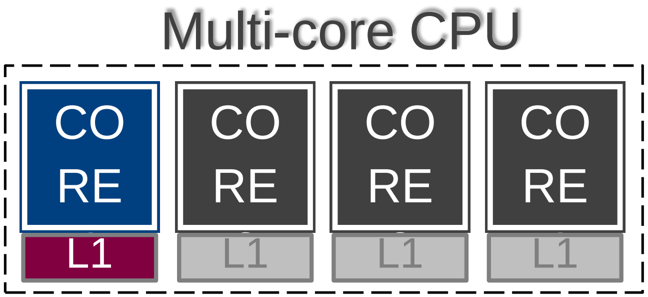
*256-bit AVX double-precision floating-point instructions

Memory operations and computations are executed in parallel (interleaved):



=> $T_T=\max\{T_C,T_M\}=\max\{\#flops/F_P, \#bytes/B_P\}$

Multi-core Architectures



Several (Identical) Processing Cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

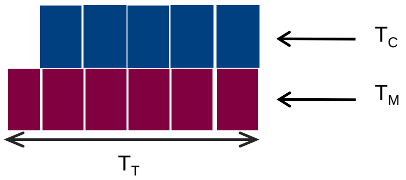
In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Memory operations and computations are executed in parallel (interleaved):

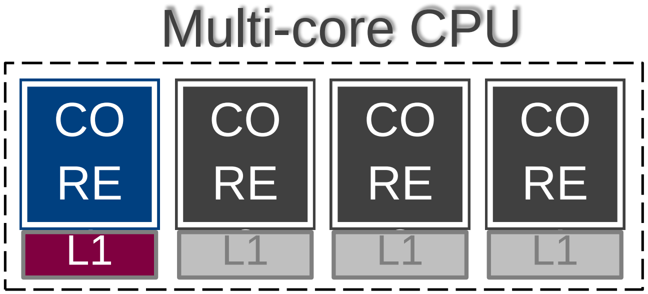


=> $T_T = \max\{T_C, T_M\} = \max\{\#flops/F_P, \#bytes/B_P\}$

Attainable Performance **(F_A) of the architecture:**

$$F_A = \#flops/T_T$$

Multi-core Architectures



Several (identical) Processing Cores

- Parallelism across the processing cores
- Instruction level parallelism (pipelining)
- In-core parallelism (several ports for different ops)

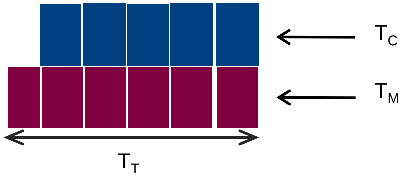
In general, real applications mix different number of flops and bytes:

```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Memory operations and computations are executed in parallel (interleaved):



=> $T_T = \max\{T_C, T_M\} = \max\{\#flops/F_P, \#bytes/B_P\}$

Attainable Performance **(F_A) of the architecture**

$$F_A = \#flops/T_T$$

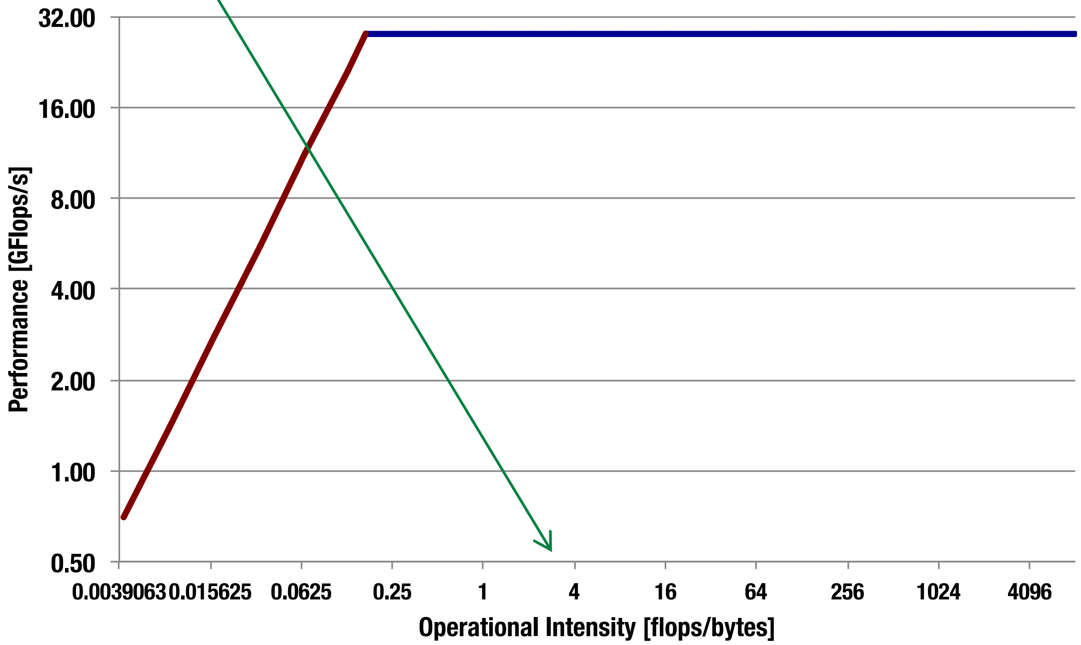
Roofline

Cache-aware Roofline Model

CACHE-AWARE ROOFLINE MODEL - insightful performance model of multi-core architectures relates:

1) Maximum Attainable Performance (**F_A=#flops/T_T**)

2) Operational Intensity (**I=#flops/#bytes**).

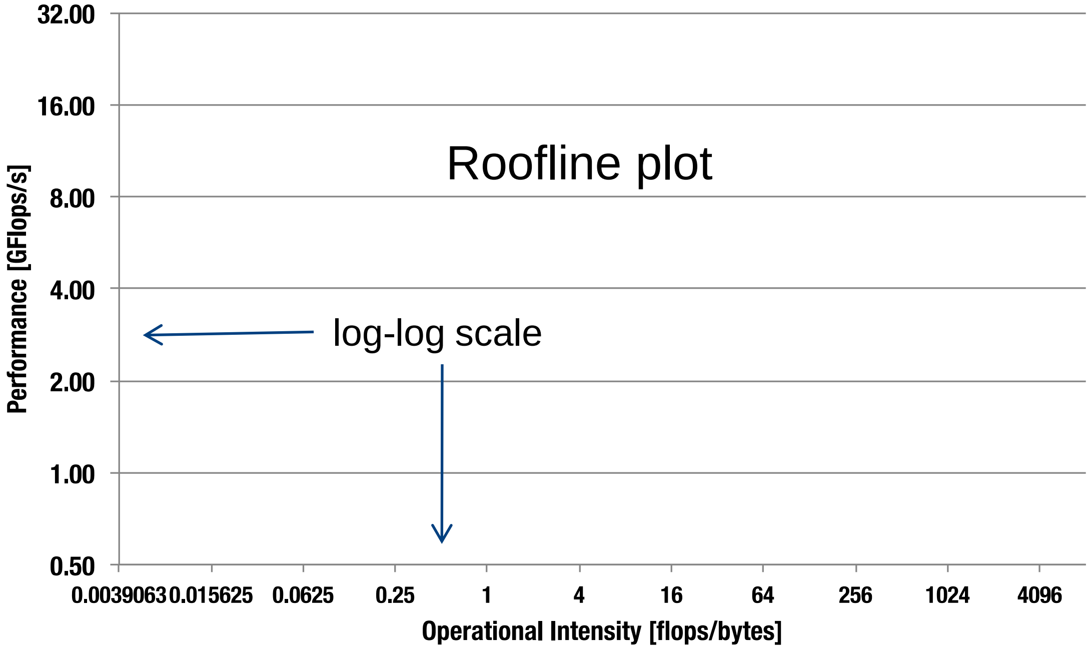


Building the Cache-aware Roofline Model

i7 3770K Ivy Bridge	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

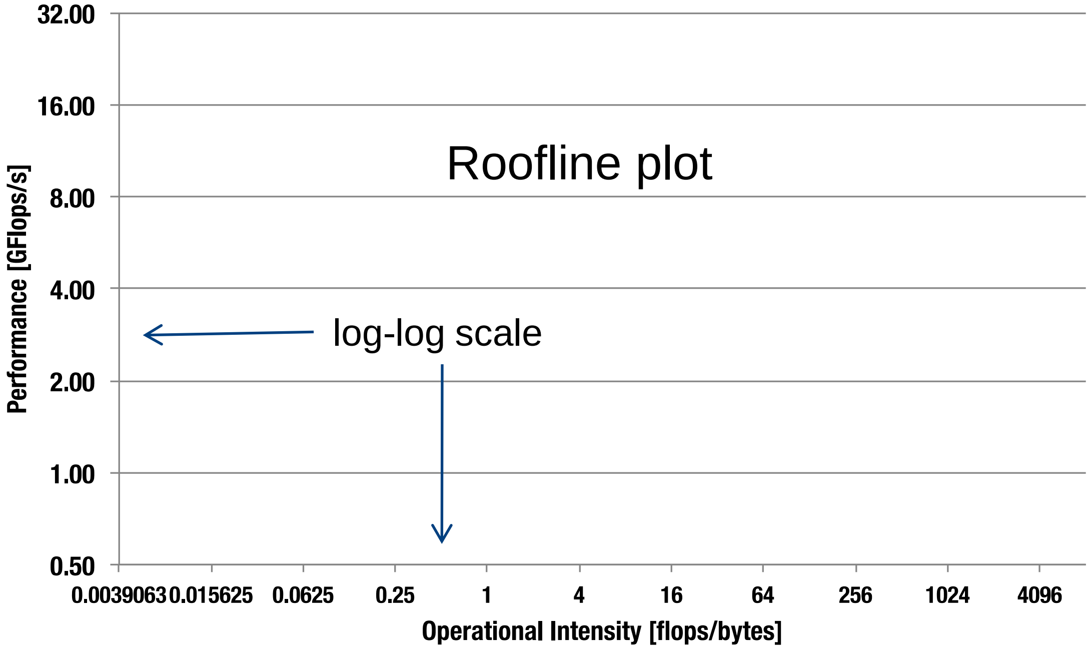
Building the Cache-aware Roofline Model

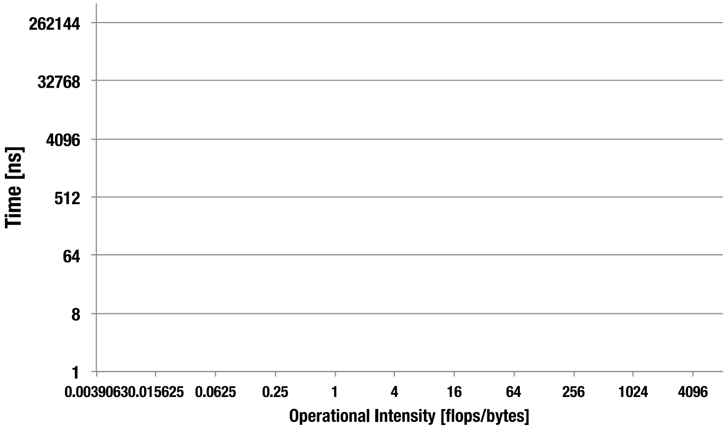


i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

Building the Cache-aware Roofline Model

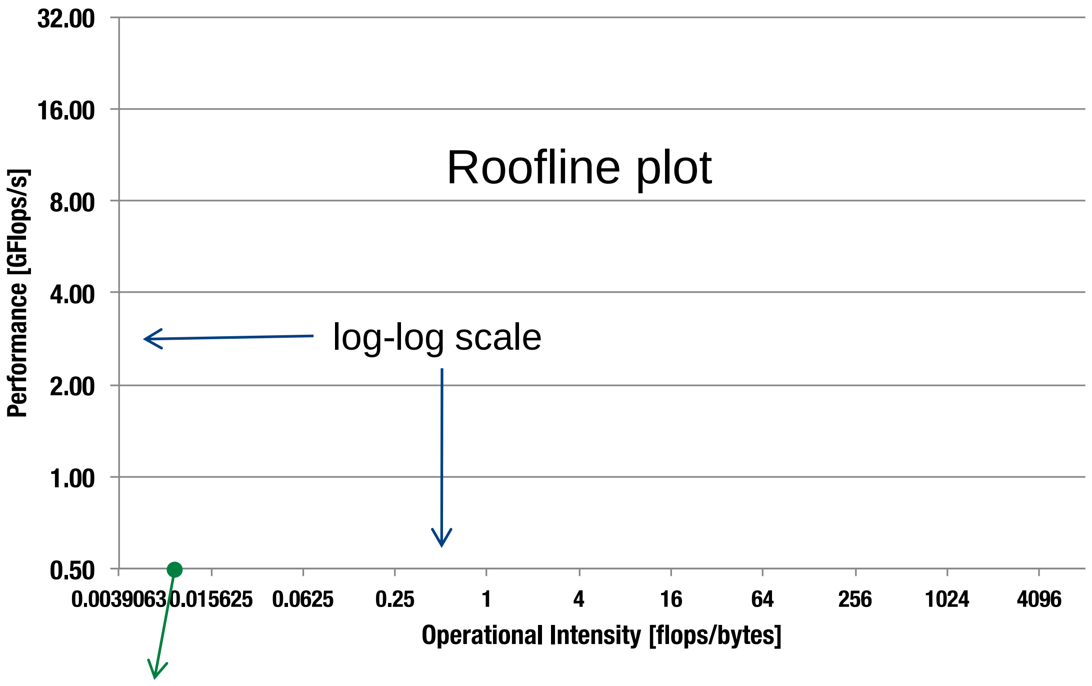


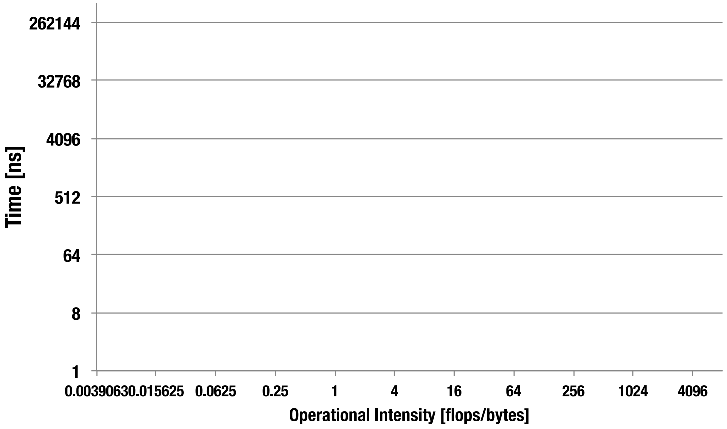


i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

Building the Cache-aware Roofline Model



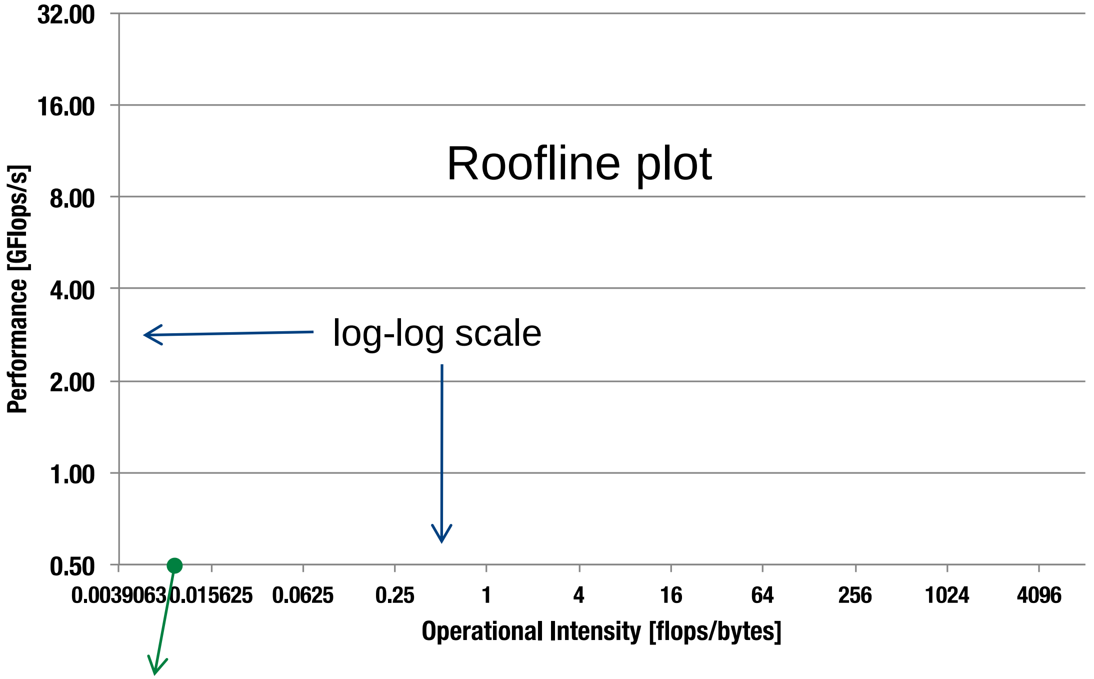


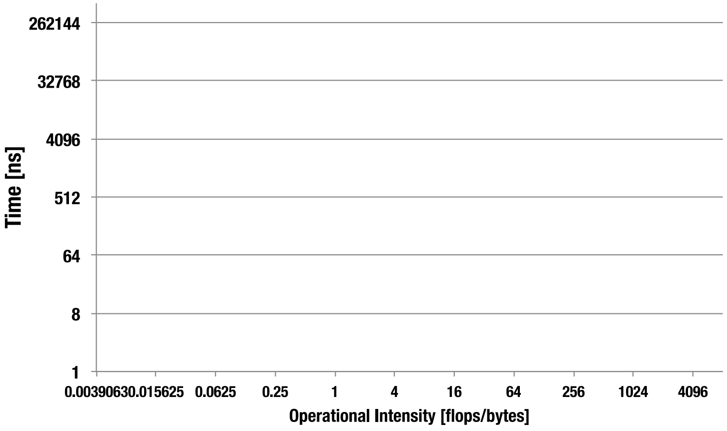
I=flops/bytes=0.0083 = 8flops/960bytes [1MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Building the Cache-aware Roofline Model





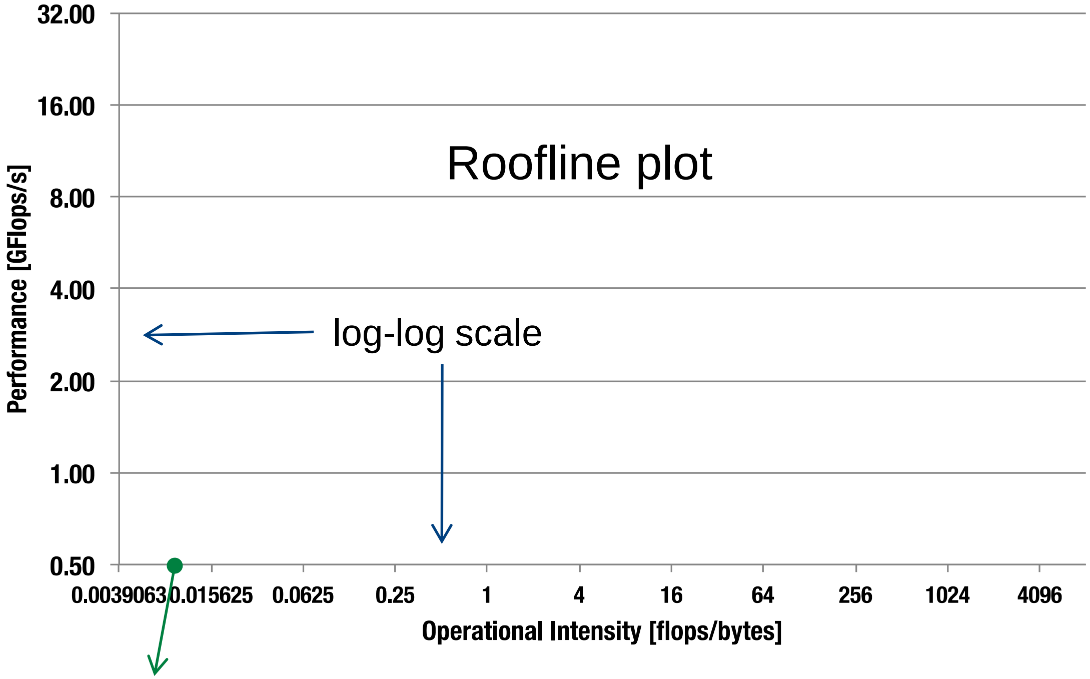
I=flops/bytes=0.0083 = 8flops/960bytes [1MAD/(**10x**(2LD+ST))*]

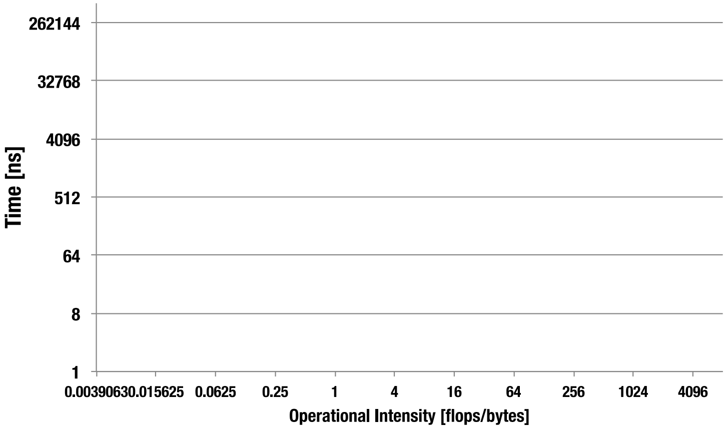
i7 3770K **Ivy** **Bridge**	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

T_C = #flops/F_P = 8flops/28 = 0.29 ns

Building the Cache-aware Roofline Model





I=flops/bytes=0.0083 = 8flops/960bytes [1MAD/(**10x**(2LD+ST))*]

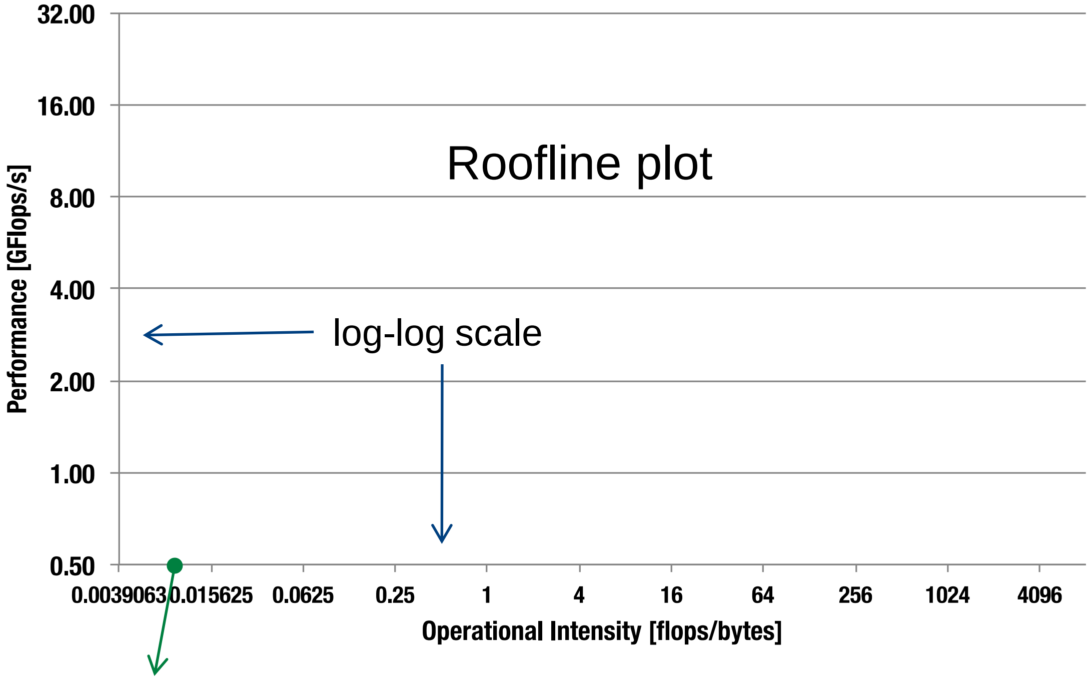
i7 3770K Ivy Bridge	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

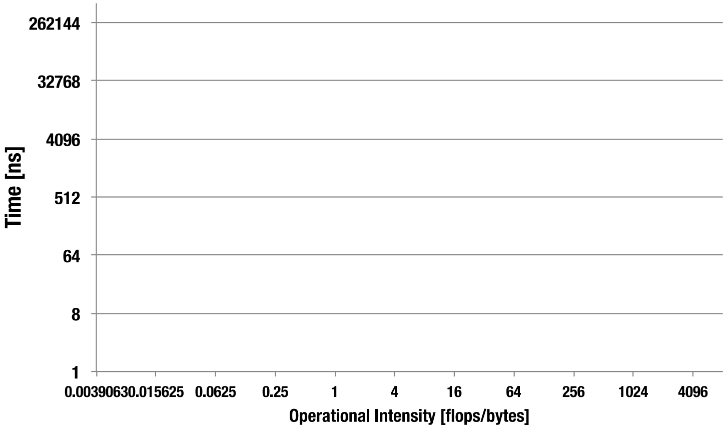
*256-bit AVX double-precision floating-point instructions

T_C **= #flops/F_P** = 8flops/28 = 0.29 ns

T_M **= #bytes/B_P** = 960bytes/168 = 5.71 ns

Building the Cache-aware Roofline Model





I=flops/bytes=0.0083 = 8flops/960bytes [1MAD/(**10x**(2LD+ST))*]

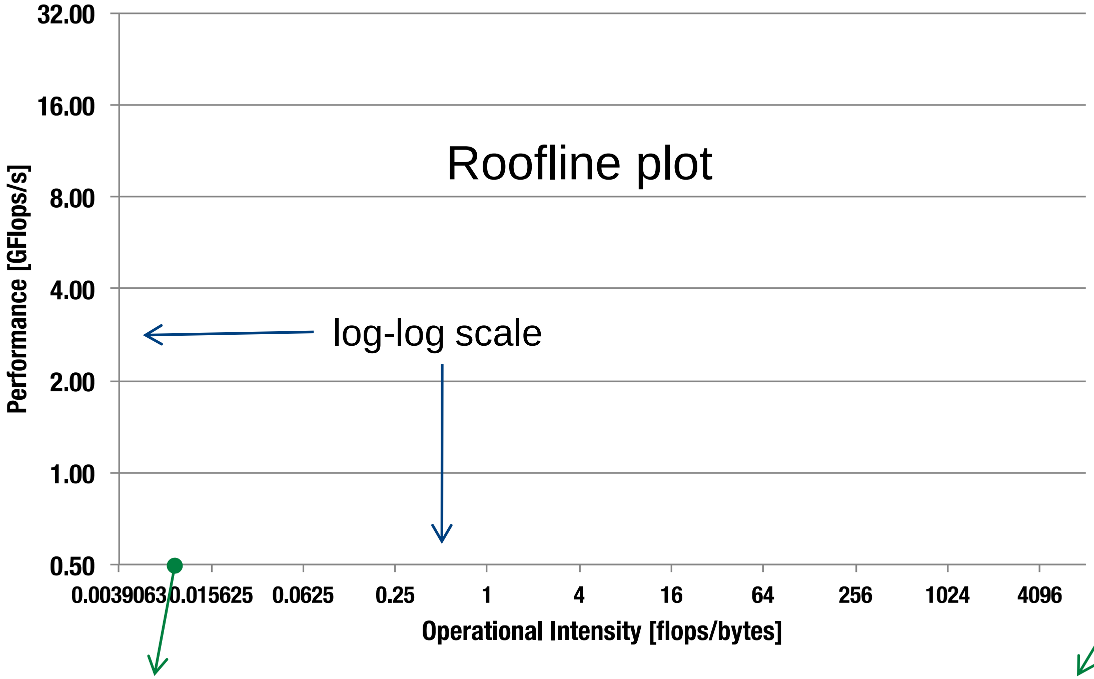
i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672
*256-bit AVX double-precision floating-point instructions		

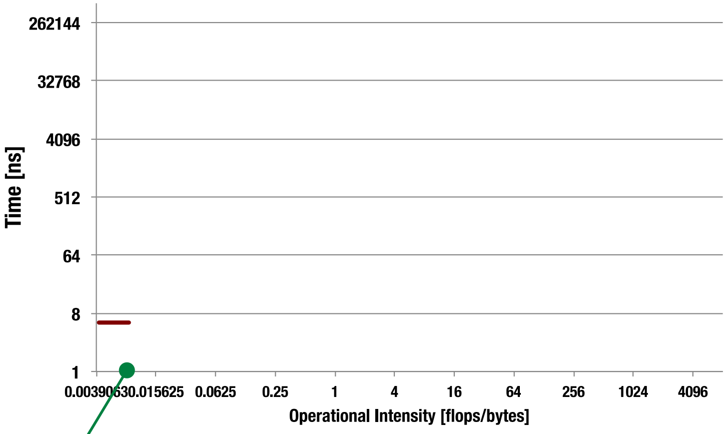
$T_C = \#flops/F_P$ = 8flops/28 = 0.29 ns

$T_M = \#bytes/B_P$ = 960bytes/168 = 5.71 ns

$T_T = \max\{T_C, T_M\}$ = 5.71 ns

Building the Cache-aware Roofline Model





I=flops/bytes=0.0083 = 8flops/960bytes [1MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

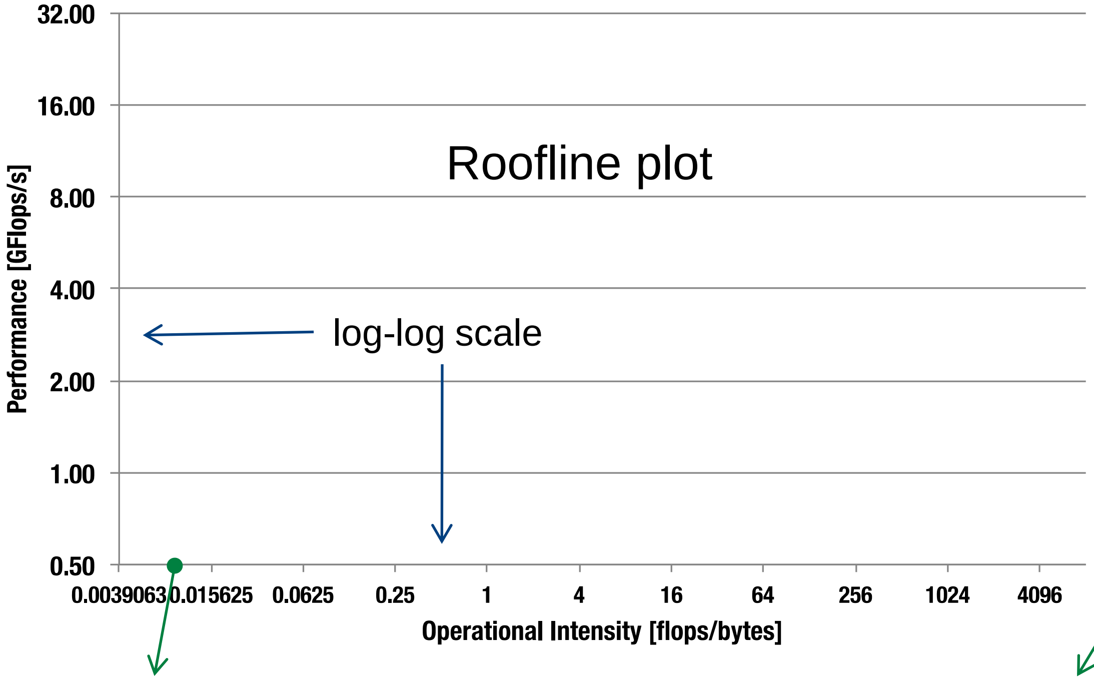
*256-bit AVX double-precision floating-point instructions

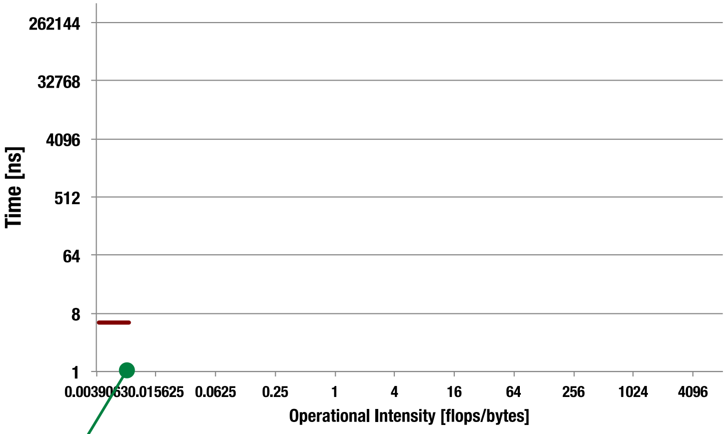
$T_C = \#flops/F_P$ = 8flops/28 = 0.29 ns

$T_M = \#bytes/B_P$ = 960bytes/168 = 5.71 ns

$T_T = \max\{T_C, T_M\}$ = 5.71 ns

Building the Cache-aware Roofline Model





I=flops/bytes=0.0083 = 8flops/960bytes [1MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

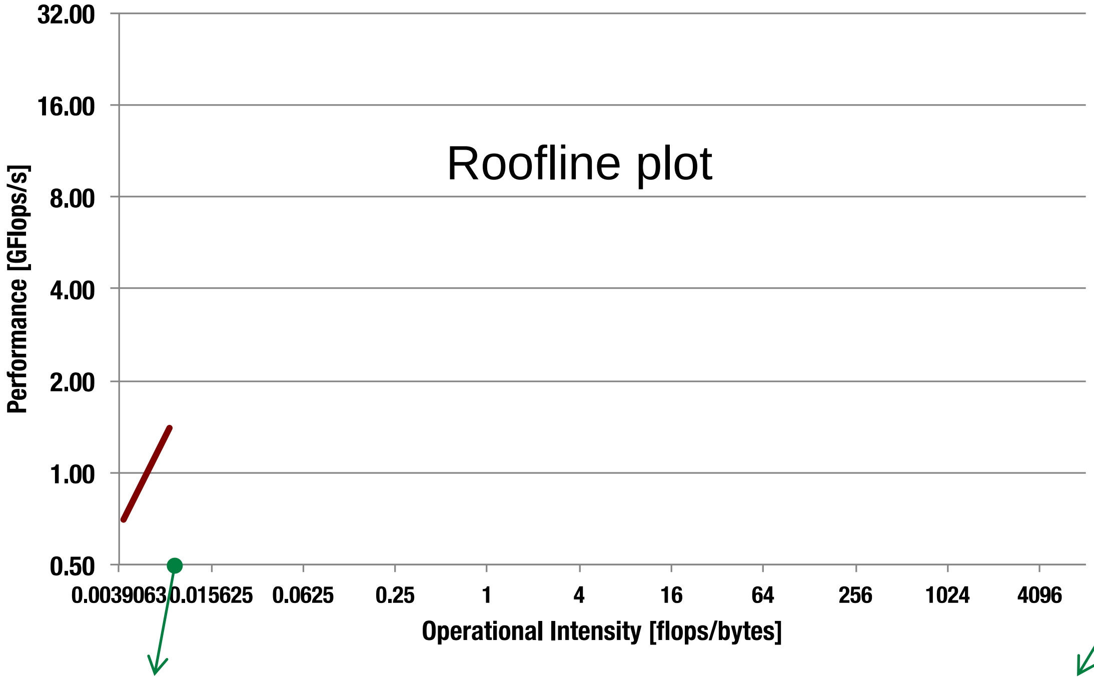
$\mathbf{T_C = \#flops/F_P}$ = 8flops/28 = 0.29 ns

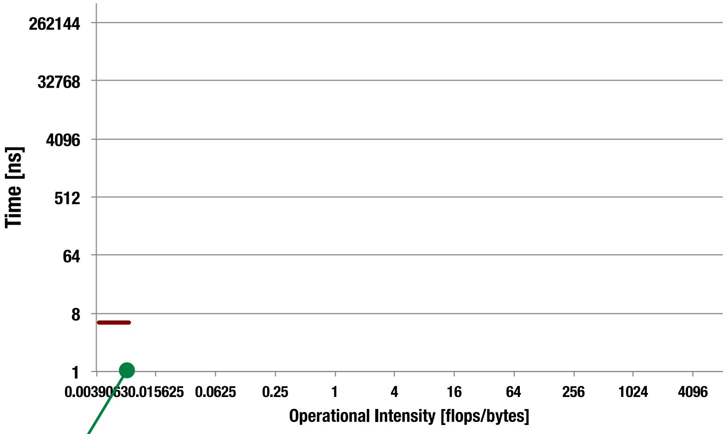
$\mathbf{T_M = \#bytes/B_P}$ = 960bytes/168 = 5.71 ns

$\mathbf{T_T = max\{T_C,T_M\}}$ **=** 5.71 ns

$\mathbf{F_A = \#flops/T_T}$ **=** 8flops/5.71 ns **= 1.4 Gflops/s**

Building the Cache-aware Roofline Model





I=**flops**/**bytes**=0.0083 = 8flops/960bytes [1MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

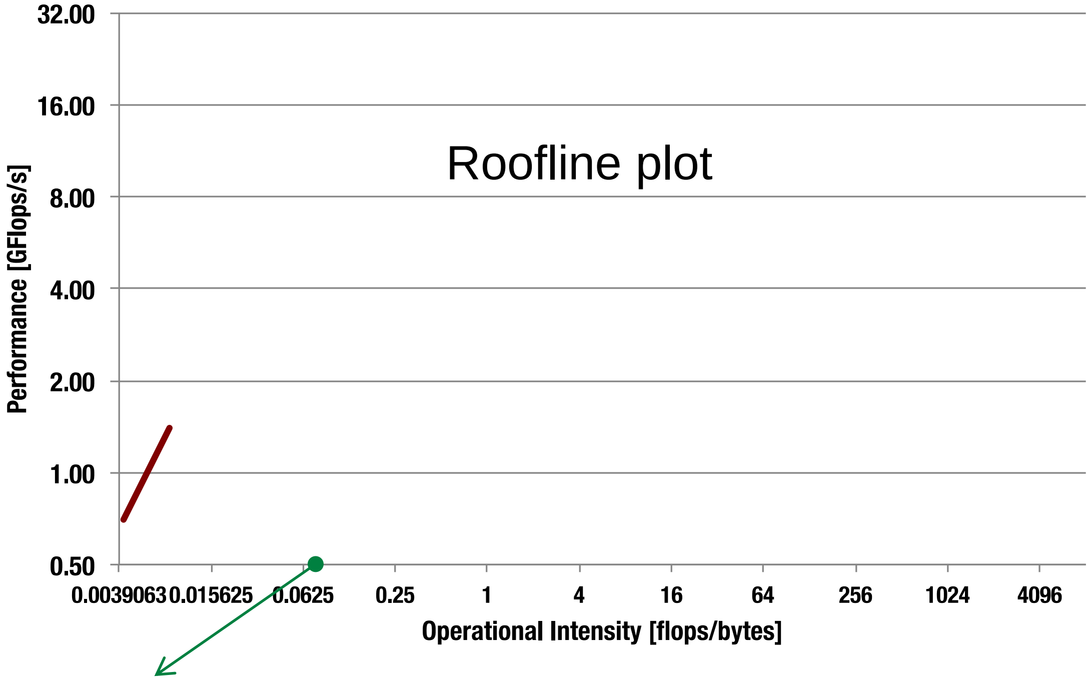
$\mathbf{T_C = \#flops/F_P}$ = 8flops/28 = 0.29 ns

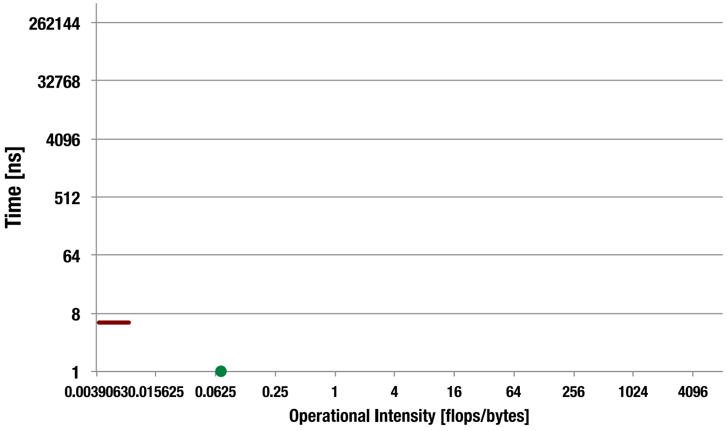
$\mathbf{T_M = \#bytes/B_P}$ = 960bytes/168 = 5.71 ns

$\mathbf{T_T = max\{T_C,T_M\}}$ = 5.71 ns

$\mathbf{F_A = \#flops/T_T}$ = 8flops/5.71 ns **= 1.4 Gflops/s**

Building the Cache-aware Roofline Model





I=flops/bytes=0.067 = 64flops/960bytes [8MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

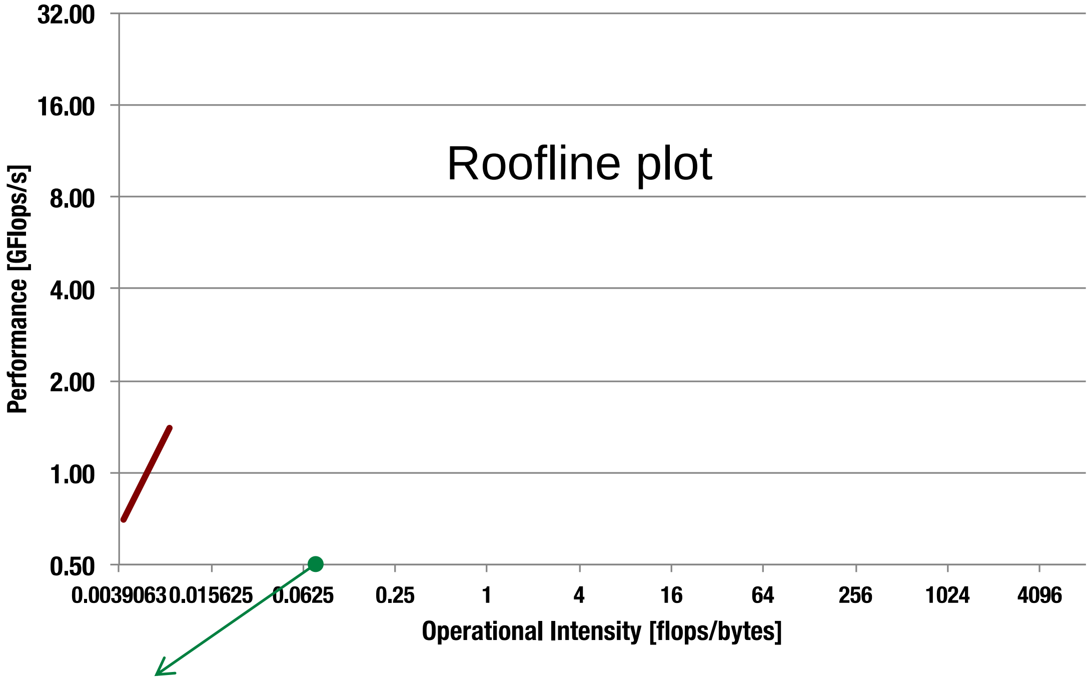
$T_C = \#\text{flops}/F_P = ?$

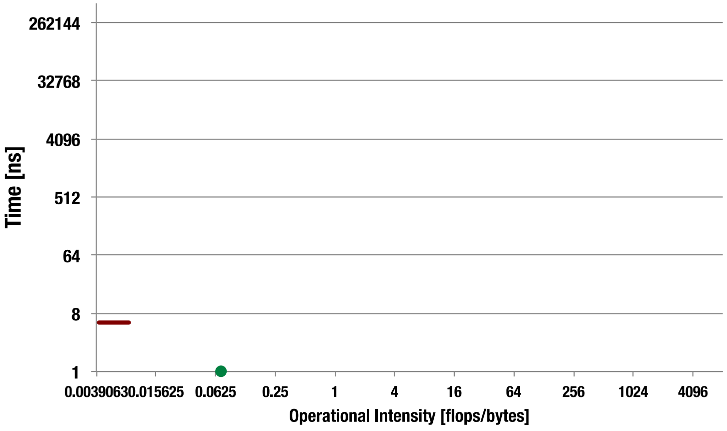
$T_M = \#\text{bytes}/B_P = ?$

$T_T = \max\{T_C, T_M\} = ?$

$F_A = \#\text{flops}/T_T = ?$

Building the Cache-aware Roofline Model





I=flops/bytes=0.067 = 64flops/960bytes [8MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

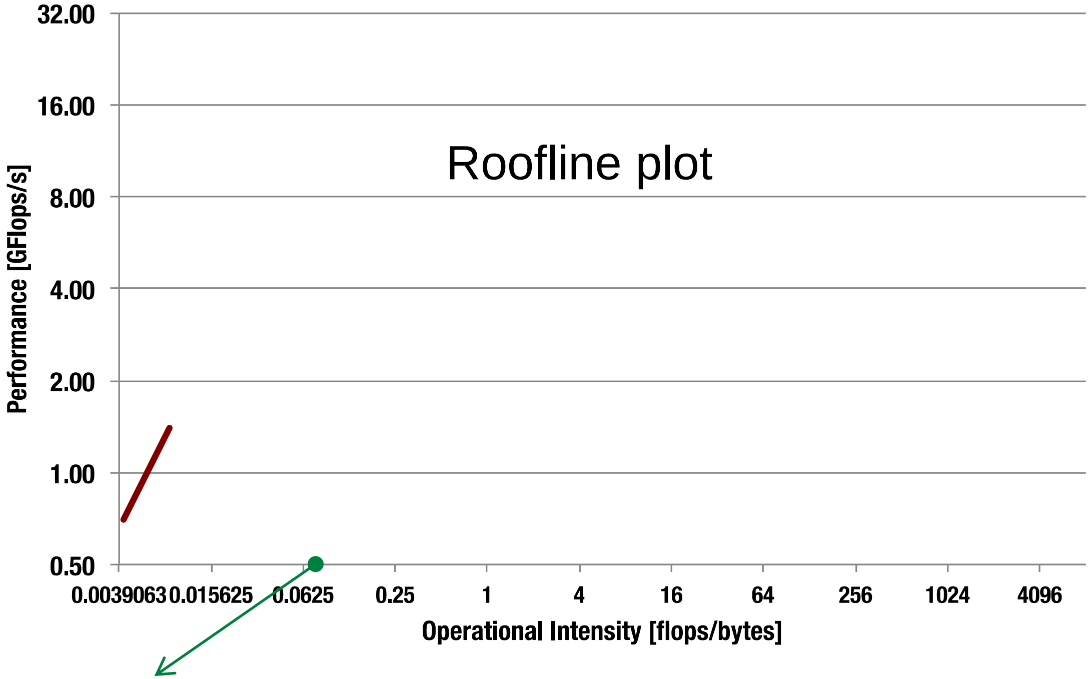
T_C = **#flops/F_P** = 64flops/28 = 2.29 ns

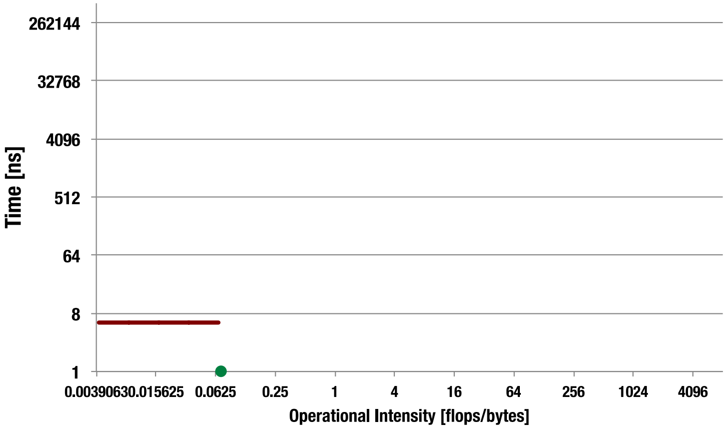
T_M = **#bytes/B_P** = 960bytes/168 = 5.71 ns

T_T = **max{T_C,T_M}** = 5.71 ns

F_A = #flops/T_T = ?

Building the Cache-aware Roofline Model





I=flops/bytes=0.067 = 64flops/960bytes [8MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

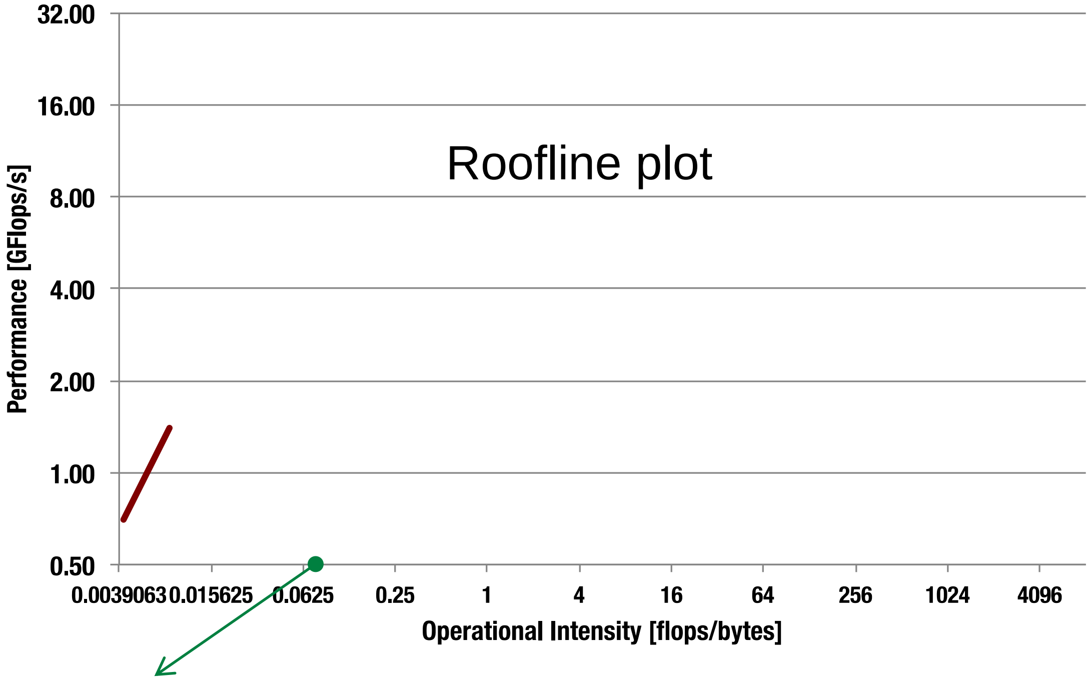
T_C = #flops/F_P = 64flops/28 = 2.29 ns

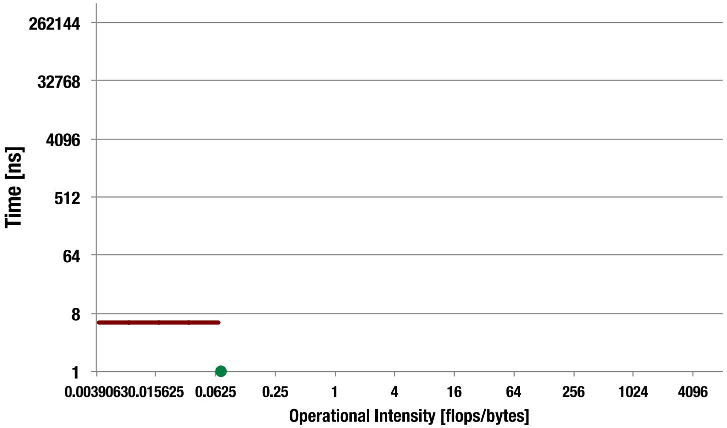
T_M = #bytes/B_P = 960bytes/168 = 5.71 ns

T_T = max{T_C,T_M} = 5.71 ns

F_A = #flops/T_T = ?

Building the Cache-aware Roofline Model





I=flops/bytes=0.067 = 64flops/960bytes [8MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

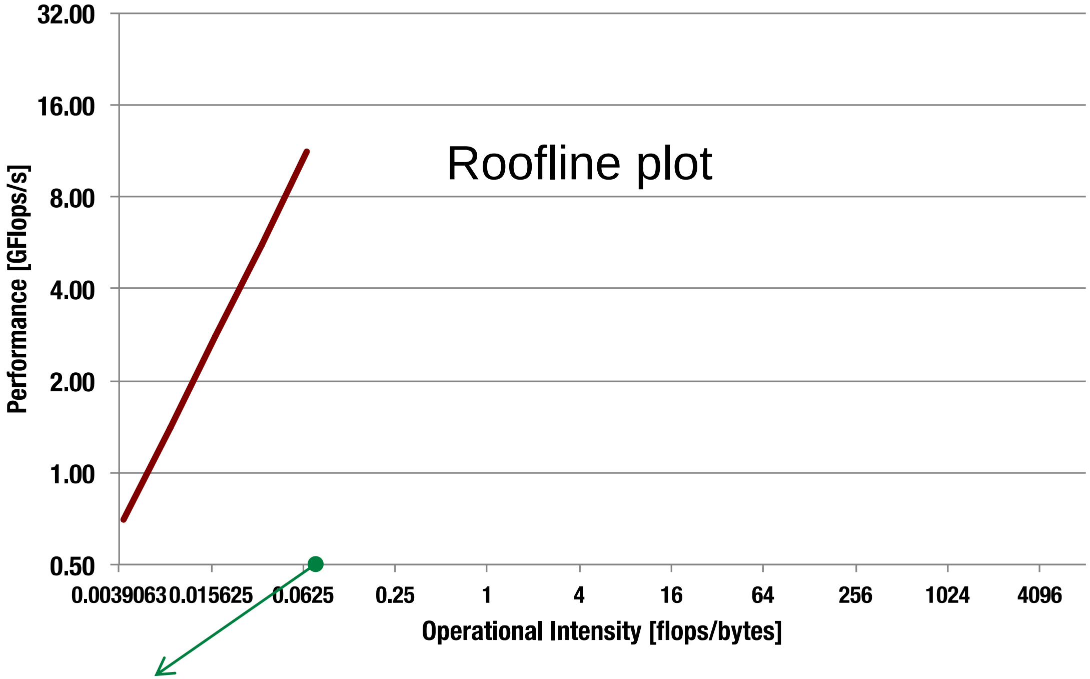
T_C = #flops/F_P = 64flops/28 = 2.29 ns

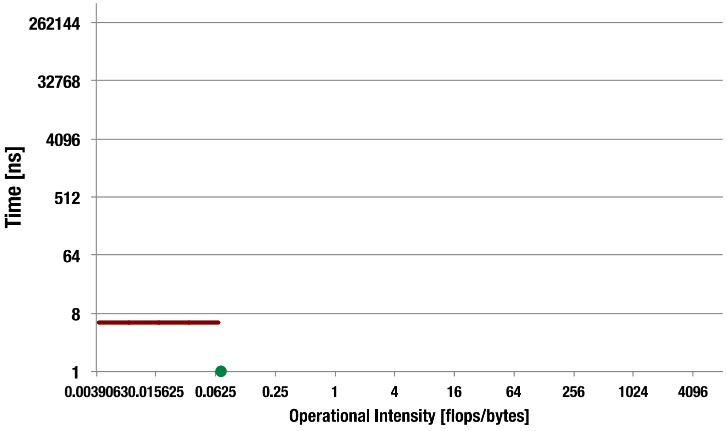
T_M = #bytes/B_P = 960bytes/168 = 5.71 ns

T_T = max{T_C,T_M} = 5.71 ns

F_A = #flops/T_T = 64flops/5.71 ns = **11.2 Gflops/s**

Building the Cache-aware Roofline Model





I=**flops**/**bytes**=0.067 = 64flops/960bytes [8MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

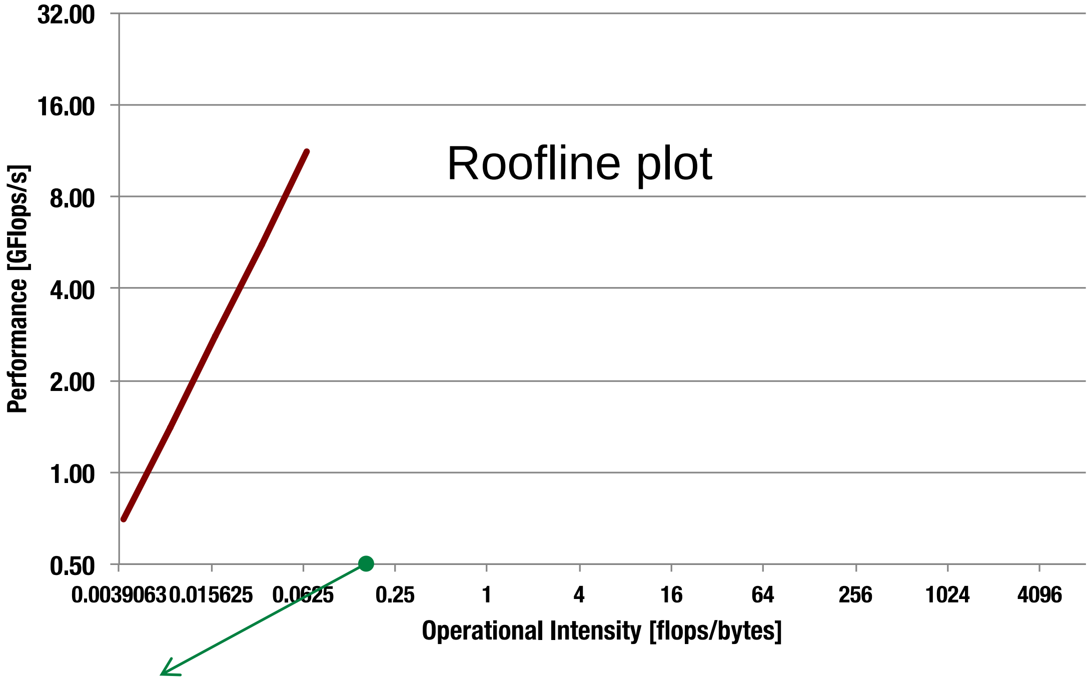
T_C = #flops/F_P = 64flops/28 = 2.29 ns

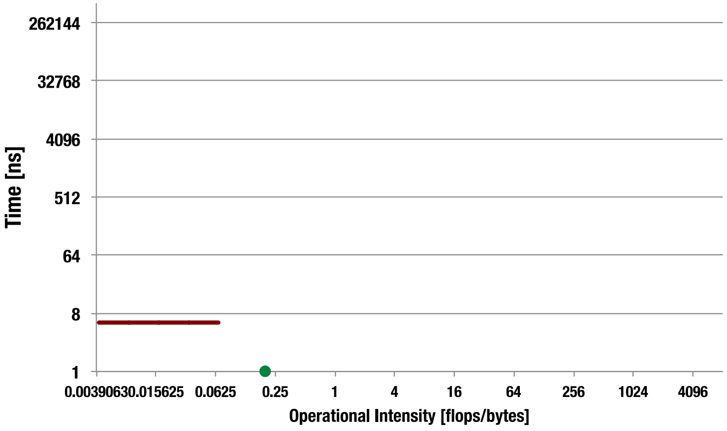
T_M = #bytes/B_P = 960bytes/168 = 5.71 ns

$T_T = \max\{T_C, T_M\}$ = 5.71 ns

F_A = #flops/T_T = 64flops/5.71 ns = **11.2 Gflops/s**

Building the Cache-aware Roofline Model





I=flops/bytes=0.016 = 160flops/960bytes [20MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

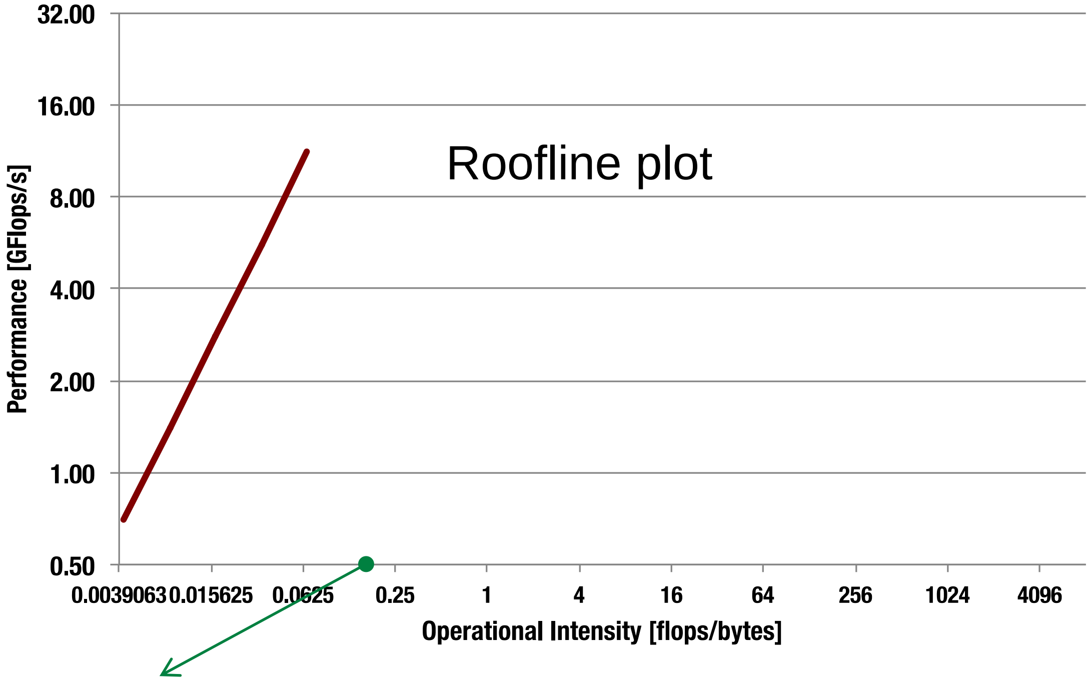
T_C **= #flops/F_P** = ?

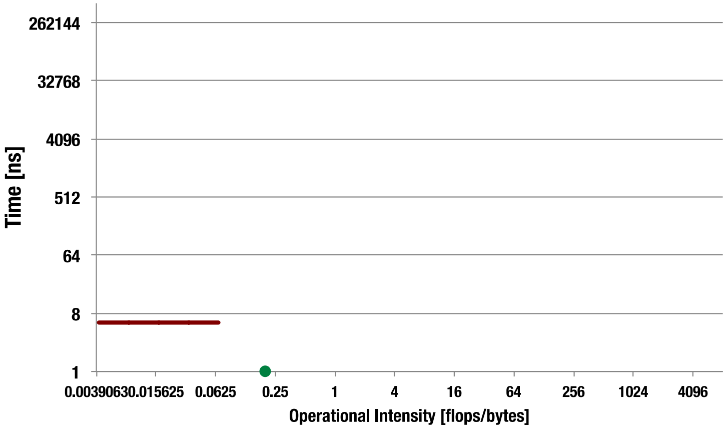
T_M **= #bytes/B_P** = ?

T_T = max{T_C,T_M} = ?

F_A = #flops/T_T = ?

Building the Cache-aware Roofline Model





I=flops/bytes=0.016 = 160flops/960bytes [20MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

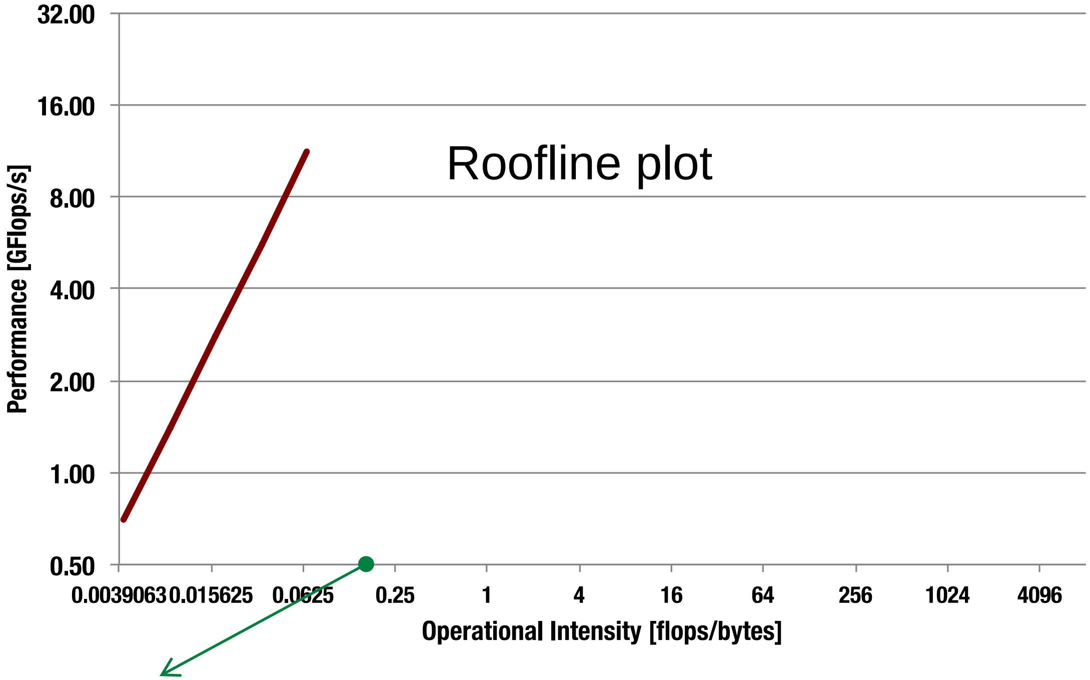
$T_C = \#flops/F_P$ = 160flops/28 = 5.71 ns

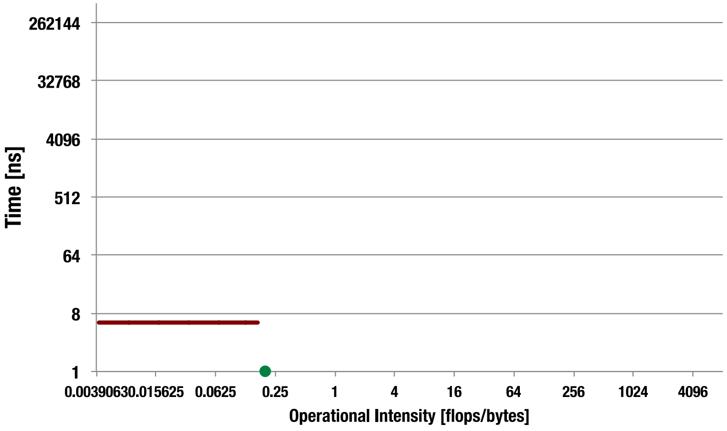
$T_M = \#bytes/B_P$ = 960bytes/168 = 5.71 ns

$T_T = \max\{T_C, T_M\}$ = 5.71 ns $(T_C = T_M)$

$F_A = \#flops/T_T = ?$

Building the Cache-aware Roofline Model





I=flops/bytes=0.016 = 160flops/960bytes [20MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

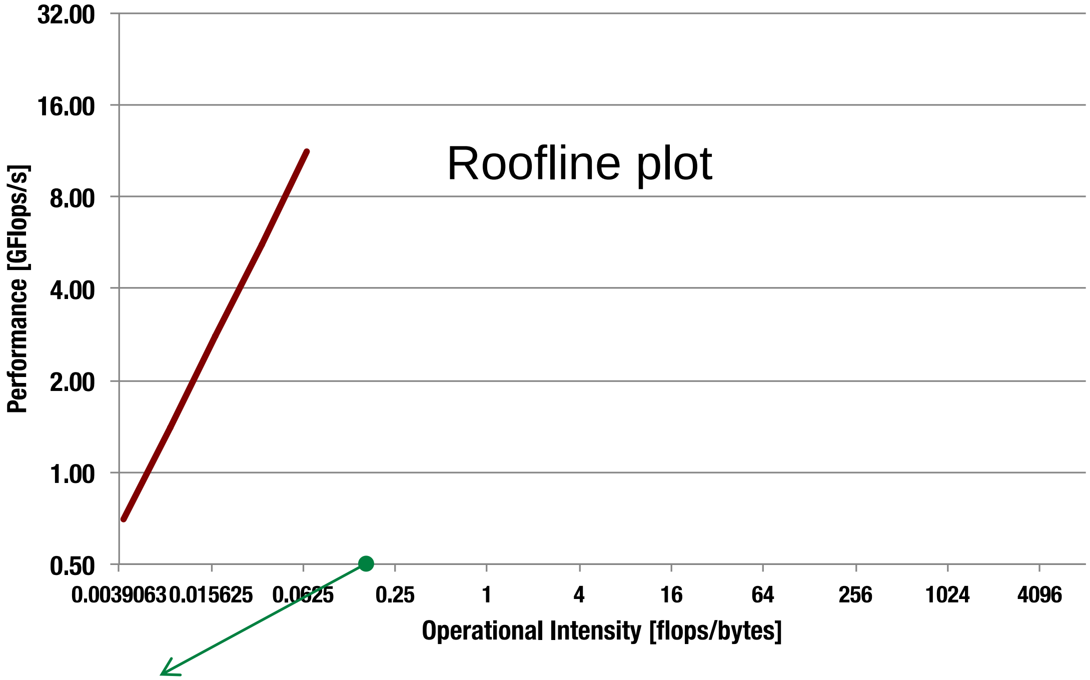
T_C = **#flops/F_P** = 160flops/28 = 5.71 ns

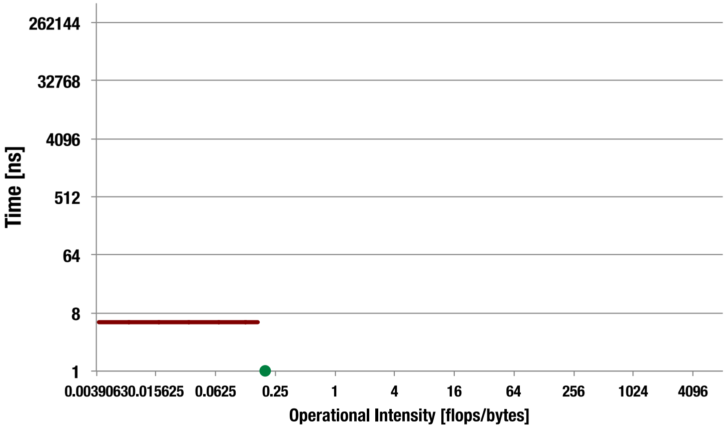
T_M = **#bytes/B_P** = 960bytes/168 = 5.71 ns

T_T = **max{T_C,T_M}** = 5.71 ns **(T_C=T_M)**

F_A = #flops/T_T = ?

Building the Cache-aware Roofline Model





I=flops/bytes=0.016 = 160flops/960bytes [20MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**
*256-bit AVX double-precision floating-point instructions		

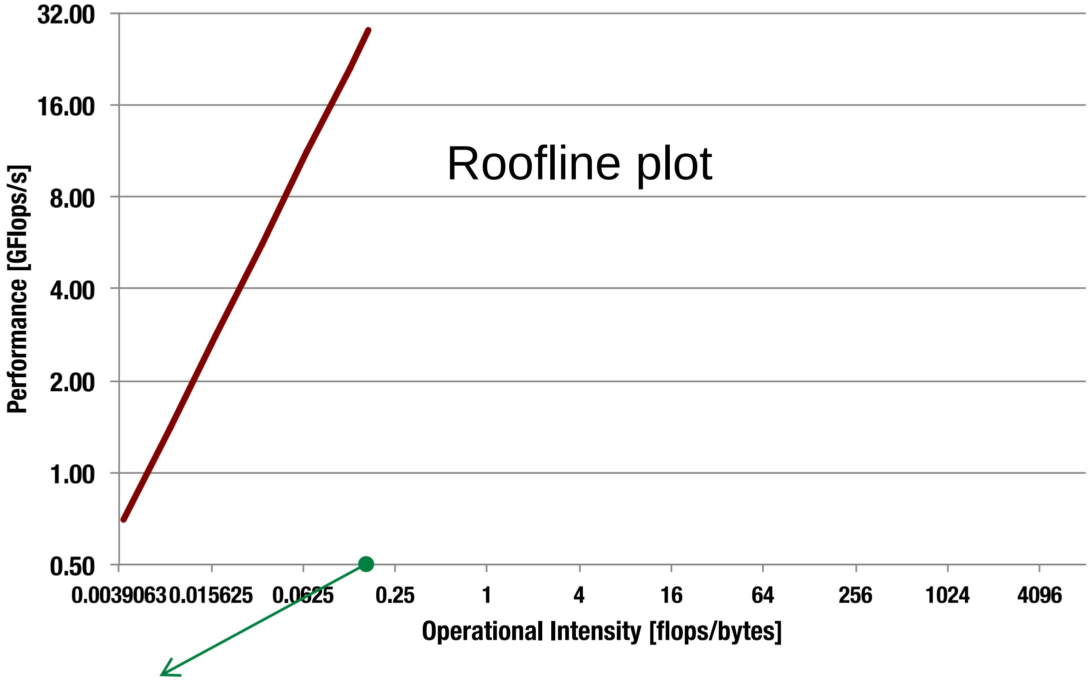
$\mathbf{T_C = \#flops/F_P}$ = 160flops/28 = 5.71 ns

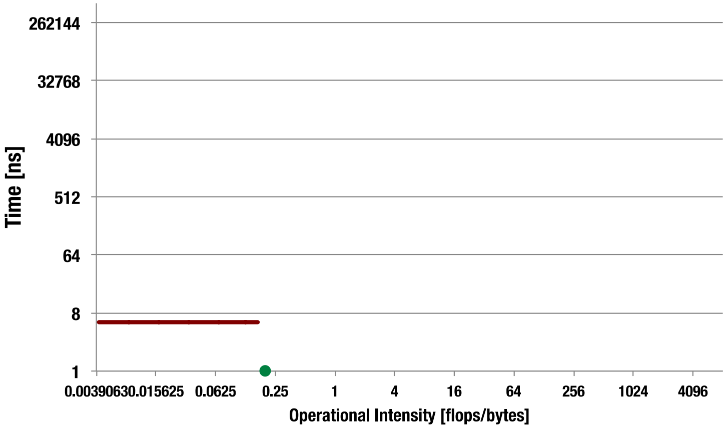
$\mathbf{T_M = \#bytes/B_P}$ = 960bytes/168 = 5.71 ns

$\mathbf{T_T = max\{T_C,T_M\}}$ **=** 5.71 ns **($\mathbf{T_C=T_M}$)**

$\mathbf{F_A = \#flops/T_T}$ **=** 160flops/5.71 ns **= 28 Gflops/s**

Building the Cache-aware Roofline Model





I=**flops**/**bytes**=0.016 = 160flops/960bytes [20MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

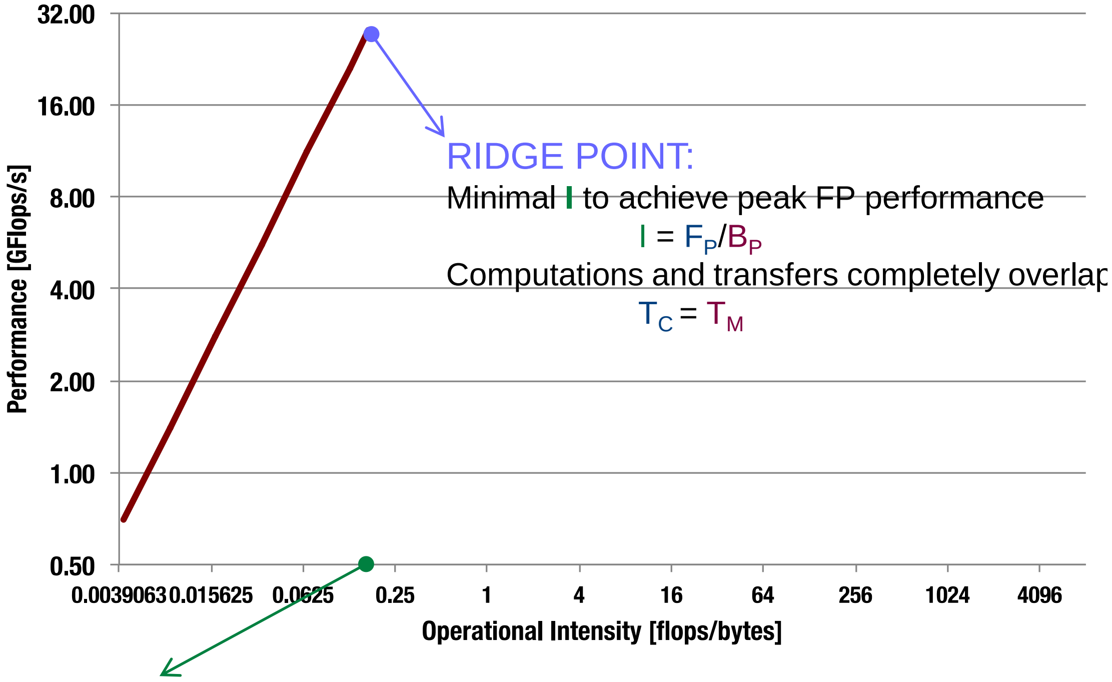
$\mathbf{T_C = \#flops/F_P}$ = 160flops/28 = 5.71 ns

$\mathbf{T_M = \#bytes/B_P}$ = 960bytes/168 = 5.71 ns

$\mathbf{T_T = max\{T_C,T_M\}}$ **=** 5.71 ns **($\mathbf{T_C=T_M}$)**

$\mathbf{F_A = \#flops/T_T}$ **=** 160flops/5.71 ns **= 28 Gflops/s**

Building the Cache-aware Roofline Model

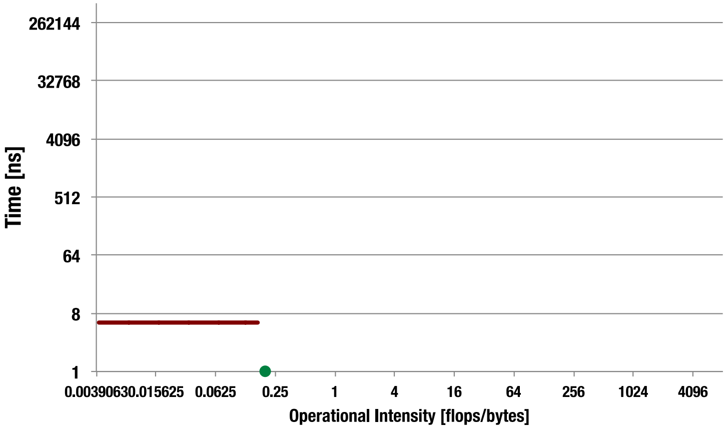


RIDGE POINT:
Minimal I to achieve peak FP performance

$I = F_P/B_P$

Computations and transfers completely overlap

$T_C = T_M$



I=flops/bytes=0.016 = 160flops/960bytes [20MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

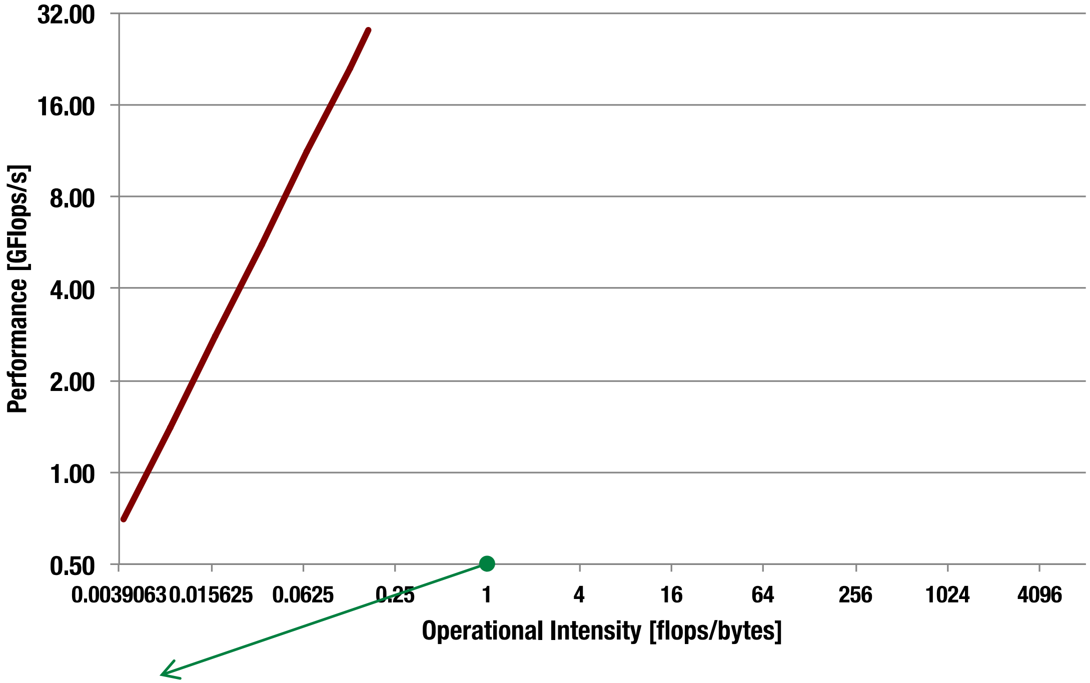
T_C = **#flops/F_P** = 160flops/28 = 5.71 ns

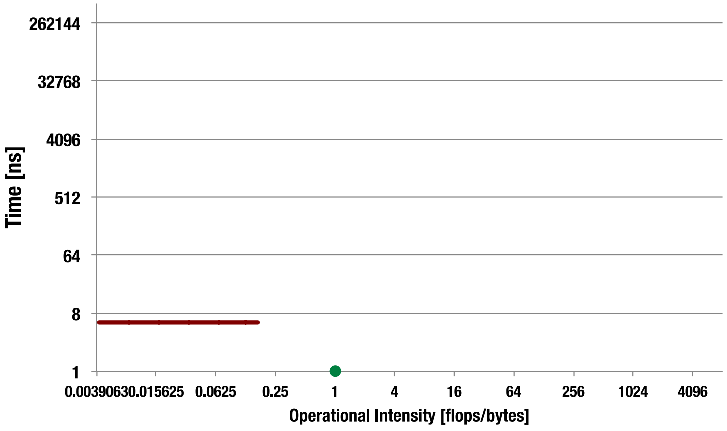
T_M = **#bytes/B_P** = 960bytes/168 = 5.71 ns

T_T = max{T_C,T_M} = 5.71 ns **(T_C=T_M)**

F_A = #flops/T_T = 160flops/5.71 ns **= 28 Gflops/s**

Building the Cache-aware Roofline Model





I=flops/bytes=1 = 960flops/960bytes [120MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

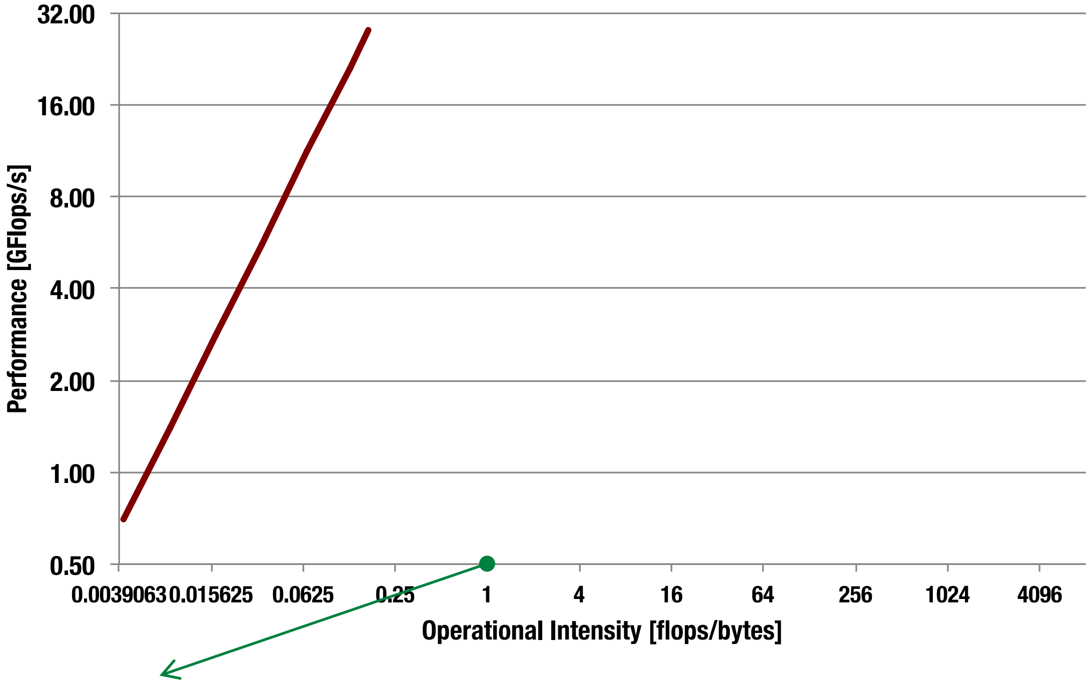
T_C = #flops/F_P = ?

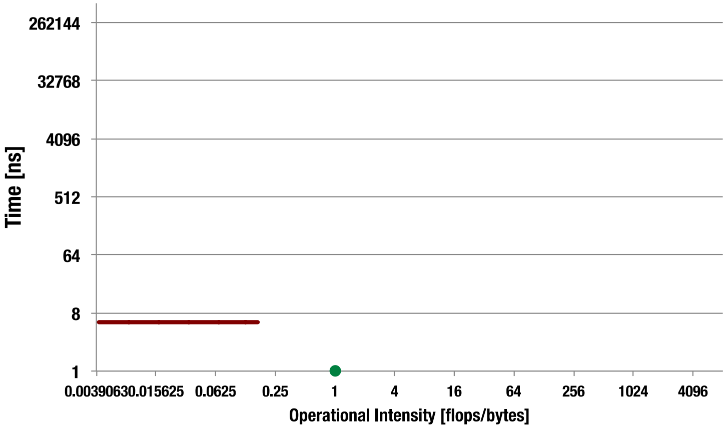
T_M = #bytes/B_P = ?

T_T = max{T_C,T_M} = ?

F_A = #flops/T_T = ?

Building the Cache-aware Roofline Model





I=flops/bytes=1 = 960flops/960bytes [120MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

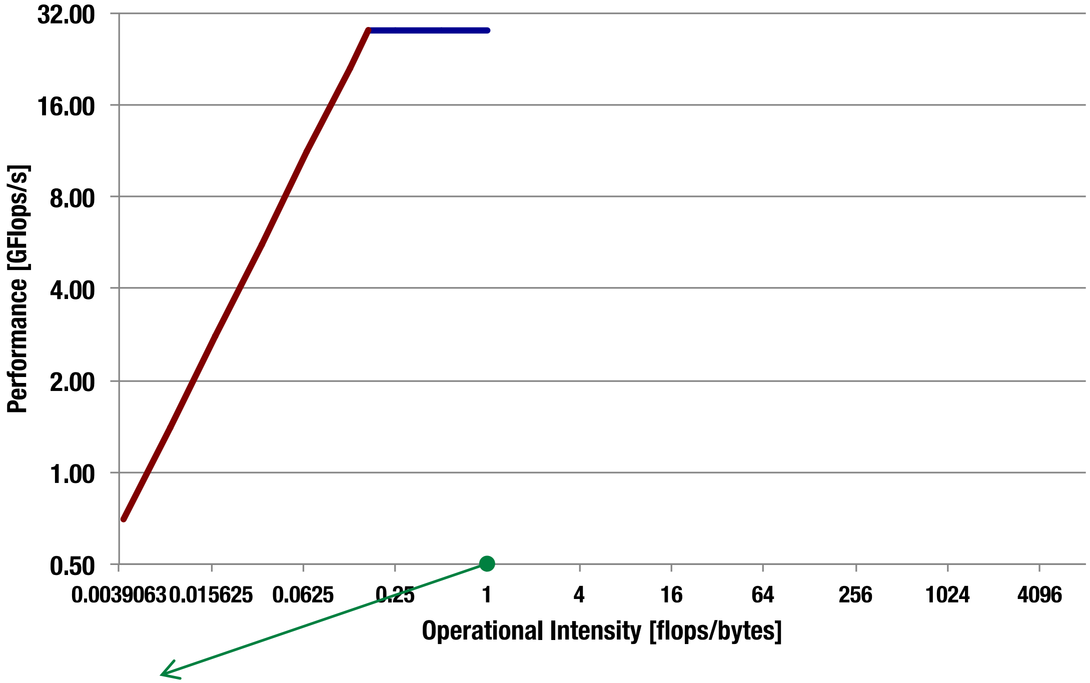
T_C = #flops/F_P = 960flops/28 = 34.29 ns

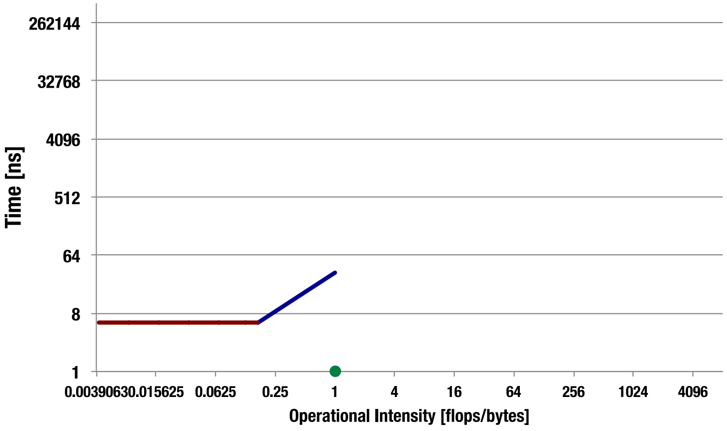
T_M = #bytes/B_P = 960bytes/168 = 5.71 ns

T_T = max{T_C,T_M} = 34.29 ns

F_A = #flops/T_T = 960flops/34.29ns **= 28 Gflops/s**

Building the Cache-aware Roofline Model





I=flops/bytes=1 = 960flops/960bytes [120MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

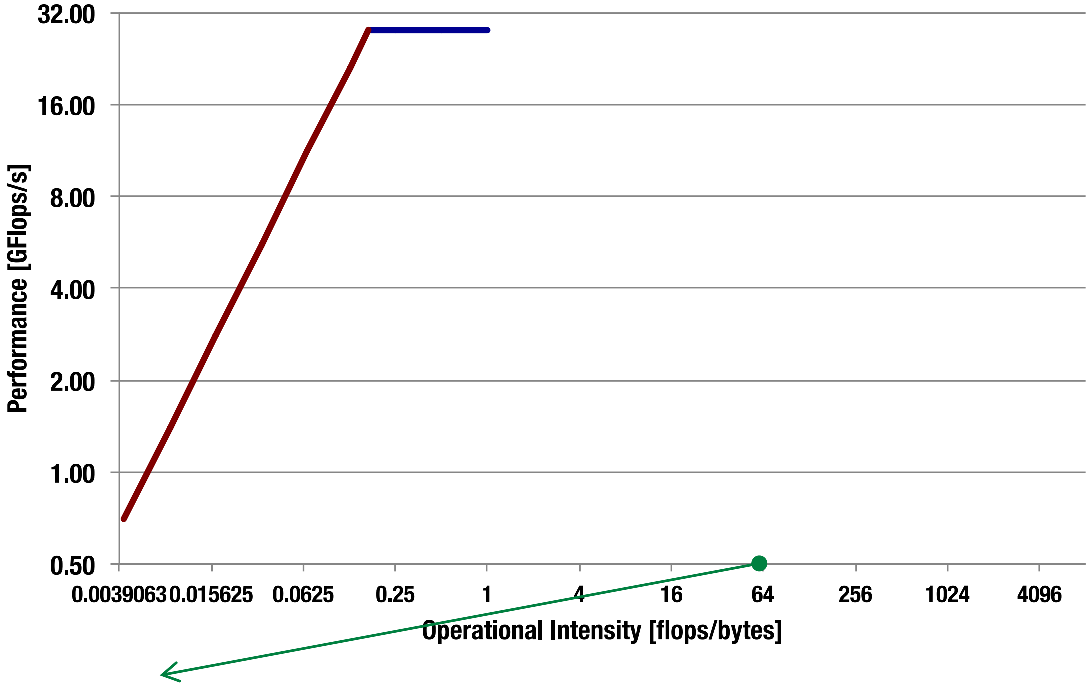
$T_C = \#flops/F_P$ = 960flops/28 = 34.29 ns

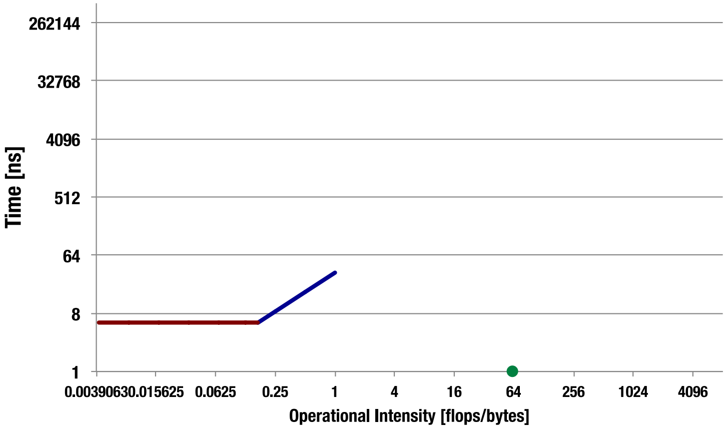
$T_M = \#bytes/B_P$ = 960bytes/168 = 5.71 ns

$T_T = \max\{T_C, T_M\}$ = 34.29 ns

$F_A = \#flops/T_T$ = 960flops/34.29ns = **28 Gflops/s**

Building the Cache-aware Roofline Model





I=flops/bytes= 64 = 61440flops/960bytes [7680MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

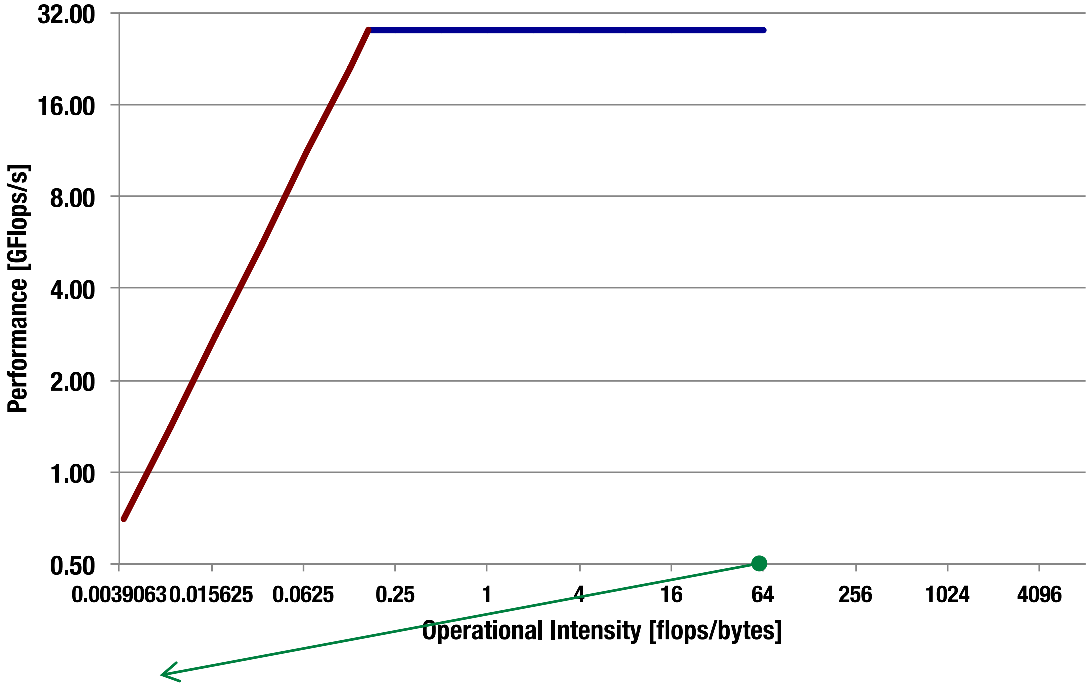
T_C = #flops/F_P = 7680flops/28 = 2194.29 ns

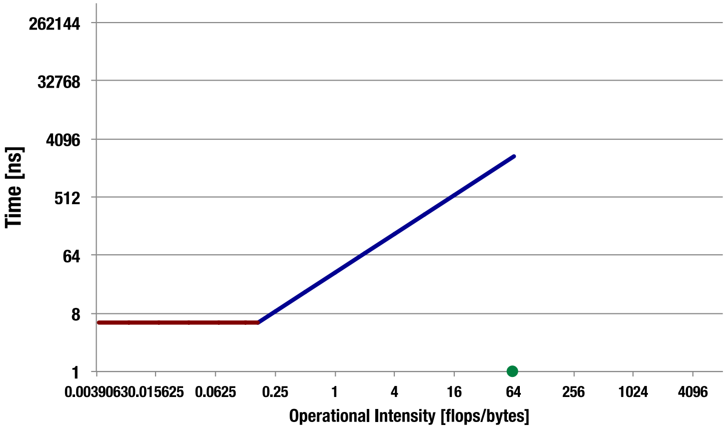
T_M = #bytes/B_P = 960bytes/168 = 5.71 ns

T_T = max{T_C,T_M} = 2194.29 ns

F_A = #flops/T_T = 28 Gflops/s

Building the Cache-aware Roofline Model





I=flops/bytes= 64 = 61440flops/960bytes [7680MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**
*256-bit AVX double-precision floating-point instructions		

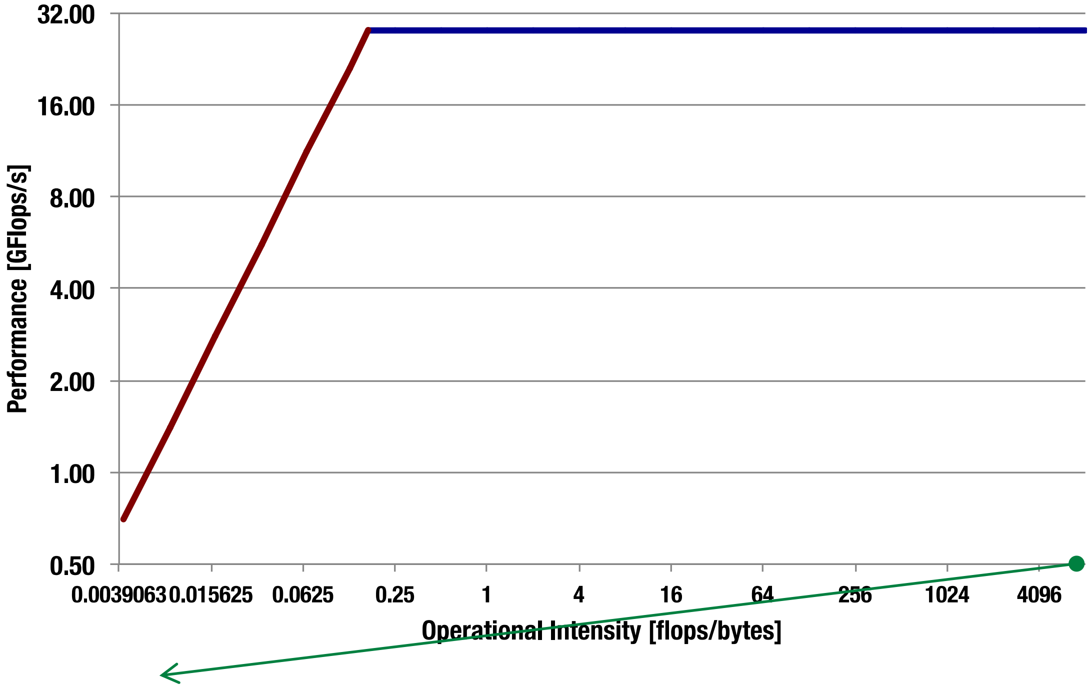
T_C = #flops/F_P = 7680flops/28 = 2194.29 ns

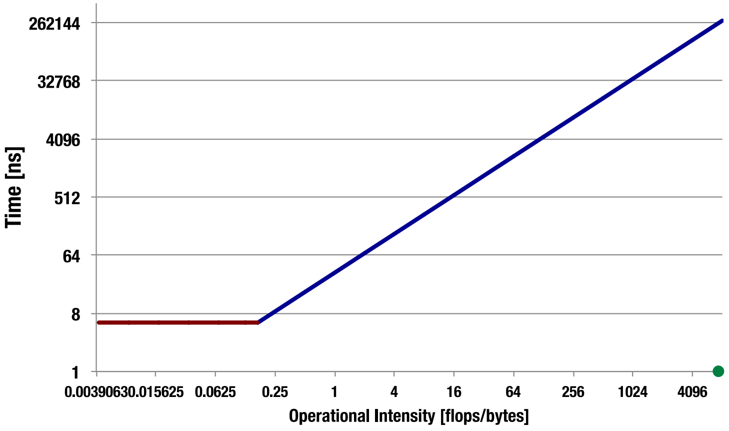
T_M = #bytes/B_P = 960bytes/168 = 5.71 ns

T_T = max{T_C,T_M} = 2194.29 ns

F_A = #flops/T_T = 28 Gflops/s

Building the Cache-aware Roofline Model





I=flops/bytes= 8192 = 7864320flops/960bytes [983040MAD/(**10x**(2LD+ST))*]

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**
*256-bit AVX double-precision floating-point instructions		

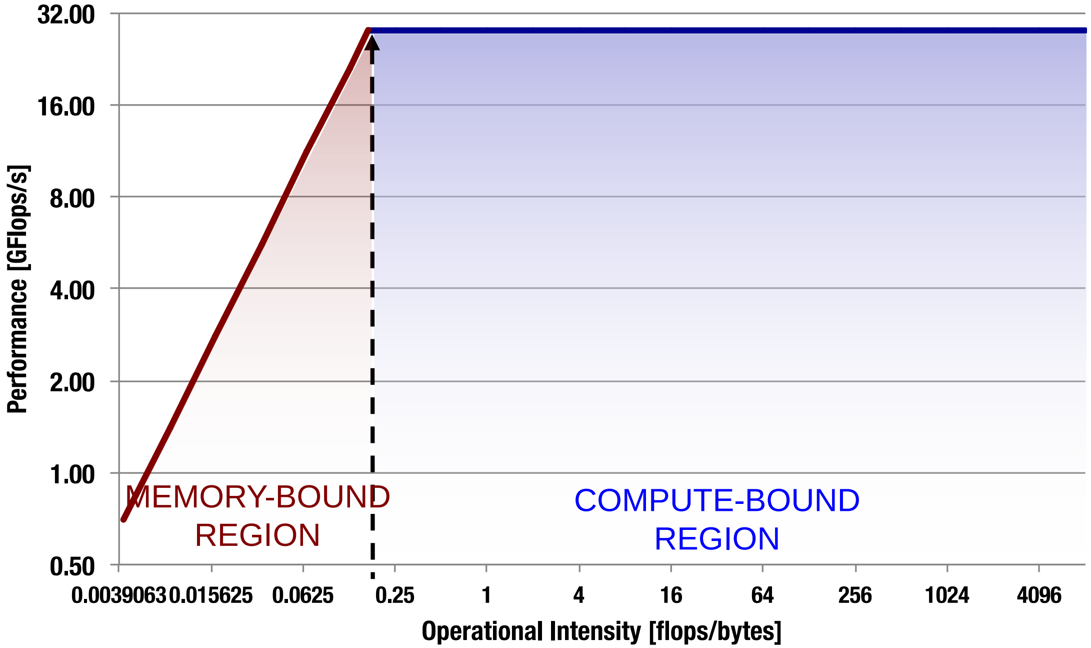
$T_C = \#flops/F_P = 280868.58$ ns

$T_M = \#bytes/B_P = 5.71$ ns

$T_T = \max\{T_C, T_M\} = 280868.58$ ns

$F_A = \#flops/T_T = 28$ **Gflops/s**

Building the Cache-aware Roofline Model



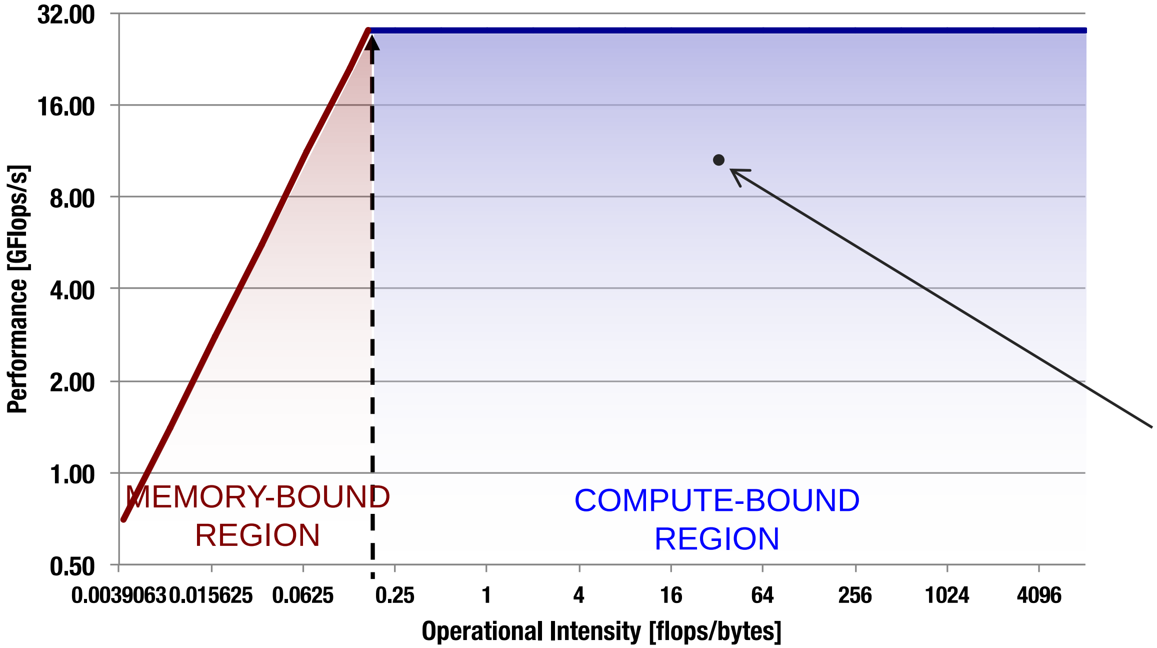
Application Characterization

- Memory-bound applications
- Compute-bound applications

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

Building the Cache-aware Roofline Model



Application Characterization

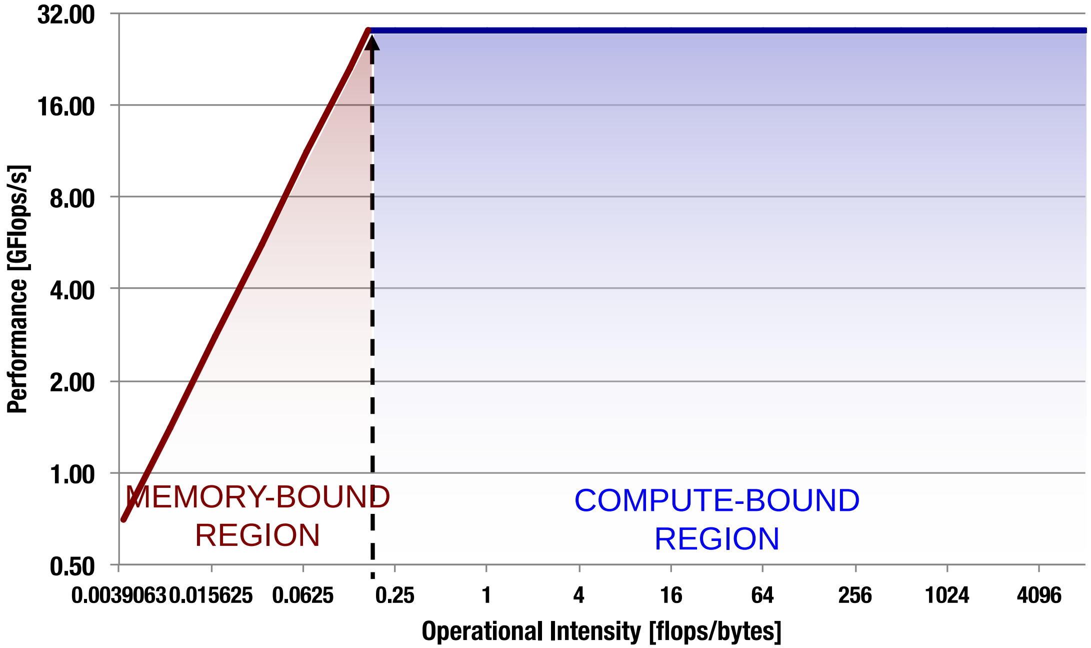
- Memory-bound applications
- Compute-bound applications

Application is a SINGLE POINT in the Cache-Aware Roofline Model!

i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Building the Cache-aware Roofline Model

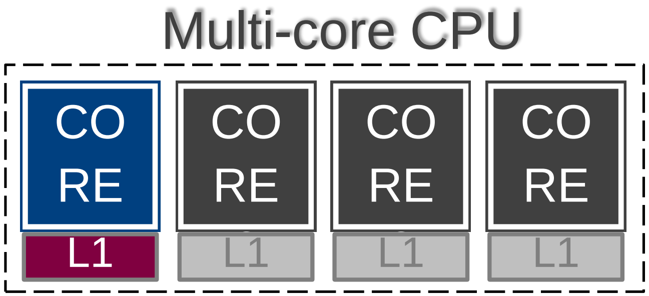


Model for 1 Core

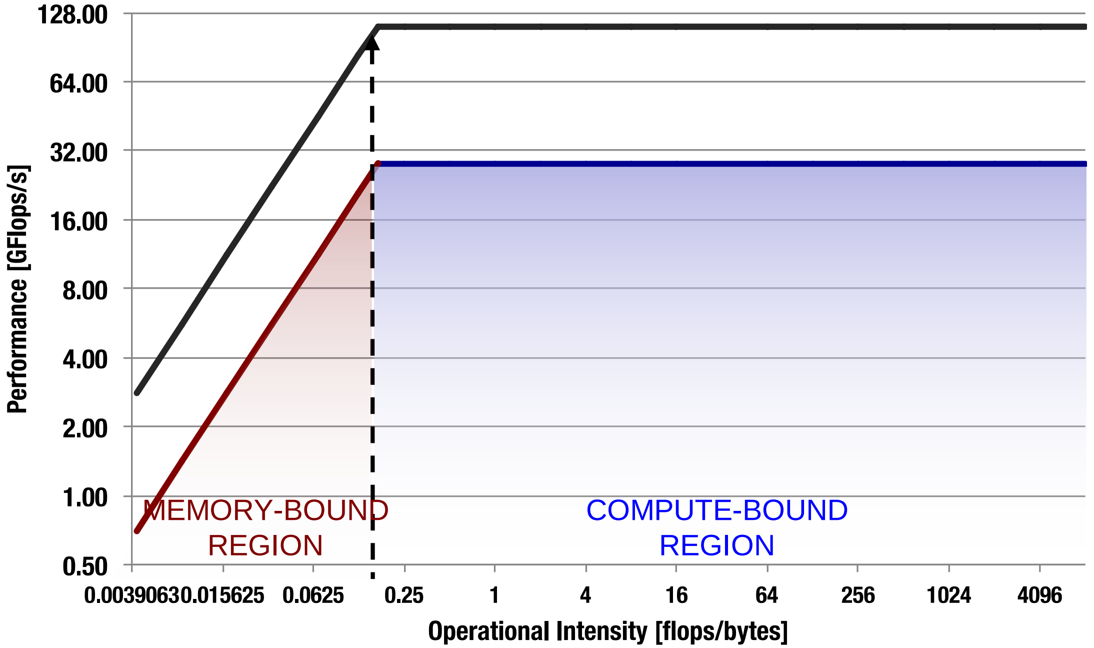
Model for 4 Cores?

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions



Building the Cache-aware Roofline Model



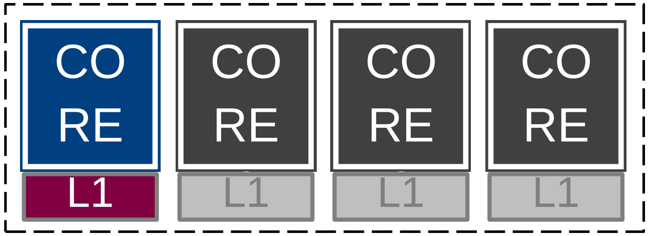
Model for 4 Cores

Model for 1 Core

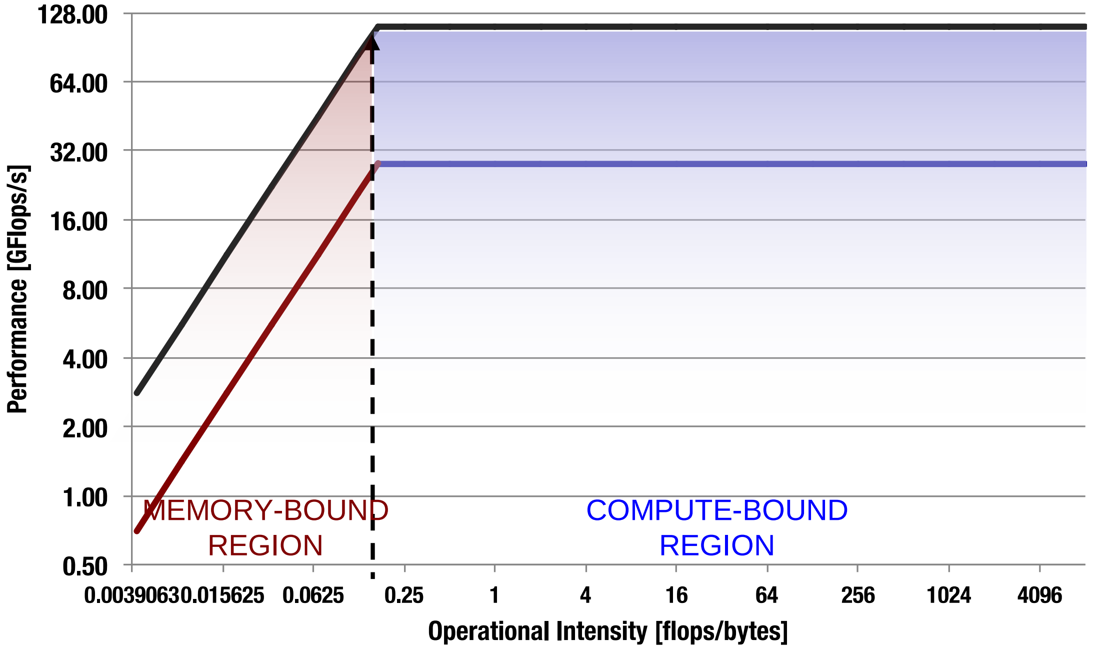
i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Multi-core CPU



Building the Cache-aware Roofline Model



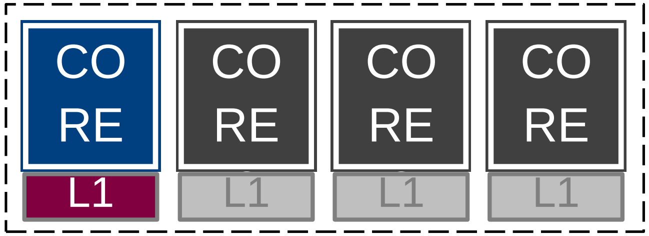
Model for 4 Cores

Model for 1 Core

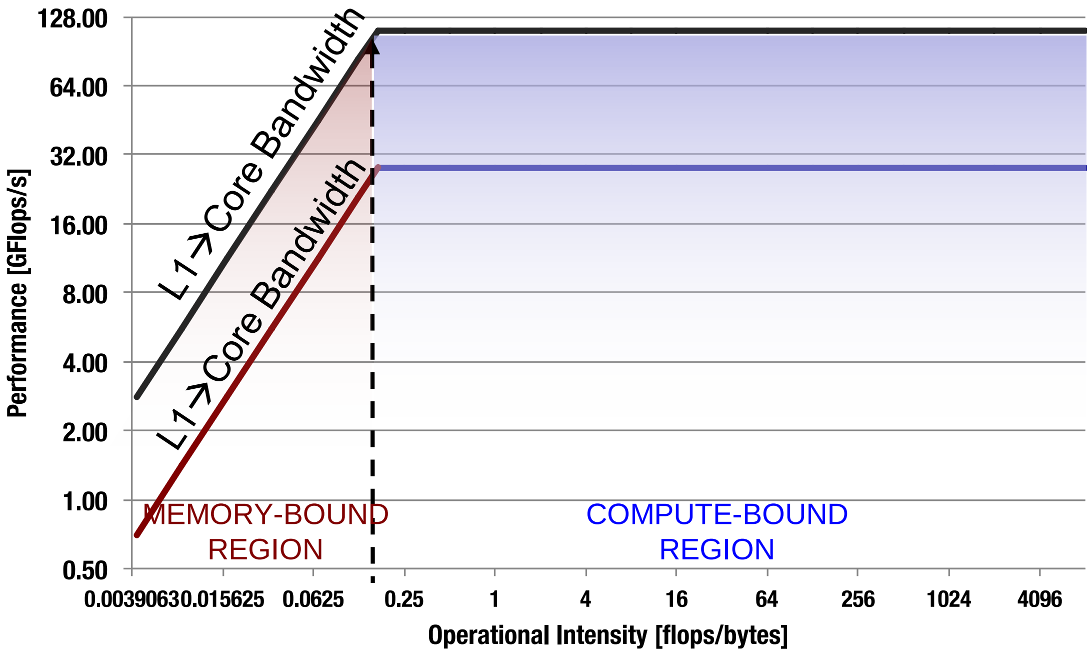
i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

Multi-core CPU



Building the Cache-aware Roofline Model



MODEL FOR 4 CORES

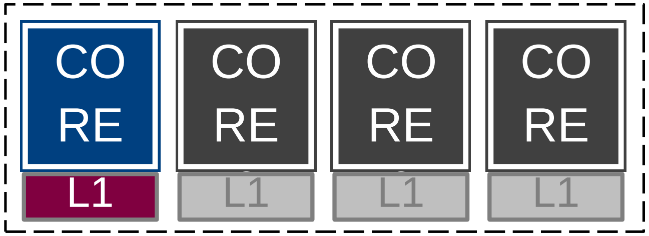
MODEL FOR 1 CORE

As for now, we just considered L1 bandwidth!

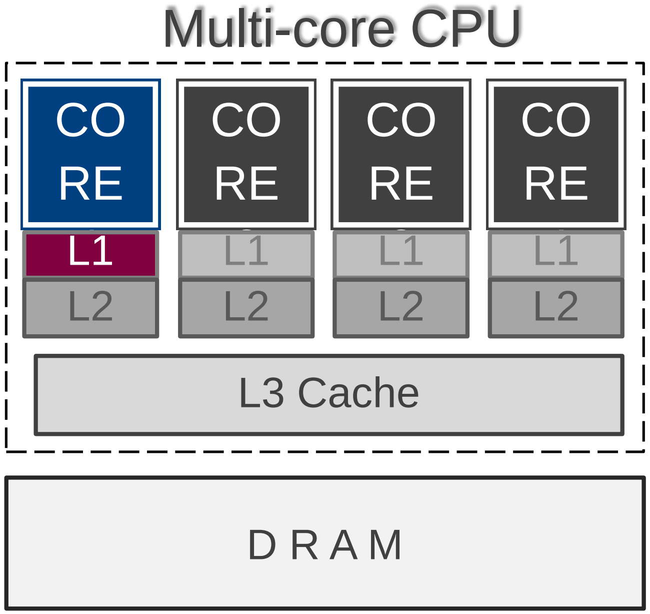
i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

Multi-core CPU



Multi-core Architectures
- Memory Hierarchy -



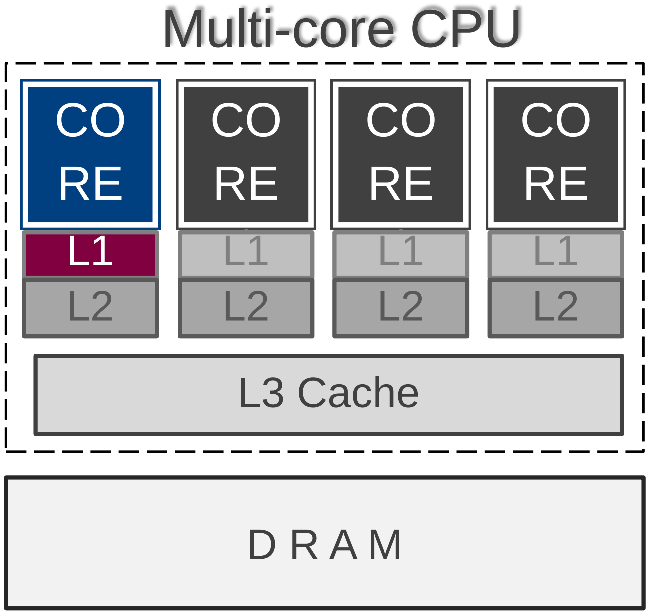
Memory Hierarchy

- Set of on-chip caches: private (L1, L2) or shared (L3)
- Global memory (DRAM)
- Caches hide the latency when accessing DRAM (also between successive cache levels)

i7 3770K Ivy Bridge	Perf. $[F_P]$ (GFlops/s)*	Bwidth L1→C $[B_P]$ (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Multi-core Architectures
- Memory Hierarchy -



Memory Hierarchy

- Set of on-chip caches: private (L1, L2) or shared (L3)
- Global memory (DRAM)
- Caches hide the latency when accessing DRAM (also between successive cache levels)

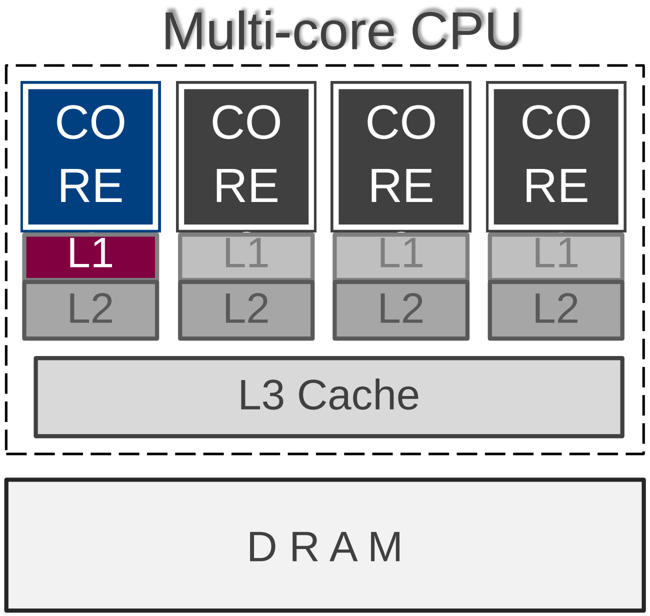
Cache-Aware Roofline Model

- Peak FP performance and L1 bandwidth obtained from processor's specifications (bottom table)
- **We need bandwidth from all other memory levels to the Core?**

i7 3770K Ivy Bridge	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Multi-core Architectures
- Memory Hierarchy -



Memory Hierarchy

- Set of on-chip caches: private (L1, L2) or shared (L3)
- Global memory (DRAM)
- Caches hide the latency when accessing DRAM (also between successive cache levels)

Cache-Aware Roofline Model

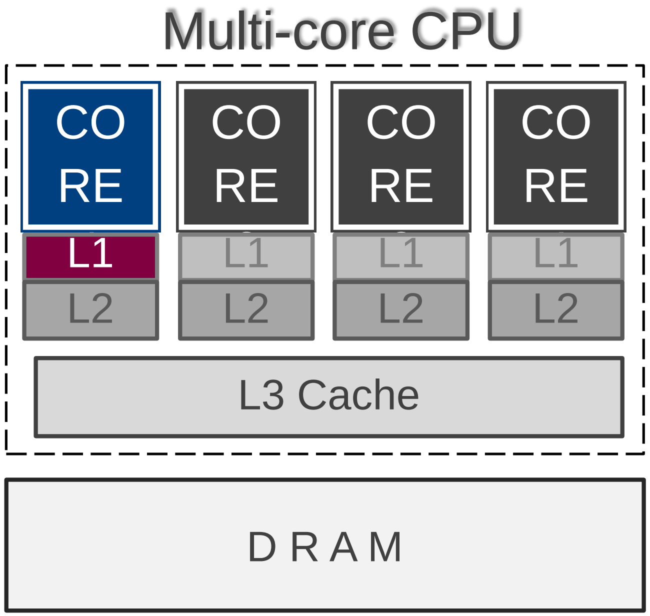
- Peak FP performance and L1 bandwidth obtained from processor's specifications (bottom table)
- **We need bandwidth from all other memory levels to the Core?**
  - Micro-benchmarks for architecture characterization

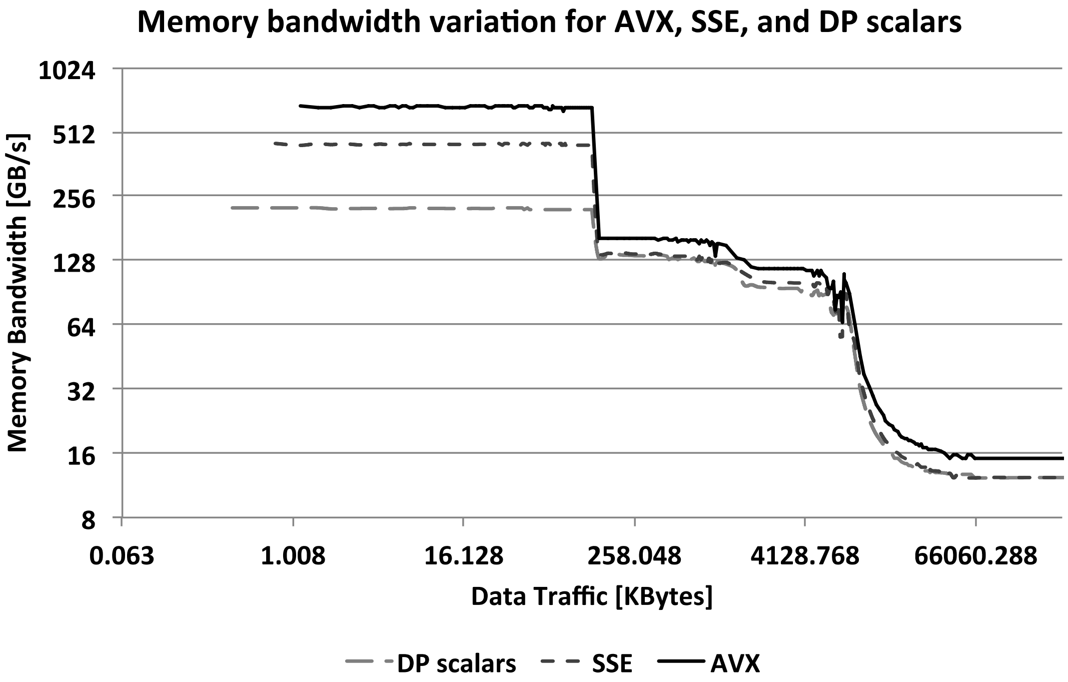
```
// AVX Assembly code: 2 Loads + 1 Store
vmovapd   0(%rax), %ymm0
vmovapd  32(%rax), %ymm1
vmovapd     %ymm2, 64(%rax)
vmovapd  96(%rax), %ymm3
vmovapd 128(%rax), %ymm4
vmovapd     %ymm5, 160(%rax)
…
```

i7 3770K Ivy Bridge	Perf. **[F_P]** (GFlops/s)*	Bwidth L1→C **[B_P]** (GB/s)*
1 Core	**28**	**168**
4 Cores	**112**	**672**

*256-bit AVX double-precision floating-point instructions

Multi-core Architectures
- Memory Hierarchy -



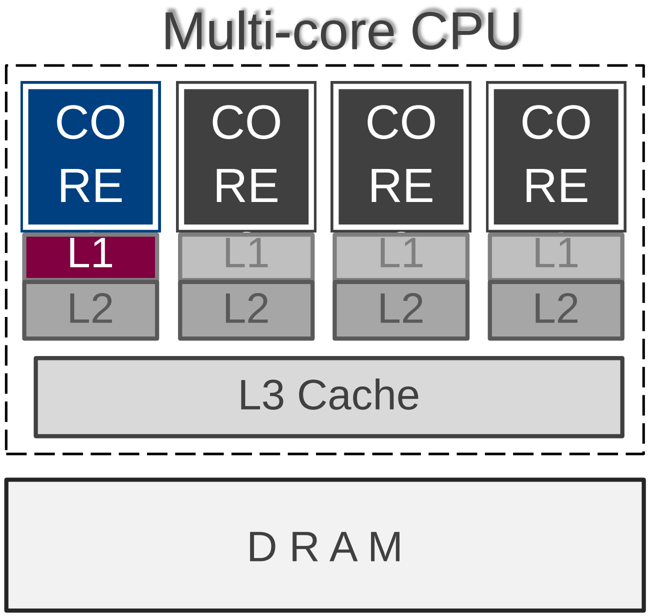


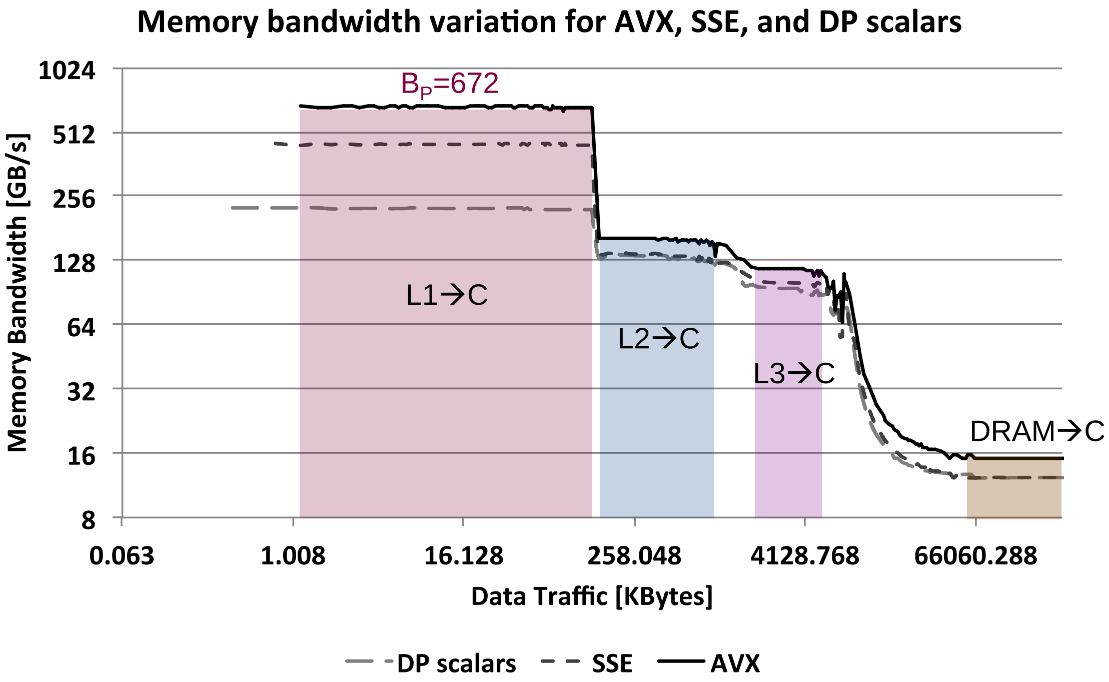
i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

```
// AVX Assembly code: 2 Loads + 1 Store
vmovapd    0(%rax), %ymm0
vmovapd   32(%rax), %ymm1
vmovapd      %ymm2, 64(%rax)
vmovapd   96(%rax), %ymm3
vmovapd  128(%rax), %ymm4
vmovapd      %ymm5, 160(%rax)
…
```

Multi-core Architectures
- Memory Hierarchy -



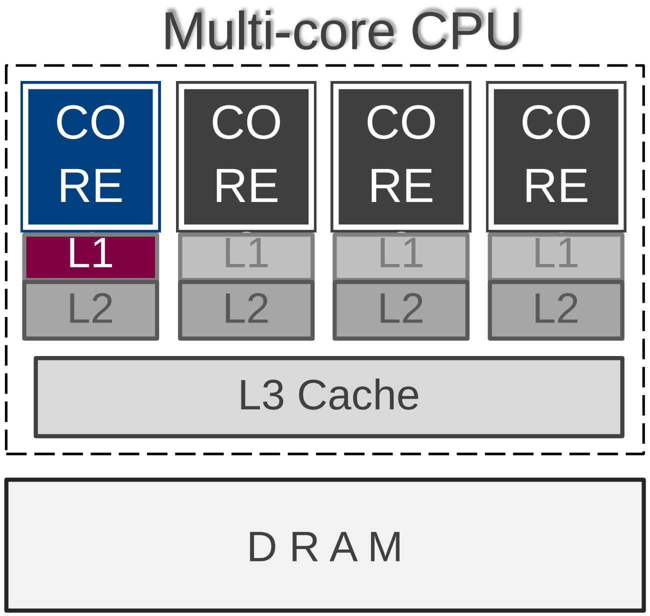


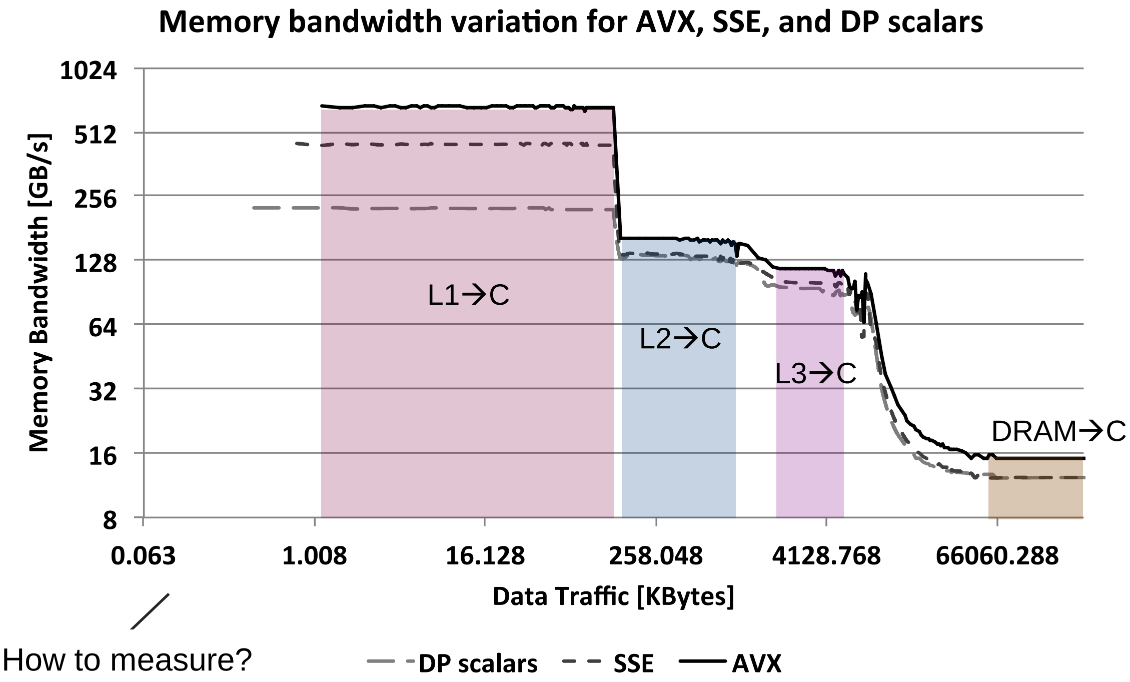
i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

```
// AVX Assembly code: 2 Loads + 1
Store
vmovapd   0(%rax), %ymm0
vmovapd  32(%rax), %ymm1
vmovapd      %ymm2, 64(%rax)
vmovapd  96(%rax), %ymm3
vmovapd 128(%rax), %ymm4
vmovapd      %ymm5, 160(%rax)
…
```

Multi-core Architectures
- Memory Hierarchy -



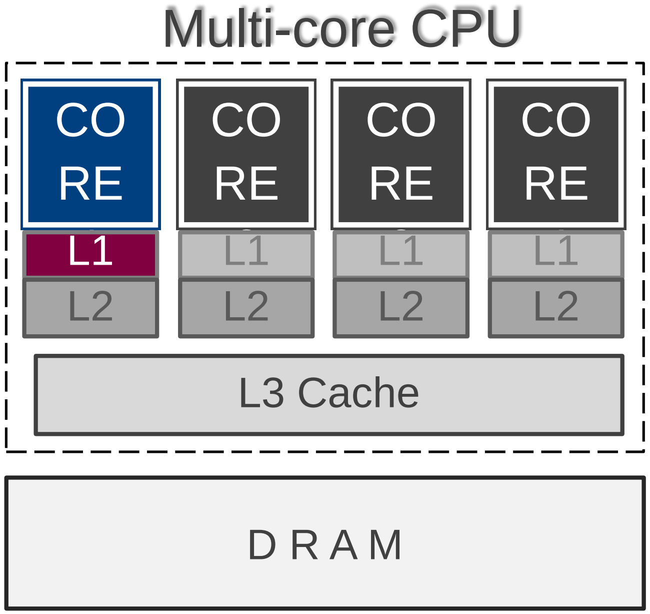


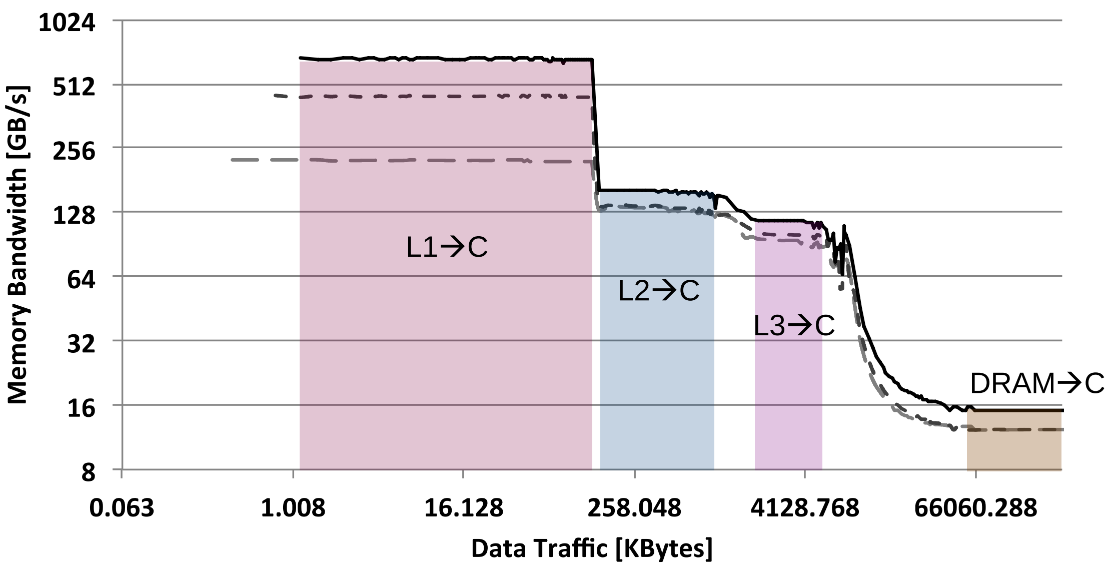
How to measure?

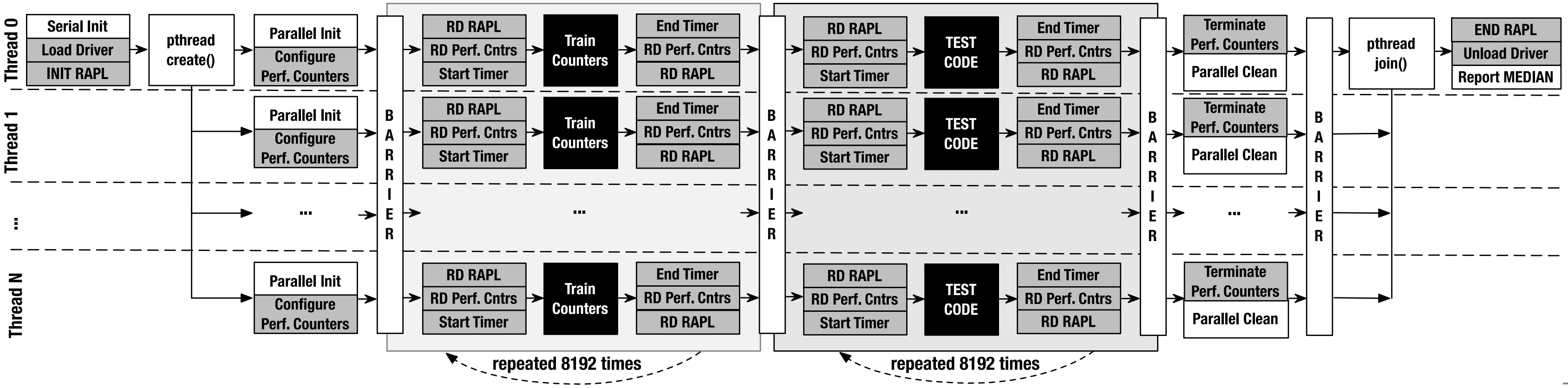
```
// Configured Performance Counters
CPU_CLK_UNHALTED.CORE/REF
MEM_UOP_RETIRED.ALL_LOADS
MEM_UOP_RETIRED.ALL_STORES
…
```

```
// AVX Assembly code: 2 Loads + 1
Store
vmovapd    0(%rax), %ymm0
vmovapd   32(%rax), %ymm1
vmovapd      %ymm2, 64(%rax)
vmovapd   96(%rax), %ymm3
vmovapd  128(%rax), %ymm4
vmovapd      %ymm5, 160(%rax)
…
```

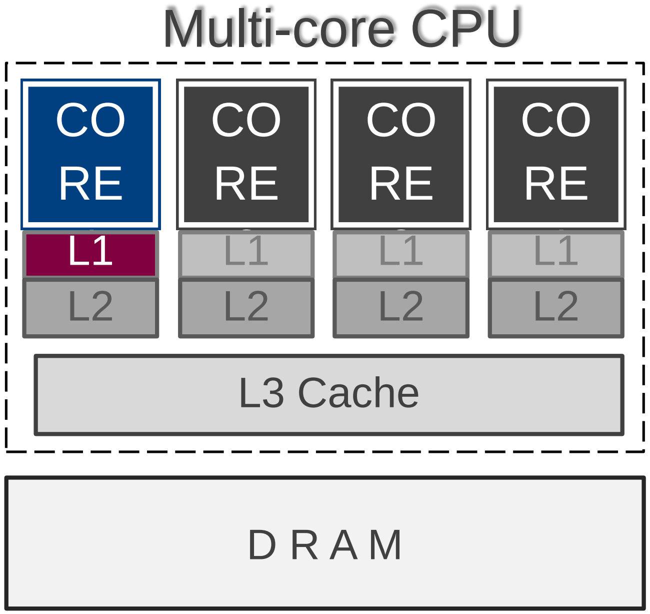
Multi-core Architectures
- Memory Hierarchy -



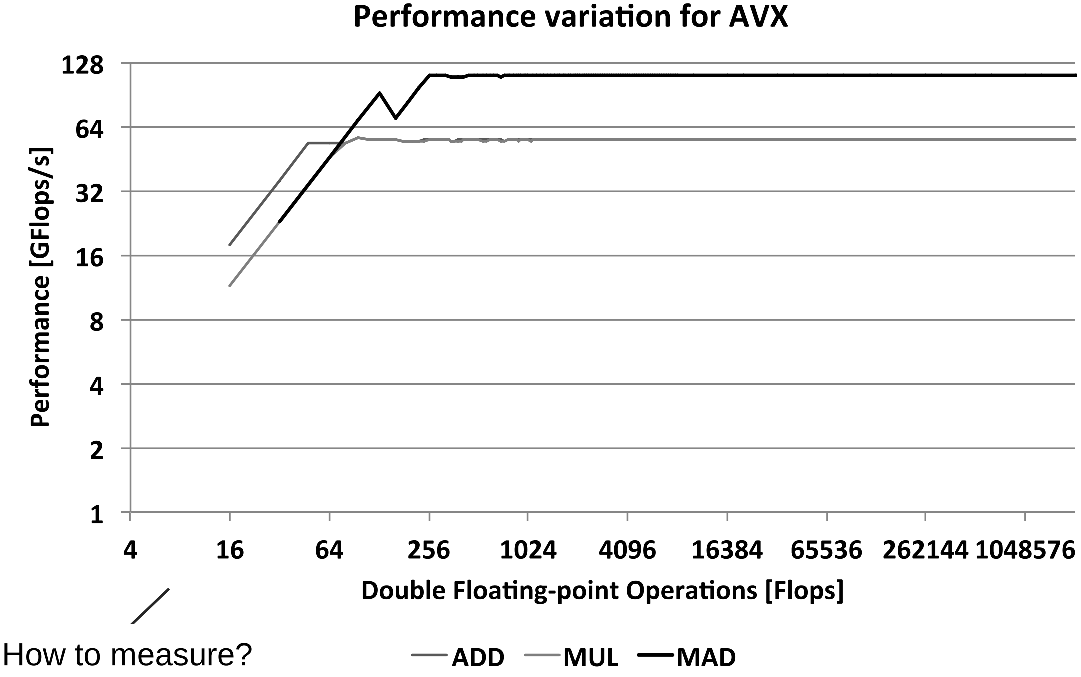

Memory bandwidth variation for AVX, SSE, and DP scalars



Multi-core Architectures
- Memory Hierarchy -



Performance variation for AVX



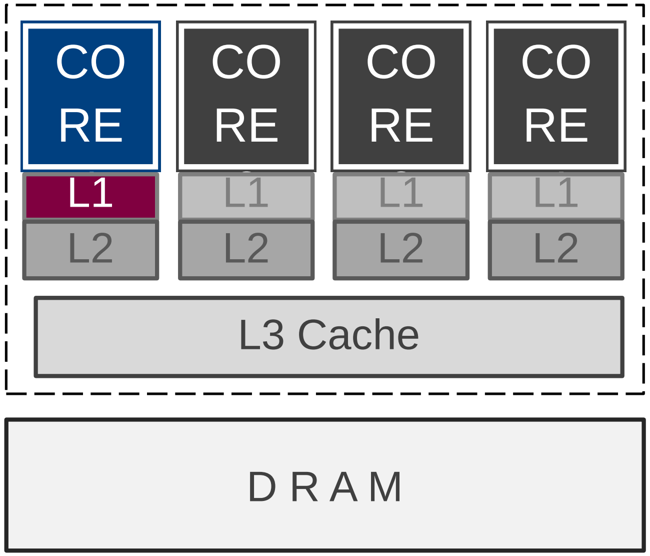
How to measure?

```
// Configured Performance Counters
CPU_CLK_UNHALTED.CORE/REF
FP_OPS_EXE_SSE_SCALAR_DBL
FP_OPS_EXE_SSE_FP_PACKED_DBL
SIMD_FP_256_PACKED_DBL
…
```

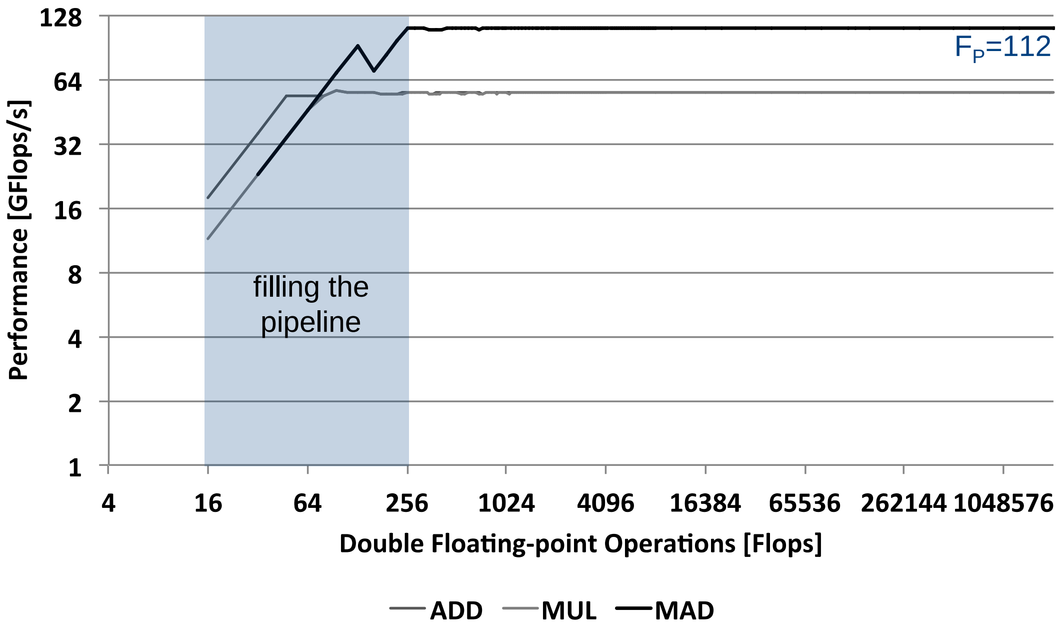
```
// AVX Assembly code: 2 Loads + 1
Store
vmulpd    %ymm0, %ymm0, %ymm0
vaddpd    %ymm1, %ymm1, %ymm1
vmulpd    %ymm2, %ymm2, %ymm2
vaddpd    %ymm3, %ymm3, %ymm3
vmulpd    %ymm4, %ymm4, %ymm4
vaddpd    %ymm5, %ymm5, %ymm5
…
```

Multi-core Architectures
- Memory Hierarchy -

Multi-core CPU



Performance variation for AVX

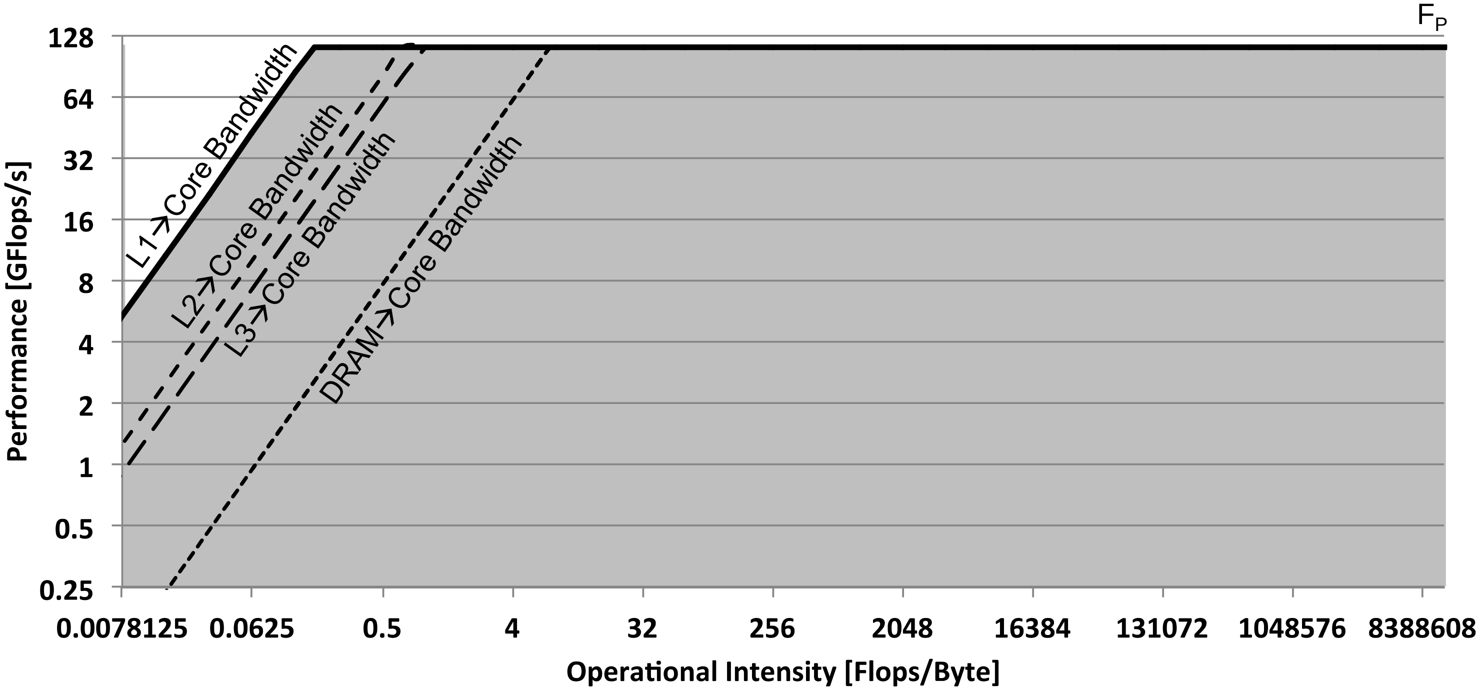


i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions

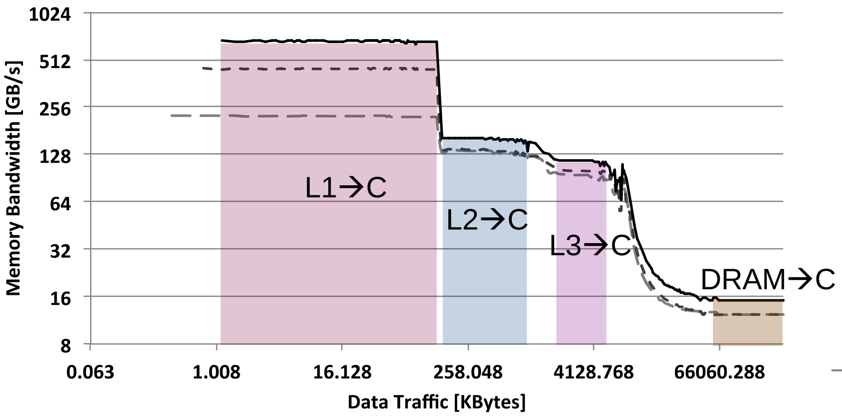
```
// AVX Assembly code: 2 Loads + 1 Store
vmulpd   %ymm0, %ymm0, %ymm0
vaddpd   %ymm1, %ymm1, %ymm1
vmulpd   %ymm2, %ymm2, %ymm2
vaddpd   %ymm3, %ymm3, %ymm3
vmulpd   %ymm4, %ymm4, %ymm4
vaddpd   %ymm5, %ymm5, %ymm5
…
```

Cache-Aware Roofline Model
- Putting it all together -

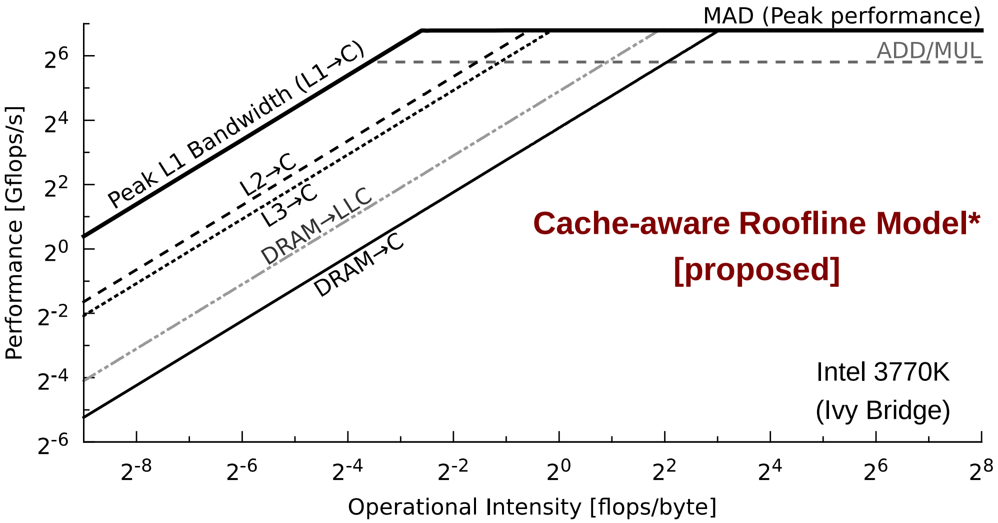


i7 3770K Ivy Bridge	Perf. [F_P] (GFlops/s)*	Bwidth L1→C [B_P] (GB/s)*
1 Core	28	168
4 Cores	112	672

*256-bit AVX double-precision floating-point instructions



Cache-aware Roofline Model: Hands On



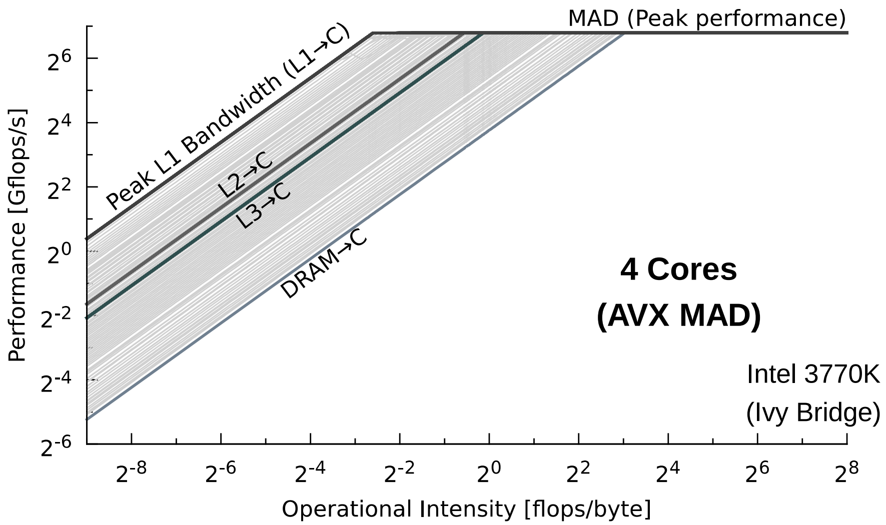
Cache-aware Roofline Model*

[proposed]

- Insightful **single plot model**
  - Shows performance limits of multicores
  - Redefined OI: flops and bytes as seen by core
  - Constructed once per architecture
- Considers **complete memory hierarchy**
  - Influence of caches and DRAM to performance
- Applicable to **other types of operations**
  - not only floating-point
- **Useful for:**
  - **Application** characterization and optimization
  - **Architecture** development and understanding

* *Ilic, A., Pratas, F. and Sousa, L., “Cache-aware Roofline Model: Upgrading the Loft”, IEEE Computer Architecture Letters, 2013*

Cache-aware Roofline Model: Hands On



- **Total Cache-aware Roofline Model**
  - Includes **all transitional states** (traversing the memory hierarchy and filling the pipeline)
  - Single-plot modeling for **different** types of compute and memory **operations**

- Insightful **single plot model**
  - Shows performance limits of multicores
  - Redefined OI: flops and bytes as seen by core
  - Constructed once per architecture

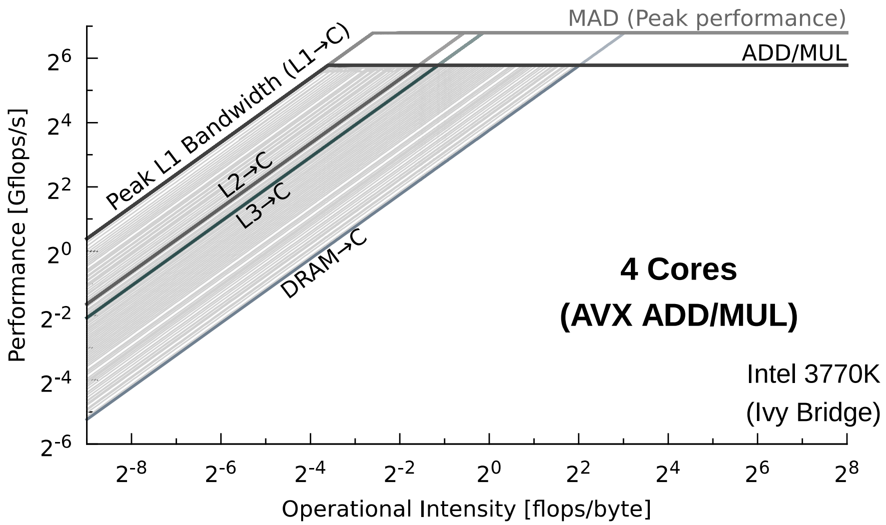
- Considers **complete memory hierarchy**
  - Influence of caches and DRAM to performance

- Applicable to **other types of operations**
  - not only floating-point

- **Useful for:**
  - **Application** characterization and optimization
  - **Architecture** development and understanding

* *Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

Cache-aware Roofline Model: Hands On



- **Total Cache-aware Roofline Model**
  - Includes **all transitional states** (traversing the memory hierarchy and filling the pipeline)
  - Single-plot modeling for **different** types of compute and memory **operations**

- Insightful **single plot model**
  - Shows performance limits of multicores
  - Redefined OI: flops and bytes as seen by core
  - Constructed once per architecture

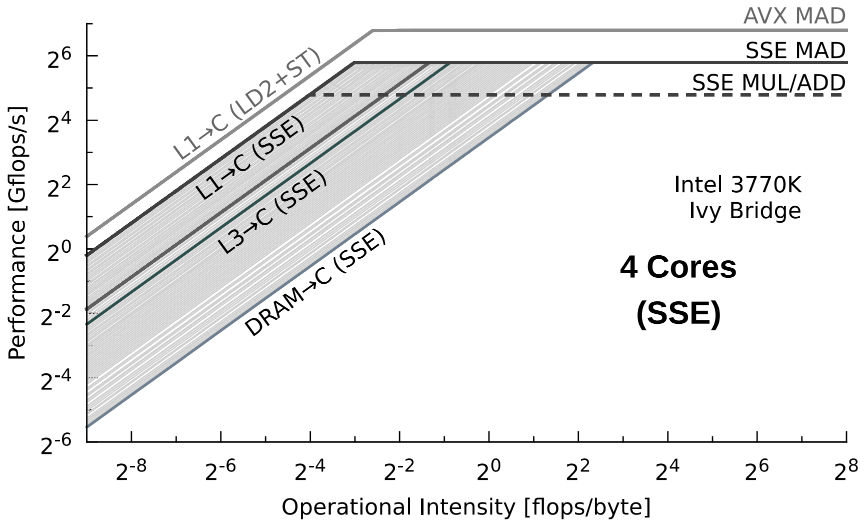
- Considers **complete memory hierarchy**
  - Influence of caches and DRAM to performance

- Applicable to **other types of operations**
  - not only floating-point

- **Useful for:**
  - **Application** characterization and optimization
  - **Architecture** development and understanding

** Ilic, A., Pratas, F. and Sousa, L., “Cache-aware Roofline Model: Upgrading the Loft”, IEEE Computer Architecture Letters, 2013*

Cache-aware Roofline Model: Hands On

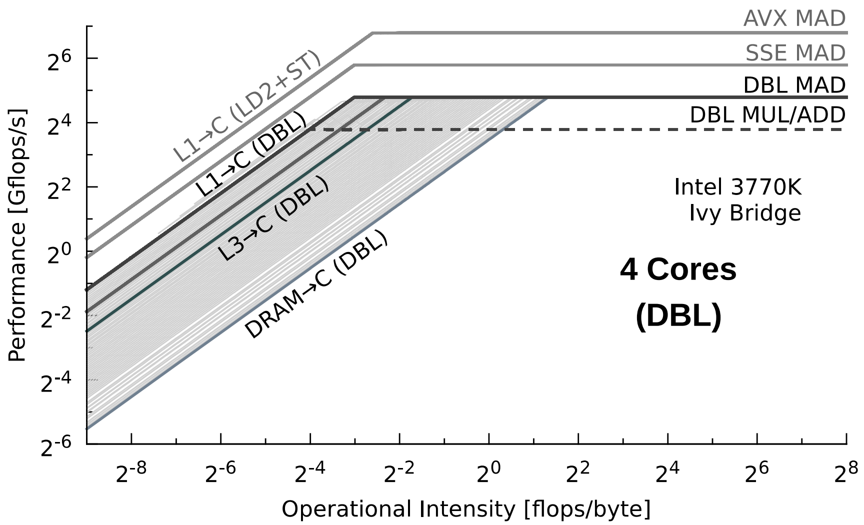


- **Total Cache-aware Roofline Model**
  - Includes **all transitional states** (traversing the memory hierarchy and filling the pipeline)
  - Single-plot modeling for **different** types of compute and memory **operations**

- Insightful **single plot model**
  - Shows performance limits of multicores
  - Redefined OI: flops and bytes as seen by core
  - Constructed once per architecture
- Considers **complete memory hierarchy**
  - Influence of caches and DRAM to performance
- Applicable to **other types of operations**
  - not only floating-point
- **Useful for:**
  - **Application** characterization and optimization
  - **Architecture** development and understanding

** Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

Cache-aware Roofline Model: Hands On



- **Total Cache-aware Roofline Model**
  - Includes **all transitional states** (traversing the memory hierarchy and filling the pipeline)
  - Single-plot modeling for **different** types of compute and memory **operations**

- Insightful **single plot model**
  - Shows performance limits of multicores
  - Redefined OI: flops and bytes as seen by core
  - Constructed once per architecture
- Considers **complete memory hierarchy**
  - Influence of caches and DRAM to performance
- Applicable to **other types of operations**
  - not only floating-point
- **Useful for:**
  - ***Application*** characterization and optimization
  - ***Architecture*** development and understanding

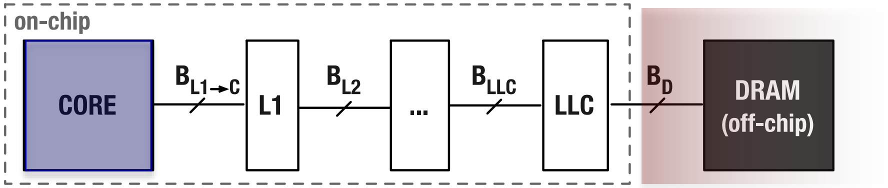
* *Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

WHAT IS HOT (WHAT IS NOT)?

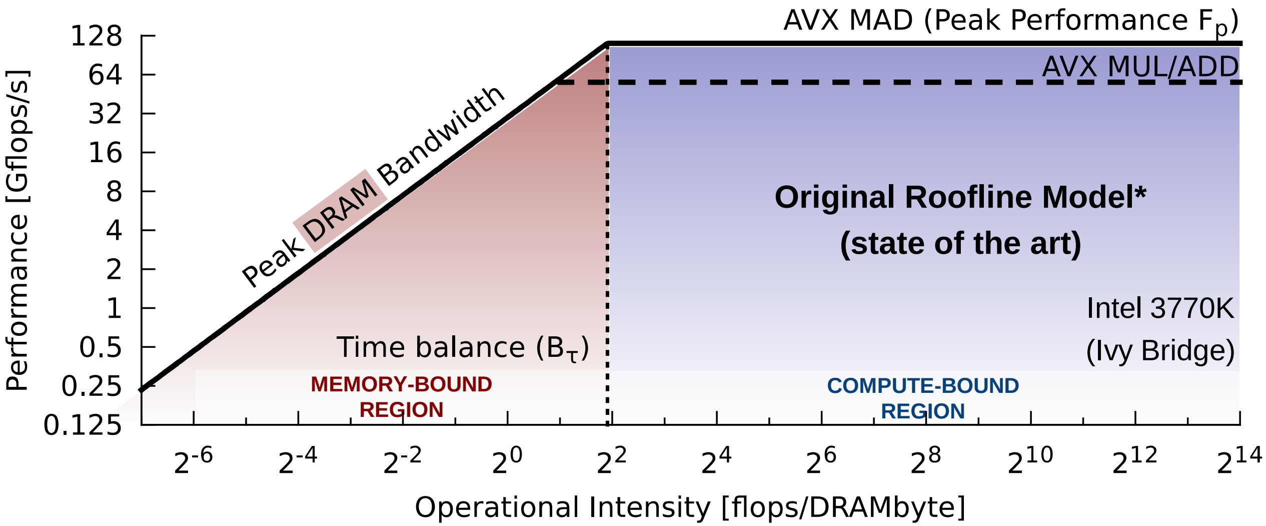
- APPLICATION CHARACTERIZATION -

The Original Roofline Model*

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)



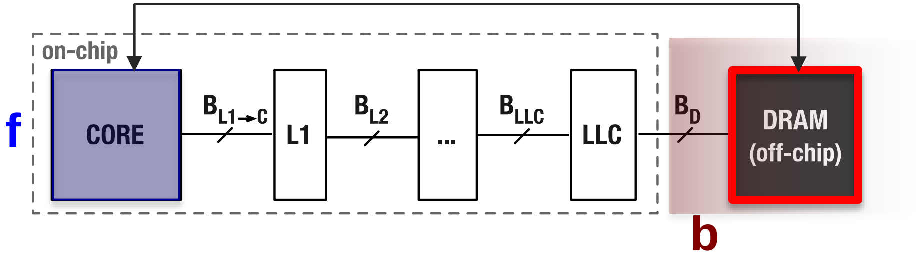
- **Performance**: Computations (*flops*) and communication (*bytes*) overlap in time

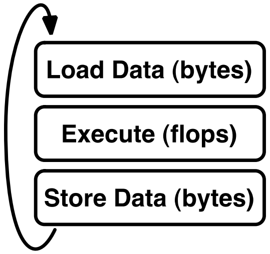


** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

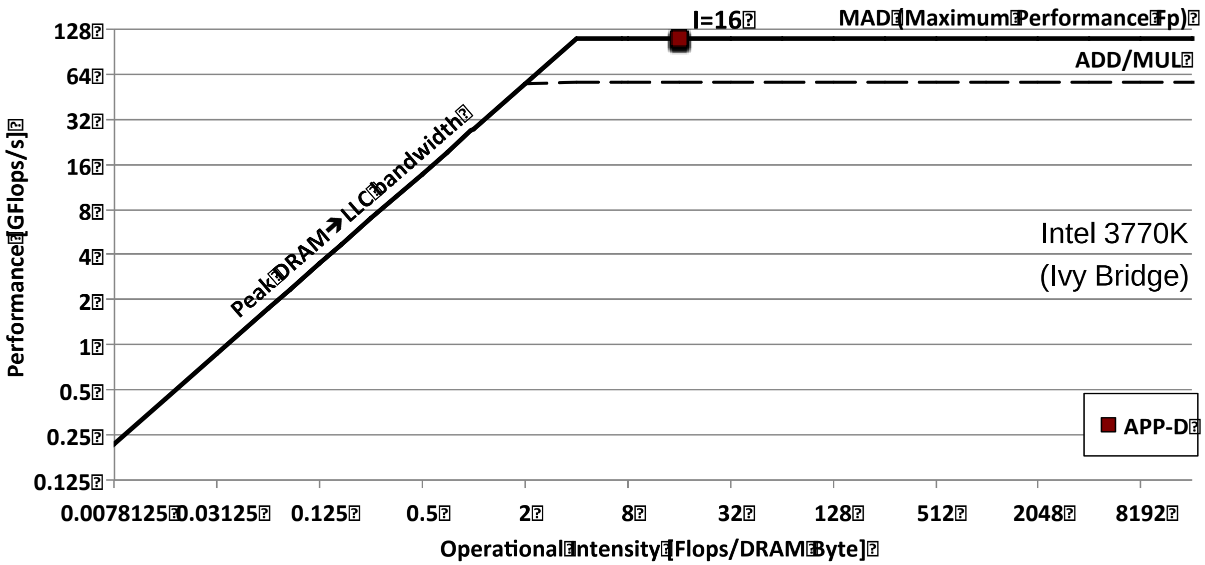
The Original Roofline Model: Hands On

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)





APP-D (data traffic from DRAM)



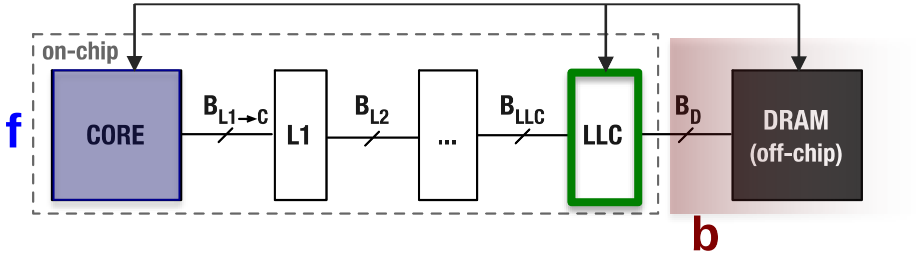
$$I=(\Sigma f_l)/(\Sigma b_l)$$

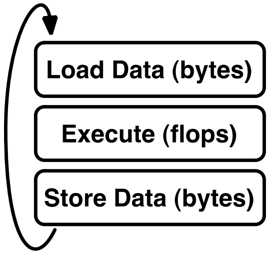
I is constant

* *Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

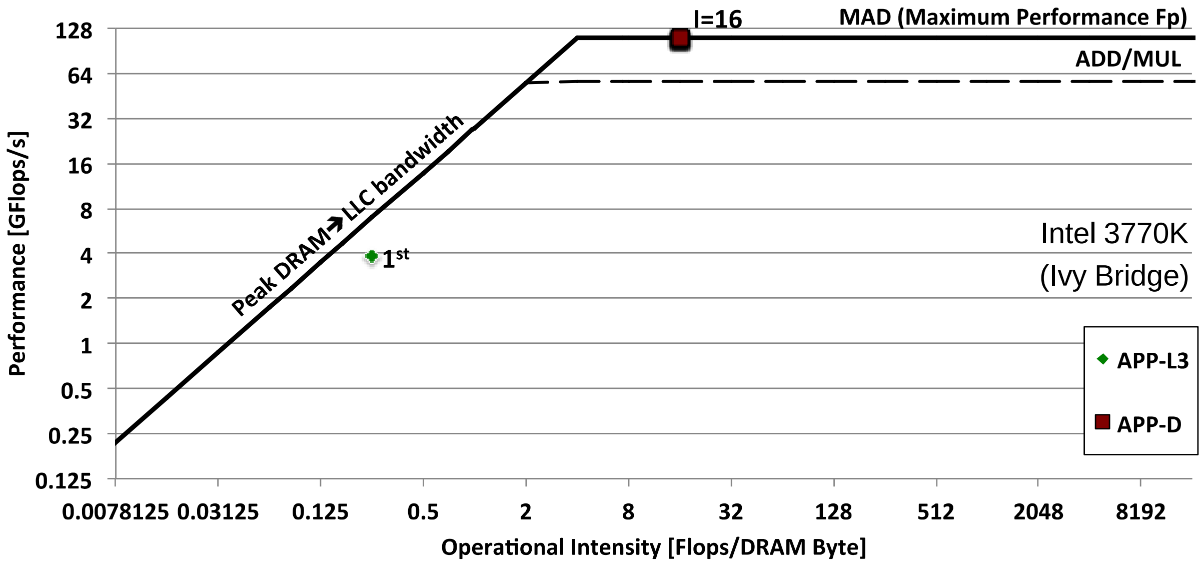
The Original Roofline Model: Hands On

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)





APP-L3 (data fits in L3)

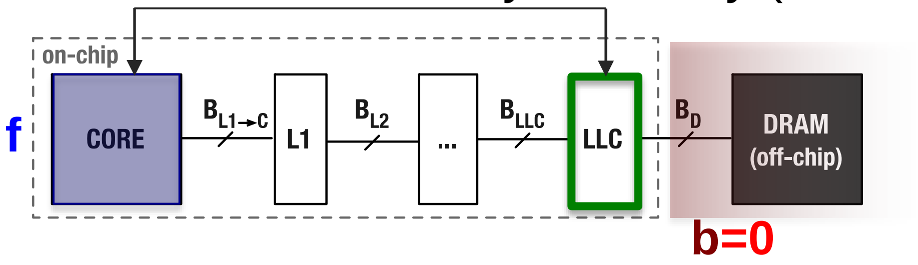


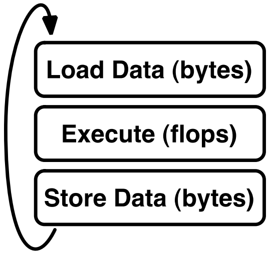
$$I_1 = f_1 / b_1$$

** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

The Original Roofline Model: Hands On

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)

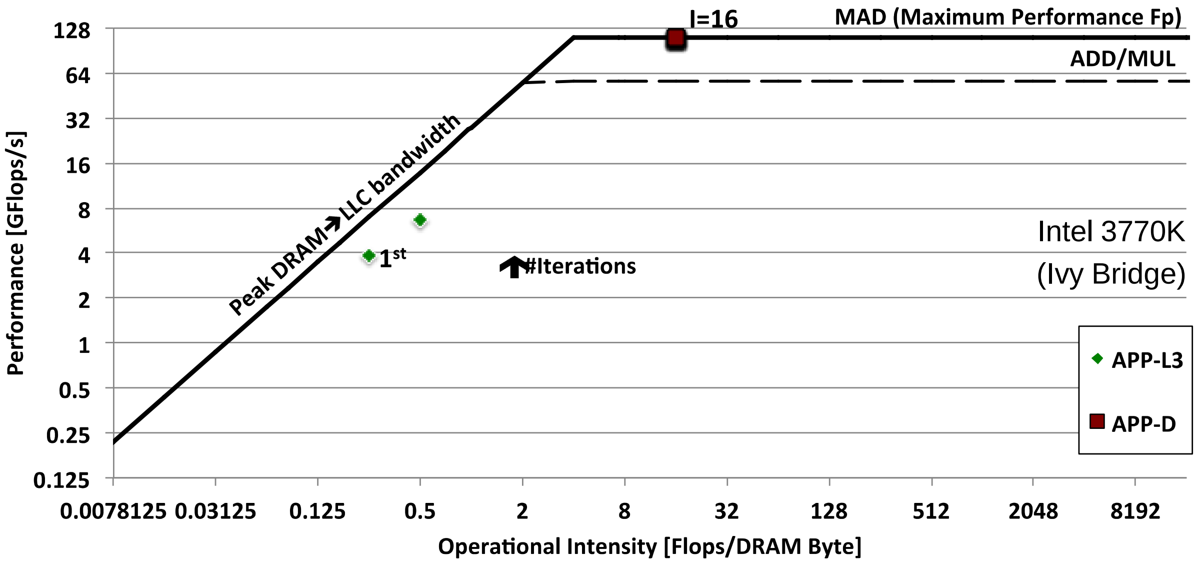




APP-L3 (data fits in L3)

$$I_1=f_1/b_1$$

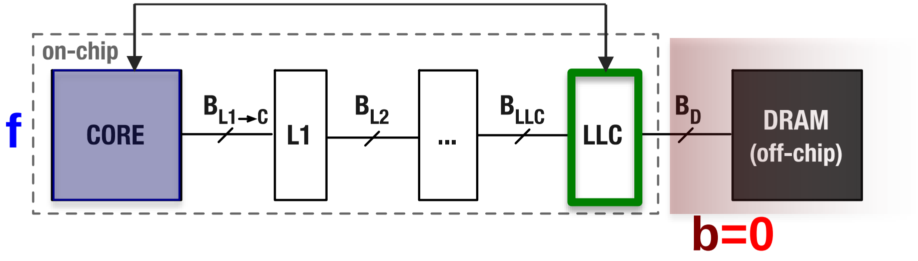
$$I_2=(f_1+f_2)/b_1$$

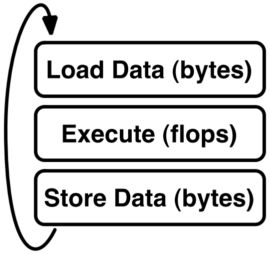


* Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)

The Original Roofline Model: Hands On

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)



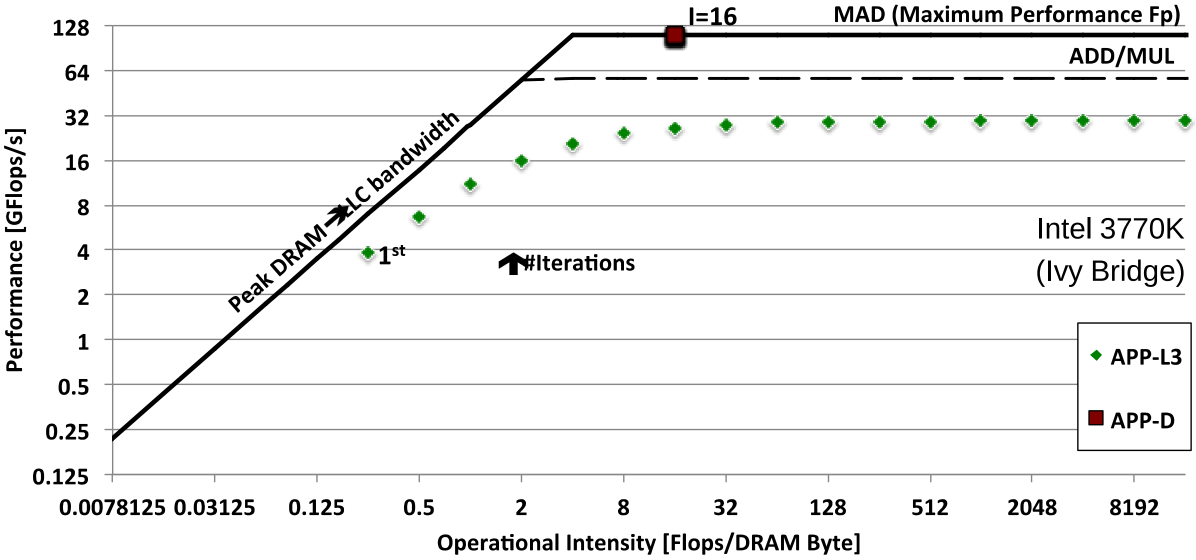


Load Data (bytes)

Execute (flops)

Store Data (bytes)

APP-L3 (data fits in L3)



$I_1=f_1/b_1$

$I_2=(f_1+f_2)/b_1$

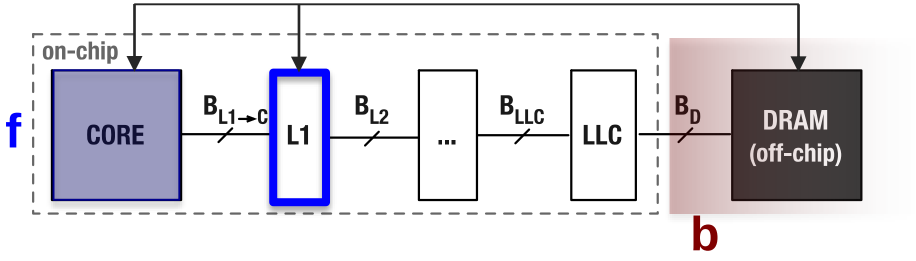
$I_i=(\Sigma f_i)/b_1$

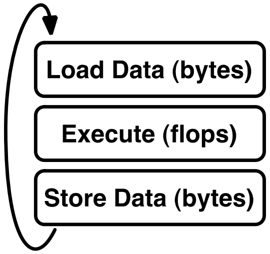
I is variable

* *Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

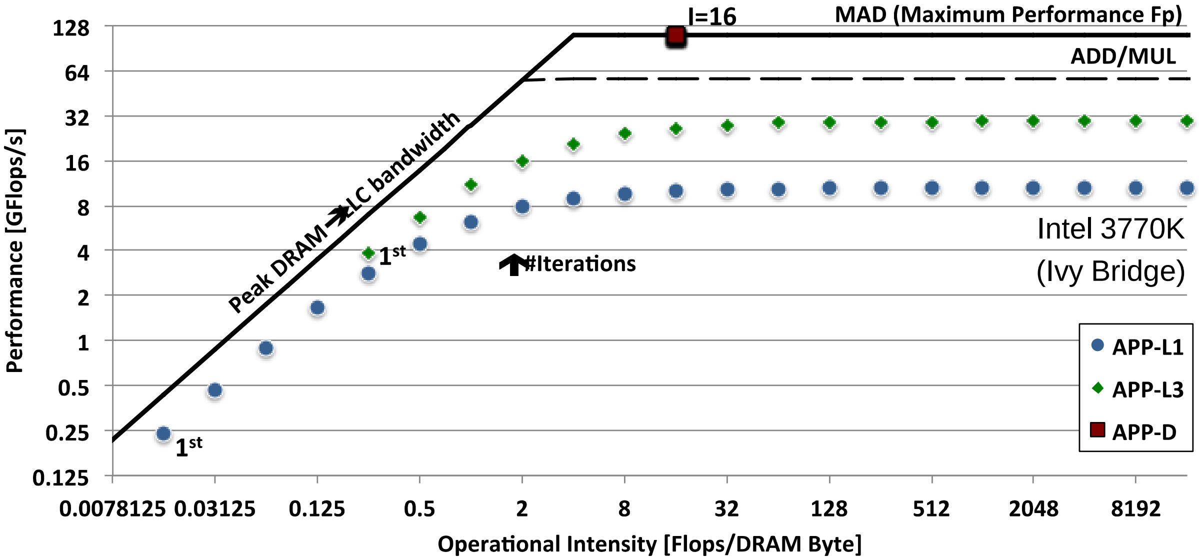
The Original Roofline Model: Hands On

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)





APP-L1 (data fits in L1)



$I_1=f_1/b_1$

$I_2=(f_1+f_2)/b_1$

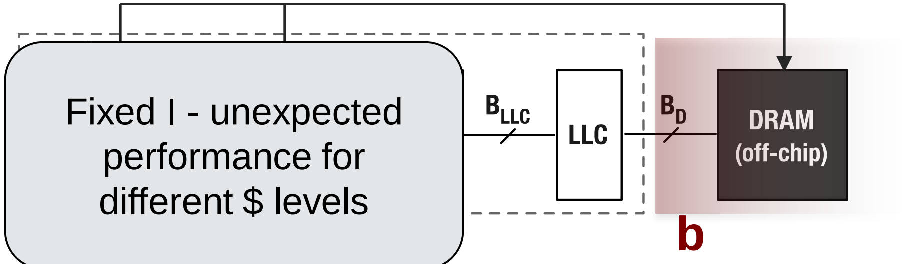
$I_i=(\Sigma f_i)/b_1$

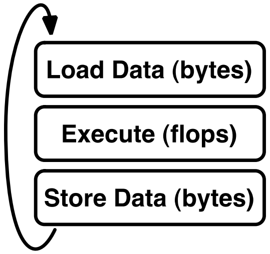
I is variable

* *Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

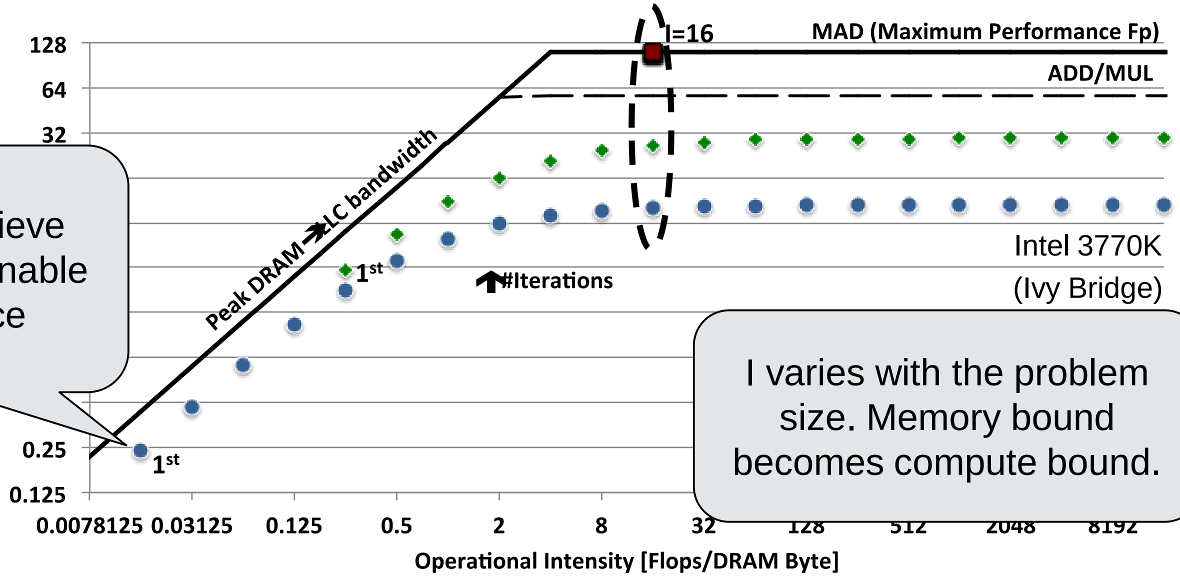
The Original Roofline Model: Hands On

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)





Fixed I - unexpected performance for different $ levels



Does not achieve maximum attainable performance

I varies with the problem size. Memory bound becomes compute bound.

$I_1 = f_1 / b_1$

$I_2 = (f_1 + f_2) / b_1$

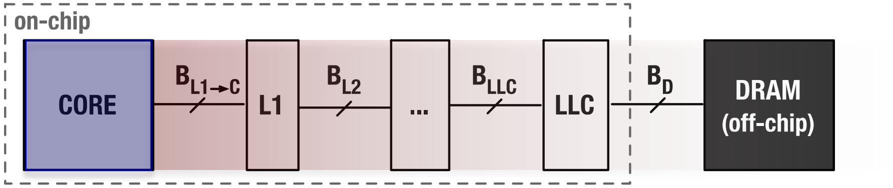
$I_i = (\Sigma f_i) / b_1$

I is variable

* *Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

Cache-aware Roofline Model

- **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)



on-chip

CORE

$B_{L1\rightarrow C}$

L1

B_{L2}

...

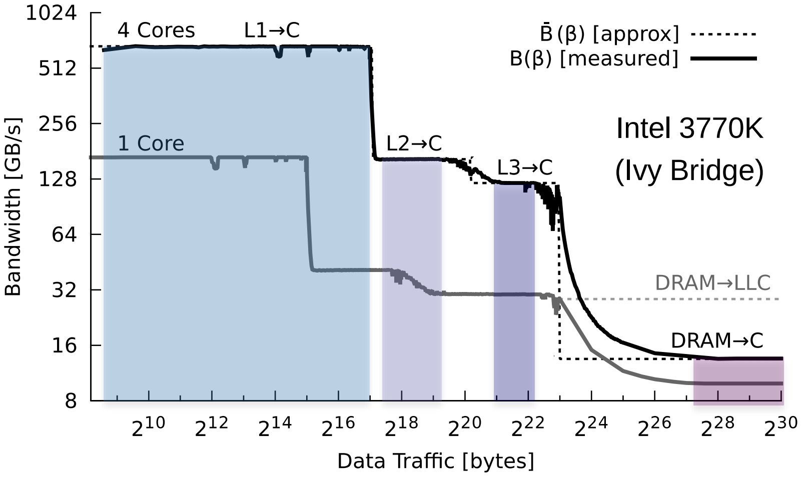
B_{LLC}

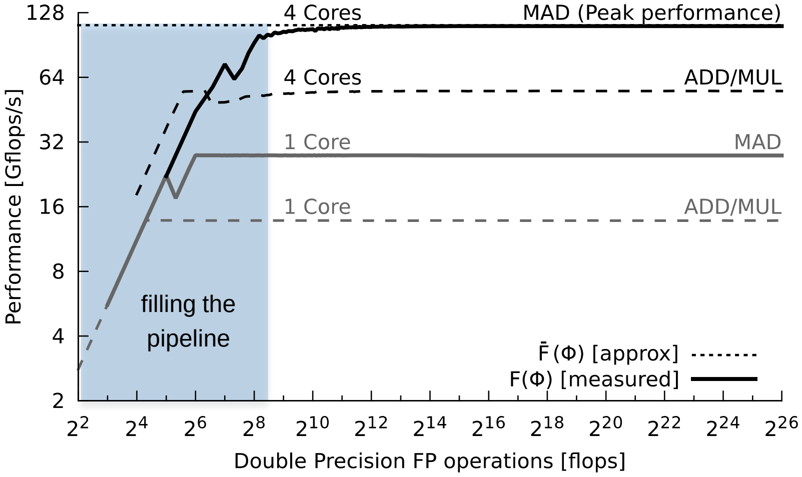
LLC

B_D

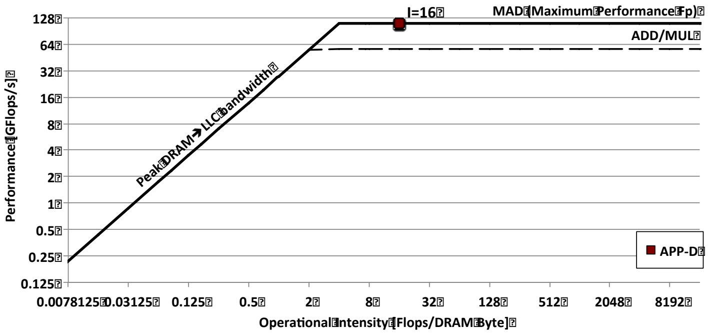
DRAM (off-chip)

- **Performance**: Computations (*flops*) and communication (*bytes*) overlap in time

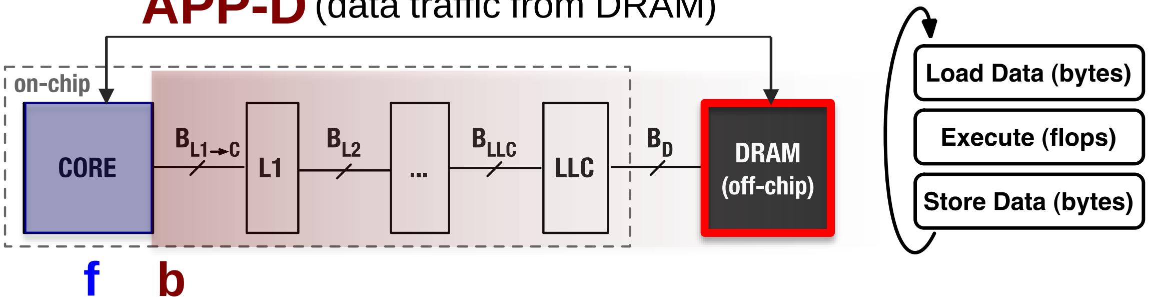


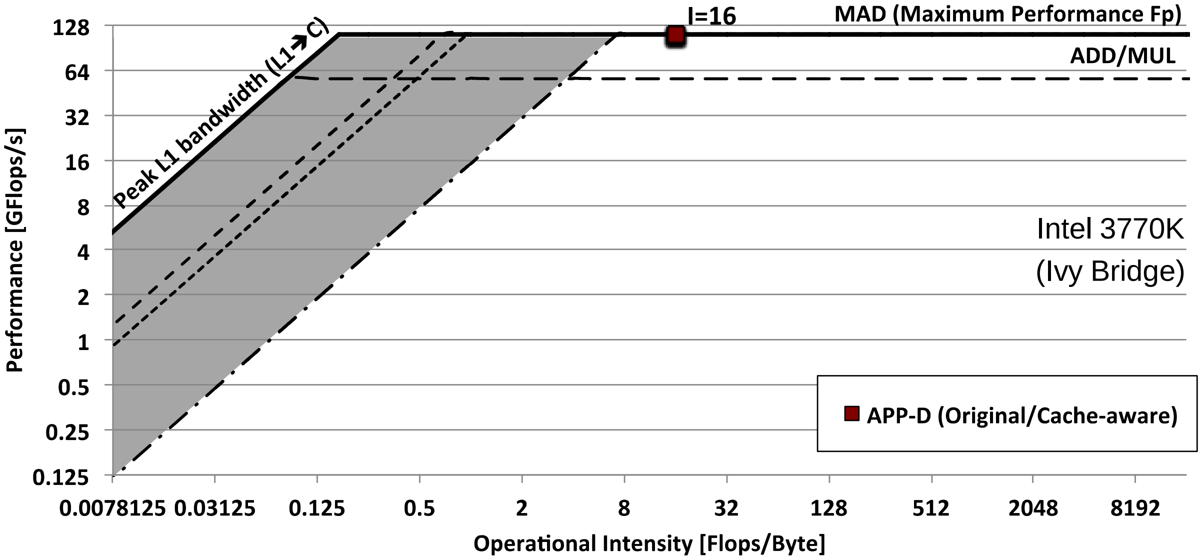


Cache-aware Roofline Model: Hands On



APP-D (data traffic from DRAM)



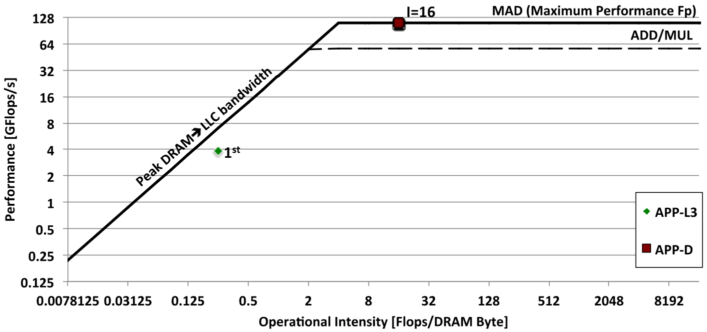


$$I=(\Sigma f_l)/(\Sigma b_l)$$

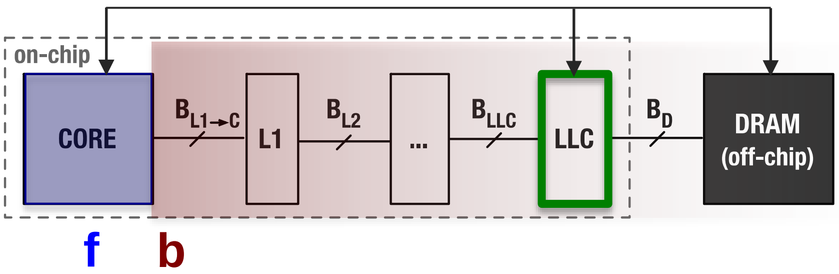
I is constant

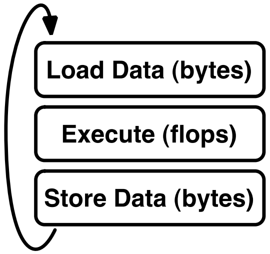
* *Ilic, A., Pratas, F. and Sousa, L., “Cache-aware Roofline Model: Upgrading the Loft”, IEEE Computer Architecture Letters, 2013*

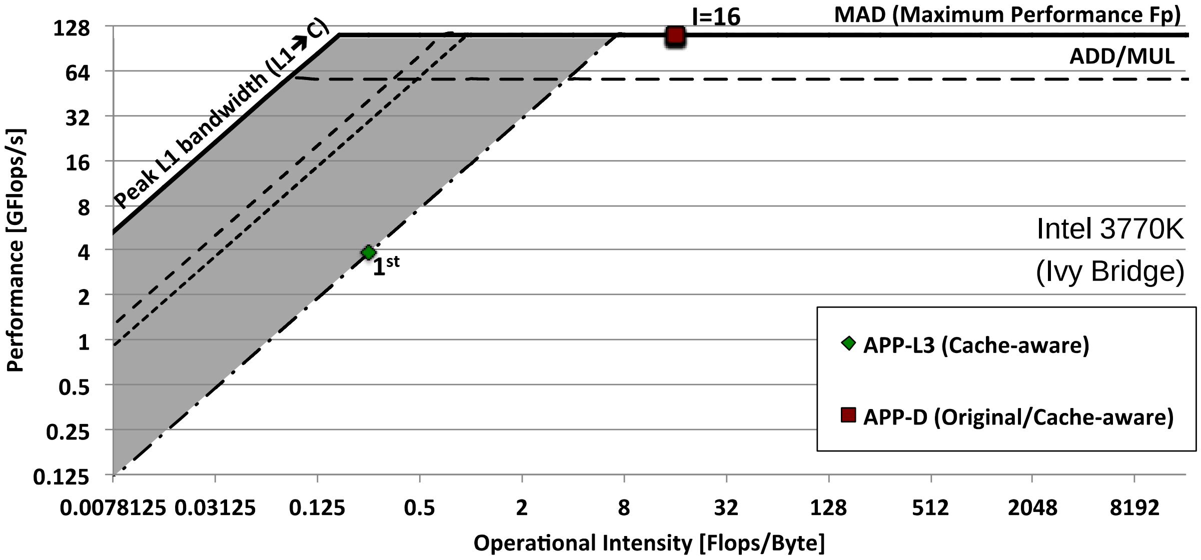
Cache-aware Roofline Model: Hands On



APP-L3 (fits in L3)





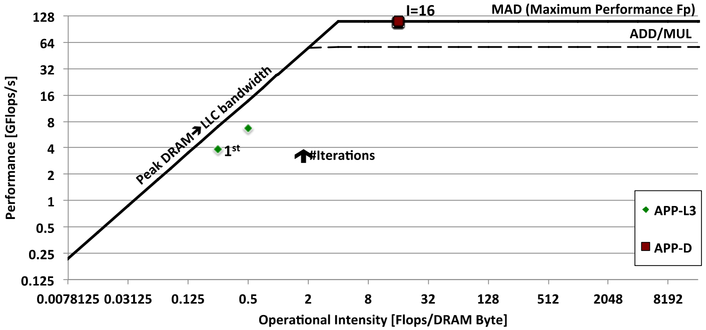


$$I=(\Sigma f_l)/(\Sigma b_l)$$

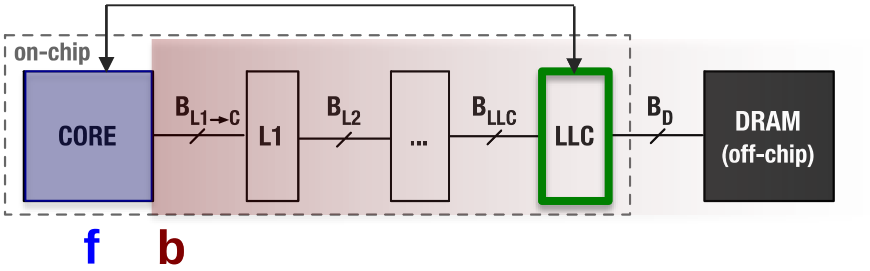
I is constant

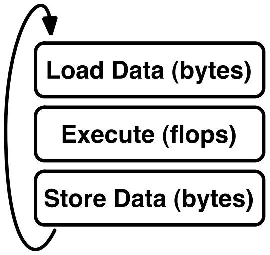
* *Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

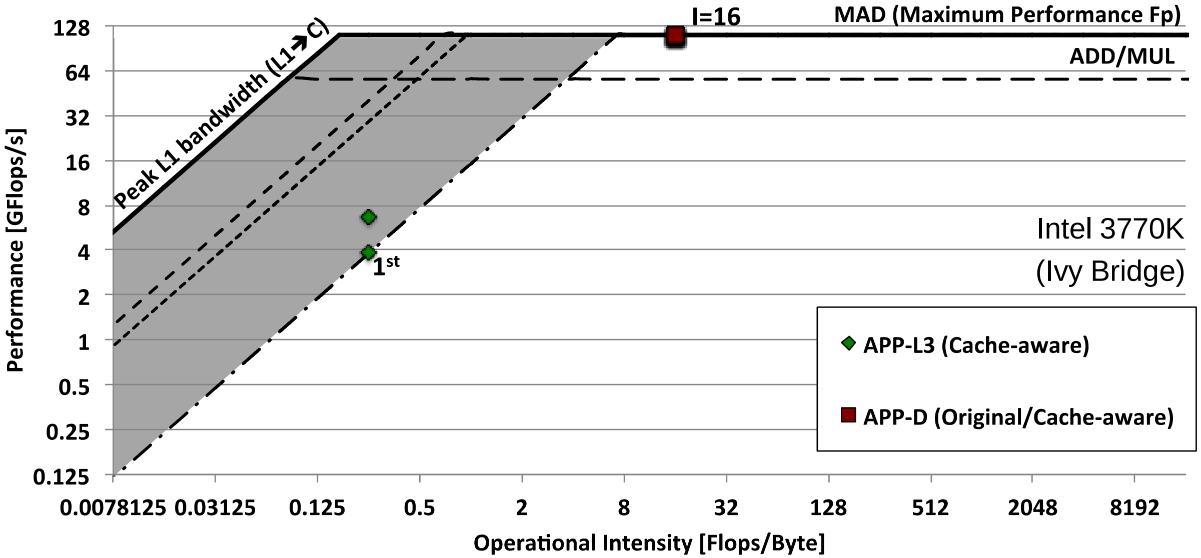
Cache-aware Roofline Model: Hands On



APP-L3 (fits in L3)





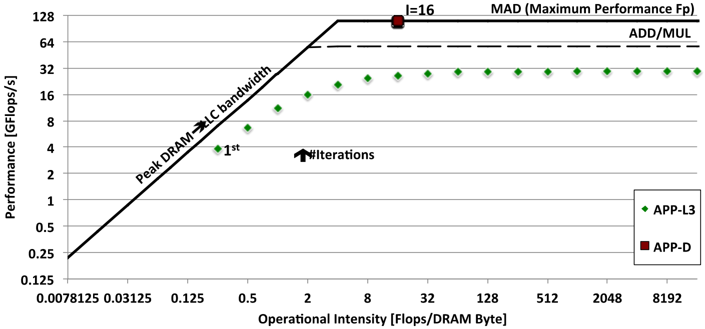


$$I=(\Sigma f_l)/(\Sigma b_l)$$

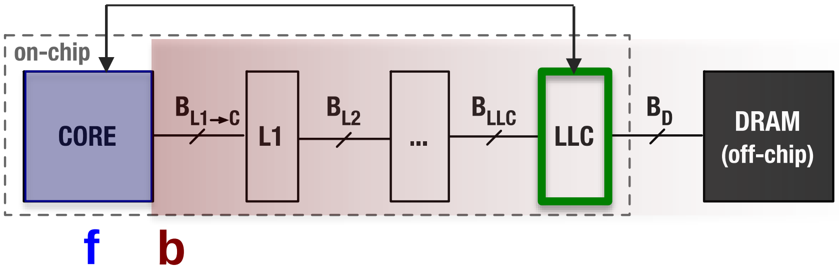
I is constant

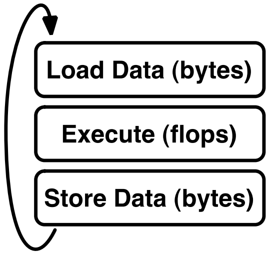
* *Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

Cache-aware Roofline Model: Hands On



APP-L3 (fits in L3)

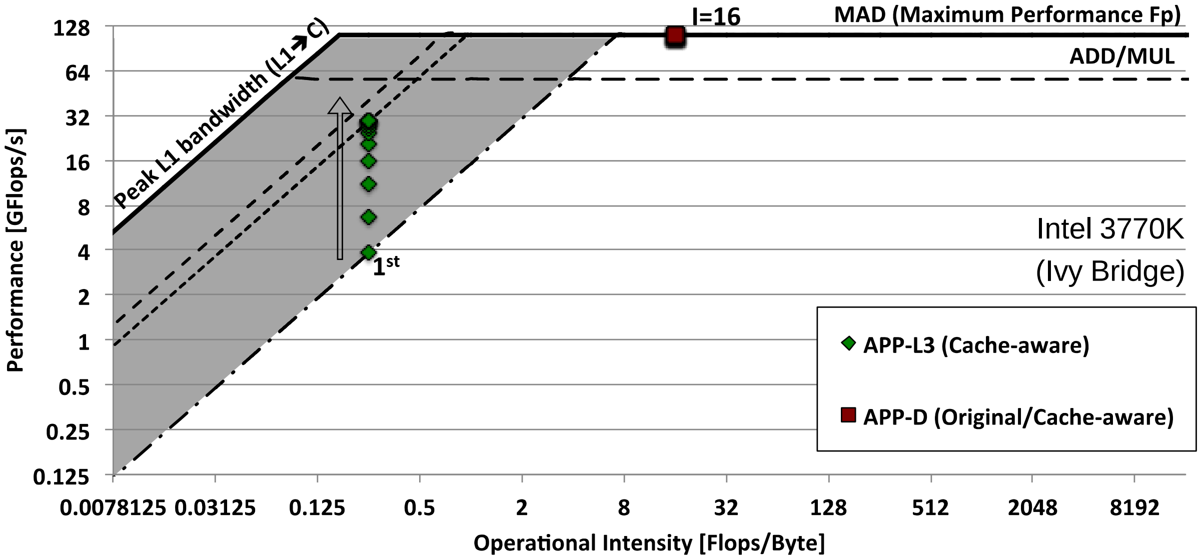




Load Data (bytes)

Execute (flops)

Store Data (bytes)

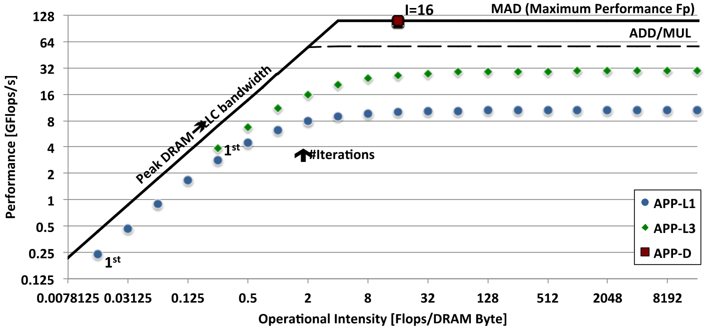


$$I=(\Sigma f_l)/(\Sigma b_l)$$

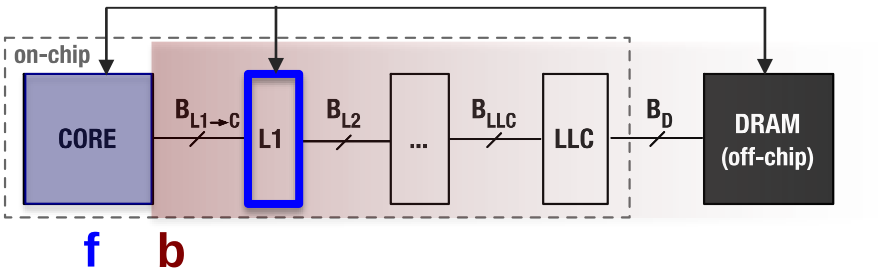
I is constant

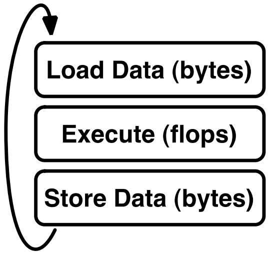
* *Ilic, A., Pratas, F. and Sousa, L., “Cache-aware Roofline Model: Upgrading the Loft”, IEEE Computer Architecture Letters, 2013*

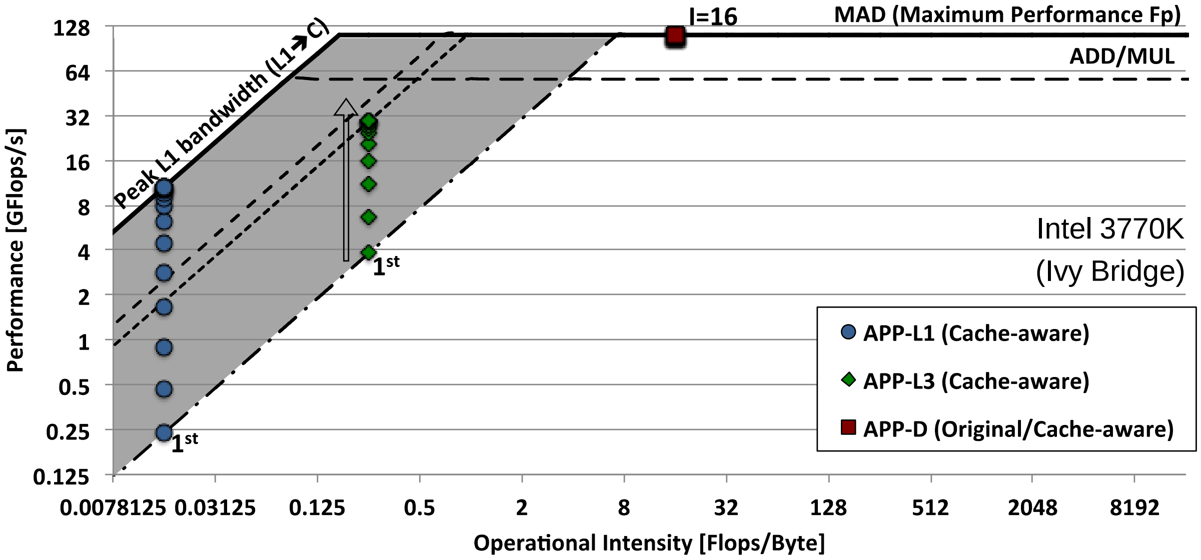
Cache-aware Roofline Model: Hands On



APP-L1 (fits in L1)





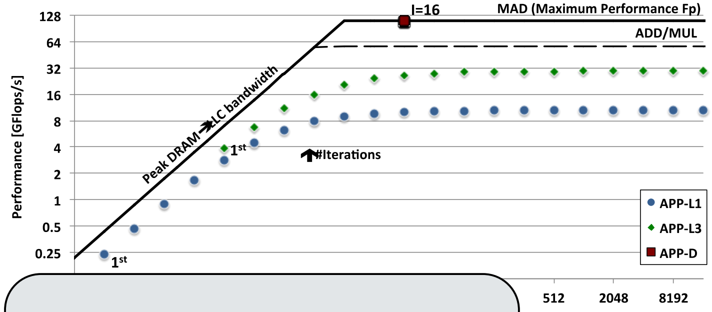


$$I=(\Sigma f_l)/(\Sigma b_l)$$

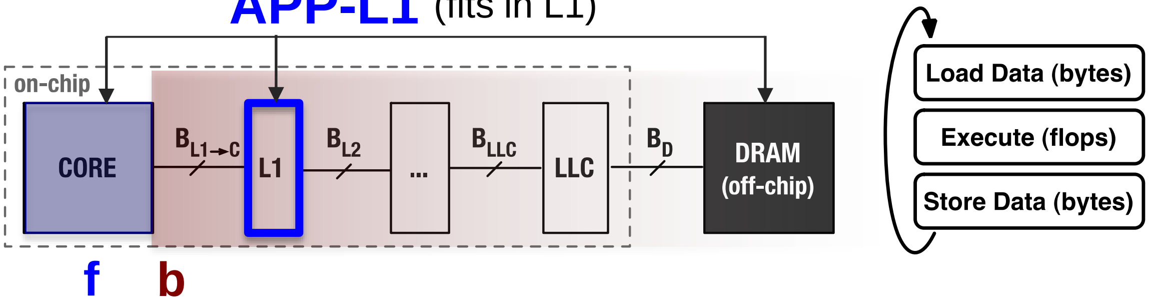
I is constant

* *Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

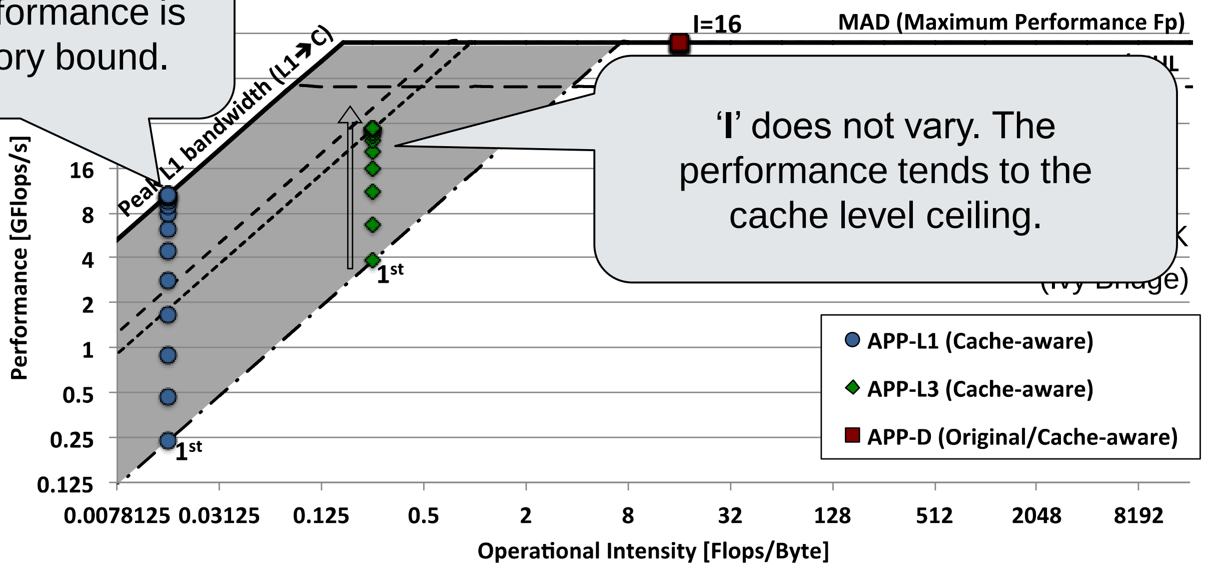
Cache-aware Roofline Model: Hands On



APP-L1 (fits in L1)



Achieves maximum attainable performance is always memory bound.



'I' does not vary. The performance tends to the cache level ceiling.

$I=(\Sigma f_I)/(\Sigma b_I)$

I is constant

* Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013

Cache-Aware Roofline Model vs. State-of-the-Art

WHAT IS HOT (WHAT IS NO

APPLICATION CHARA

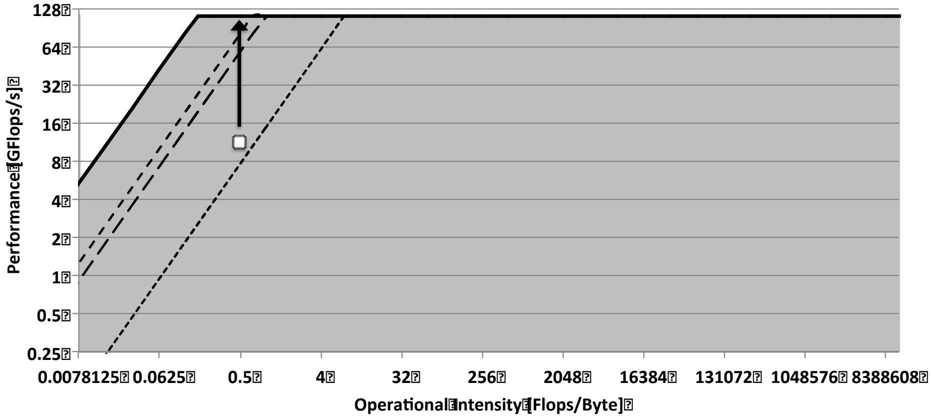
C

Hands-on:

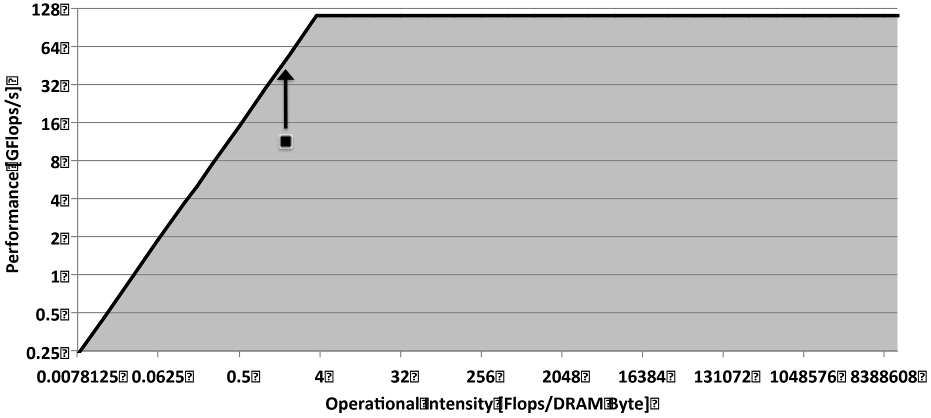
Application Optimization

Practical Example: Dense Matrix Multiplication

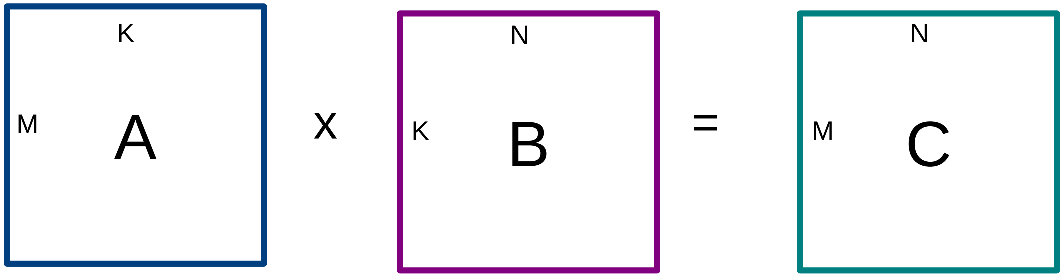
Cache-aware Roofline Model



Original Roofline Model



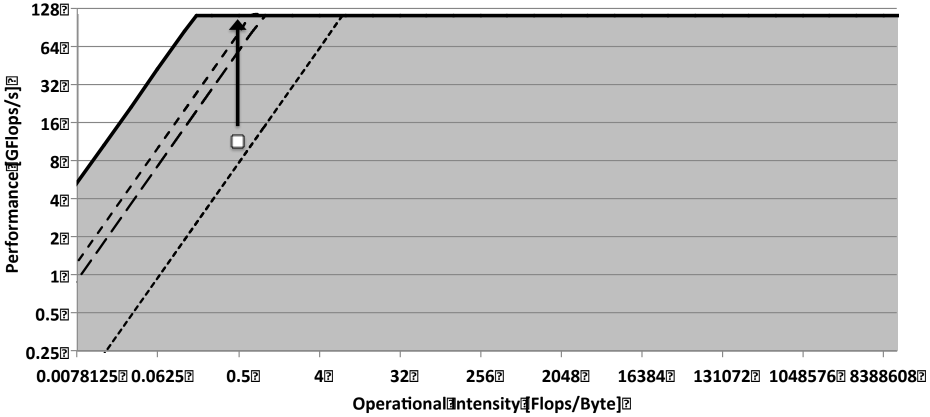
```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```



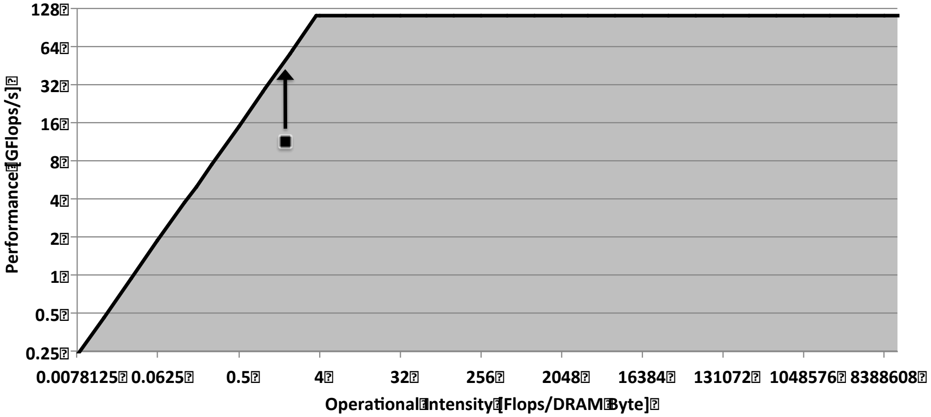
1) Basic implementation: All matrices stored in row-major order.

Practical Example: Dense Matrix Multiplication

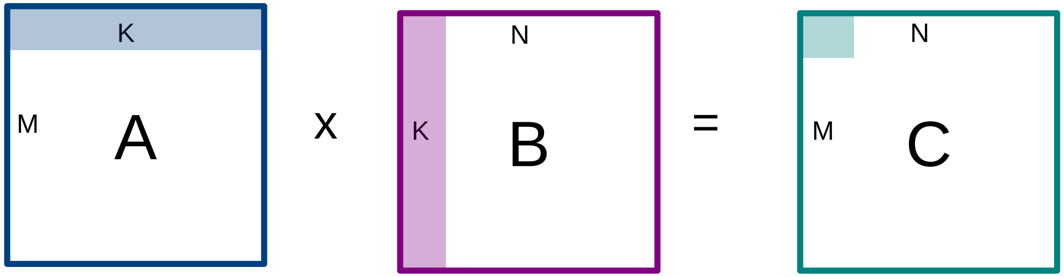
Cache-aware Roofline Model



Original Roofline Model



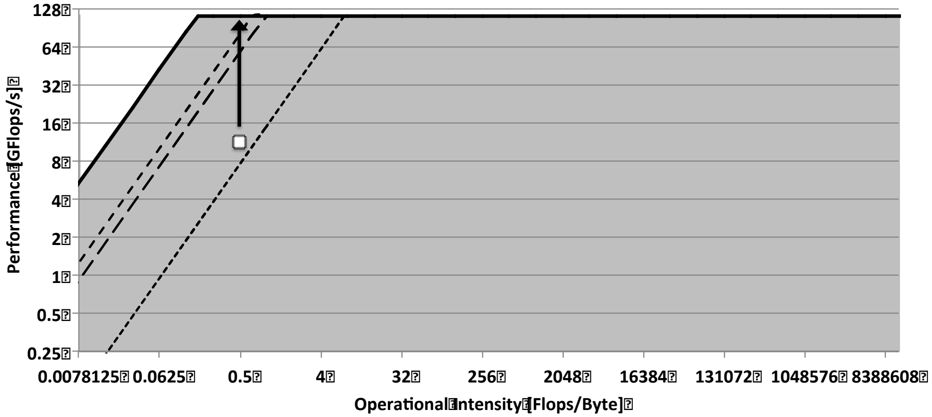
```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```



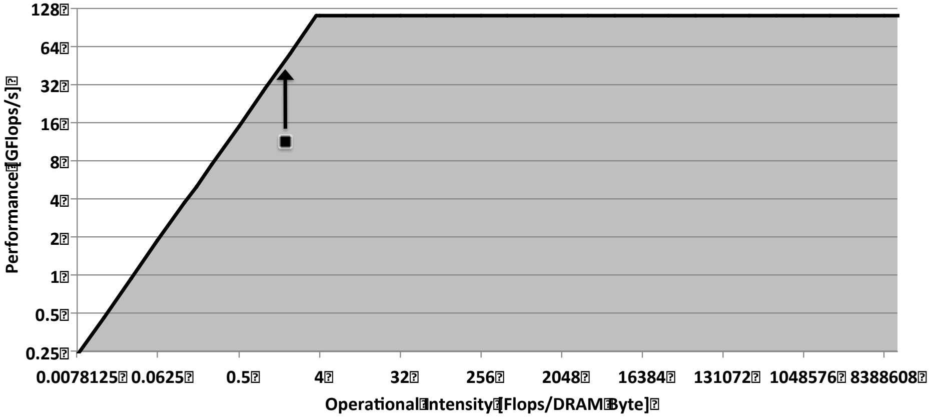
1) Basic implementation: All matrices stored in row-major order.

Practical Example: Dense Matrix Multiplication

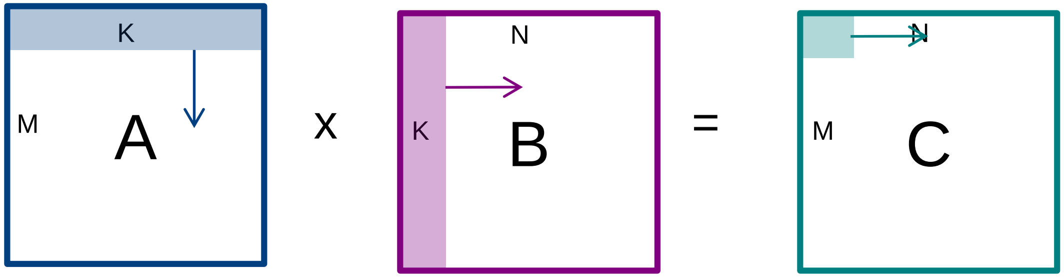
Cache-aware Roofline Model



Original Roofline Model



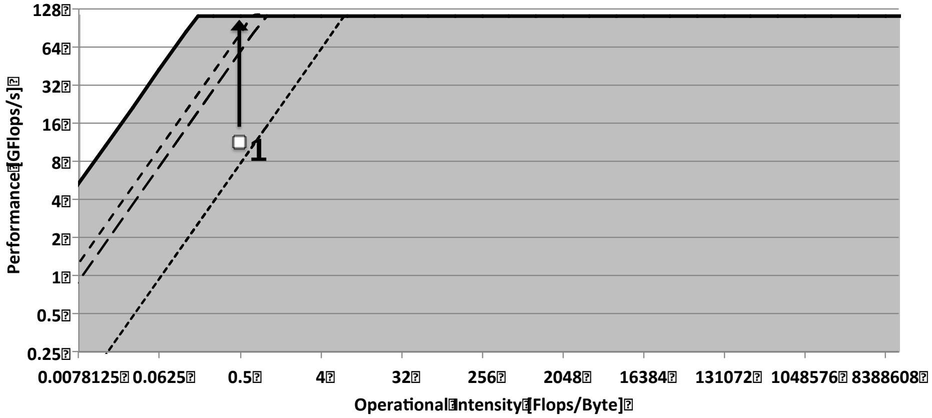
```
// matrix multiplication example
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```



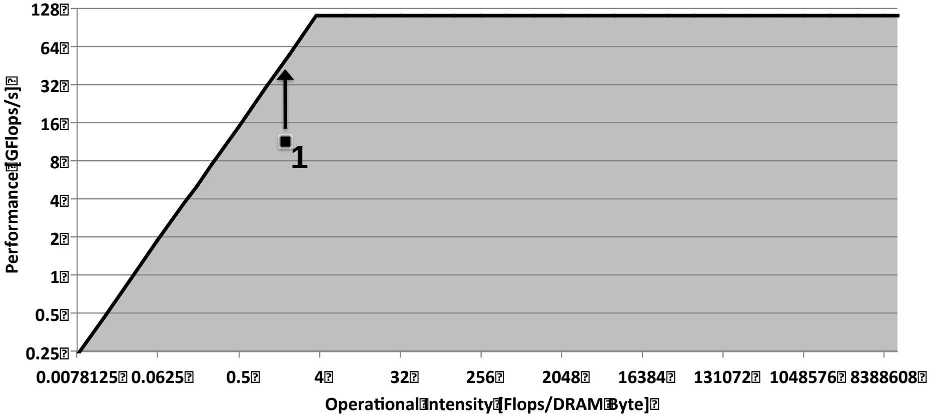
1) Basic implementation: All matrices stored in row-major order.

Practical Example: Dense Matrix Multiplication

Cache-aware Roofline Model



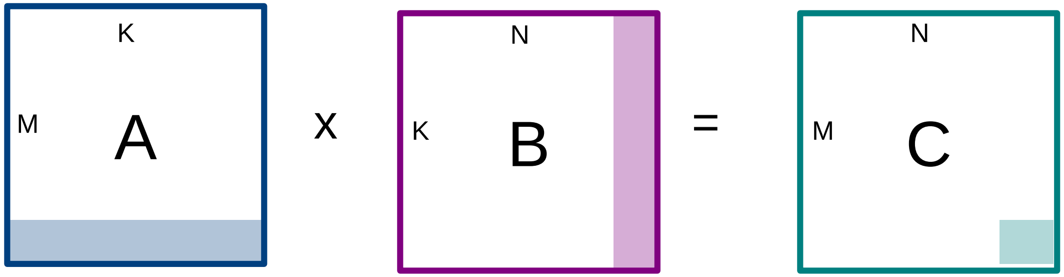
Original Roofline Model



- app in the compute bound region
- mainly limited by DRAM
- can be optimized to hit higher cache levels

- app in the memory bound region
- mainly limited by DRAM
- can be optimized up to the slanted part

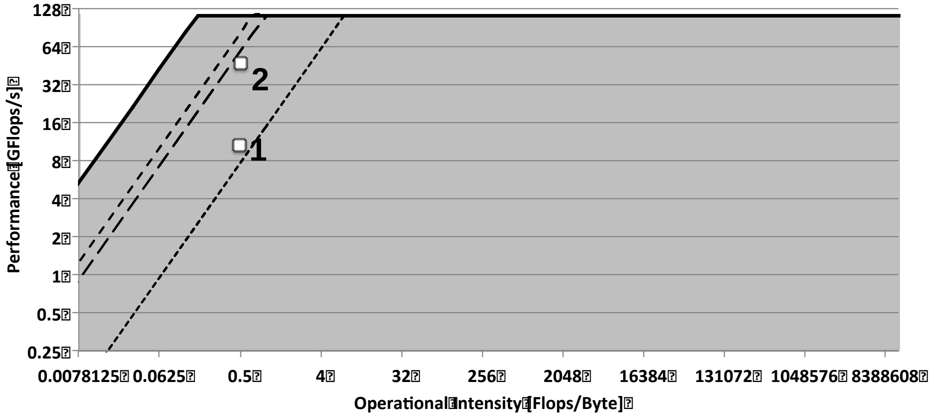
```
// matrix multiplication example
// VER 1: Row major matrices
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```



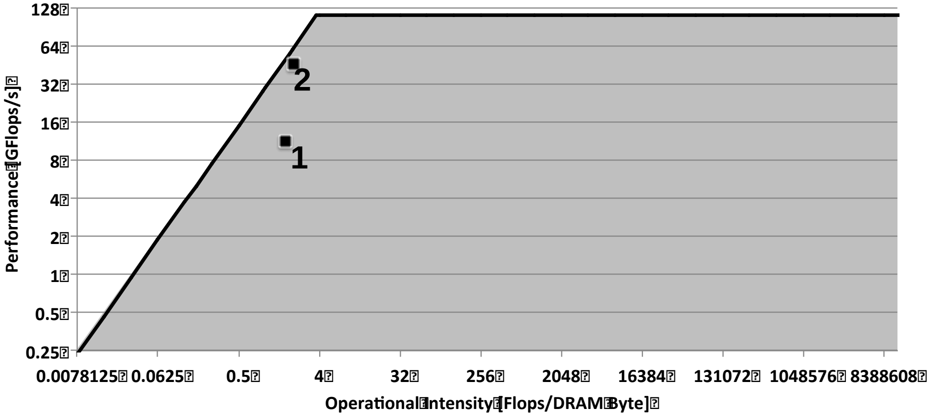
1) Basic implementation: All matrices stored in row-major order.

Practical Example: Dense Matrix Multiplication

Cache-aware Roofline Model



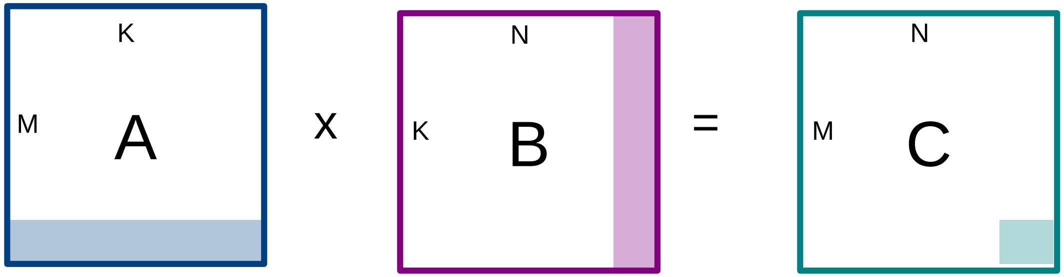
Original Roofline Model



- app in the compute bound region
- almost hits L3
- **can be further optimized** to hit higher cache levels

- app in the memory bound region
- performance *hits the roof* of the model
- suggests that the **optimization is finished**

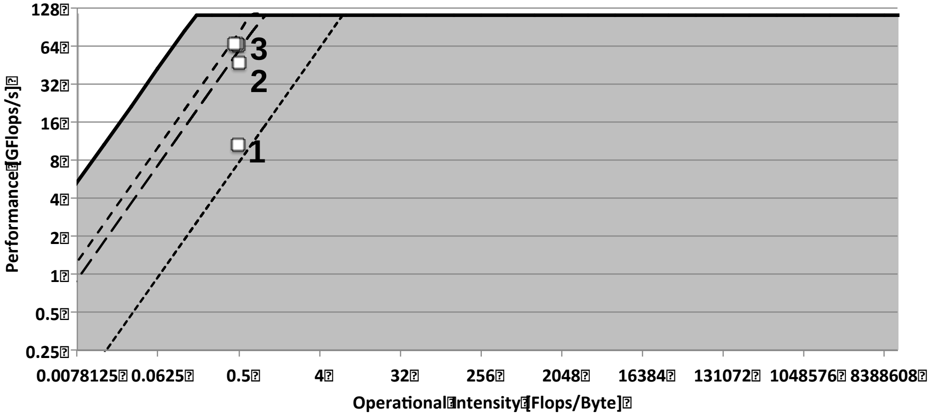
```
// matrix multiplication example
// VER 1: Row major matrices
// OPT 2: Transpose B matrix
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```



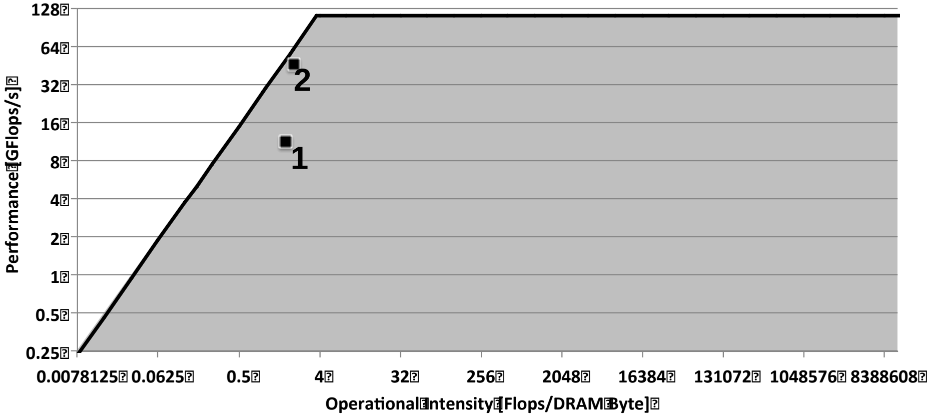
2) Transposition: One matrix is transposed into column-major

Practical Example: Dense Matrix Multiplication

Cache-aware Roofline Model



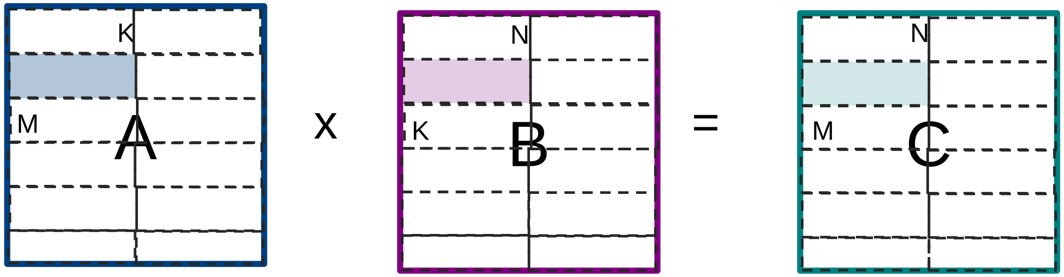
Original Roofline Model



- performance is further improved
- breaking the cache level ceilings **towards the roof**

- optimization process finished

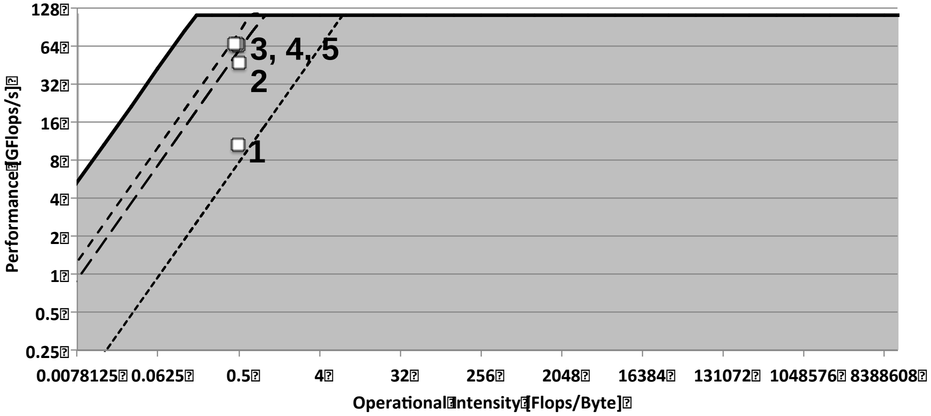
```
// matrix multiplication example
// OPT 3: Blocking for L3
for i=1 to M
   for j=1 to N
      for k=1 to K
         C[i,j] += A[i,k]*B[k,j]
```



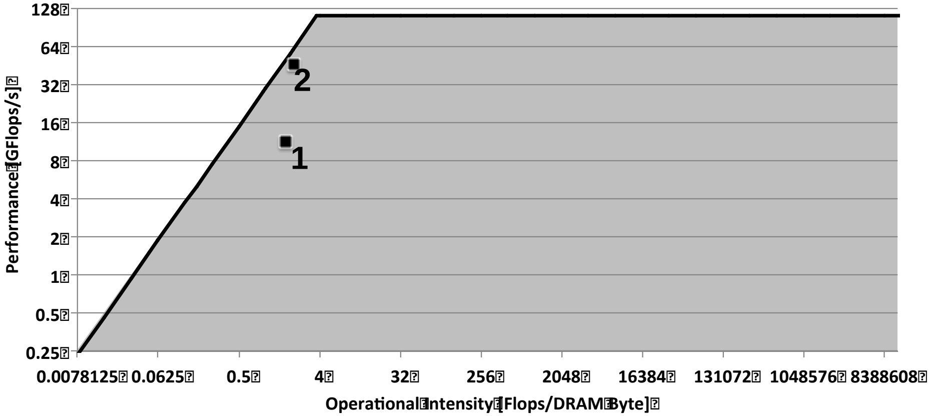
3) Blocking for L3: All matrices are blocked to efficiently exploit L3

Practical Example: Dense Matrix Multiplication

Cache-aware Roofline Model



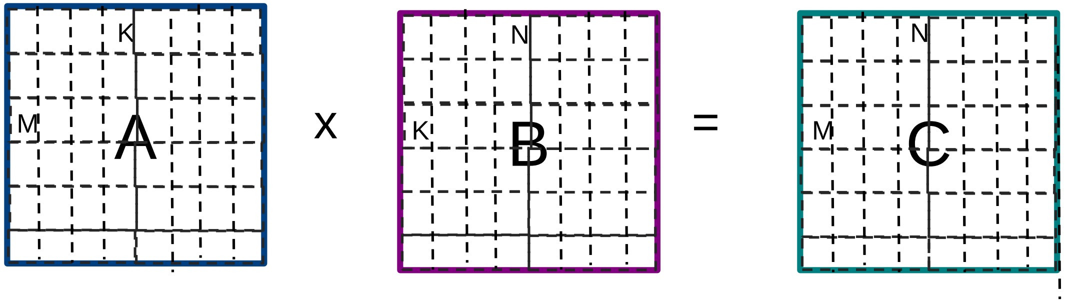
Original Roofline Model



- performance is further improved
- breaking the cache level ceilings **towards the roof**

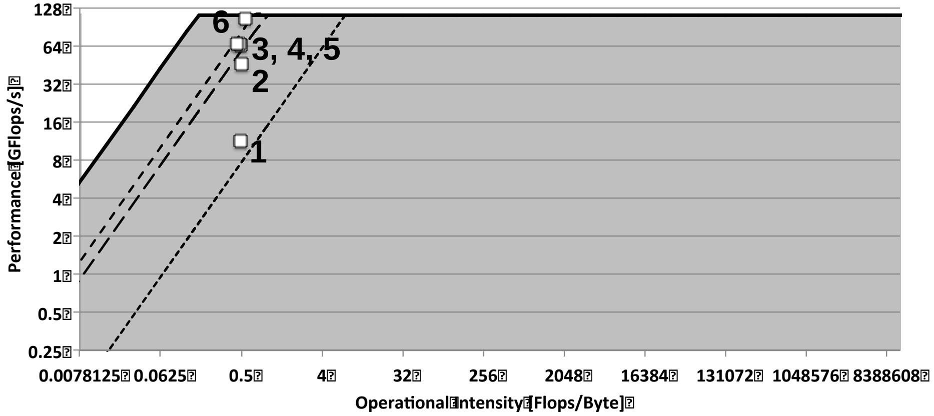
- **optimization process finished**

```
// matrix multiplication example
// VER 1: Row major matrices
// OPT 2: Transpose B matrix
// OPT 3: Blocking for L3
// OPT 4: Blocking for L2
// OPT 5: Blocking for L1
```



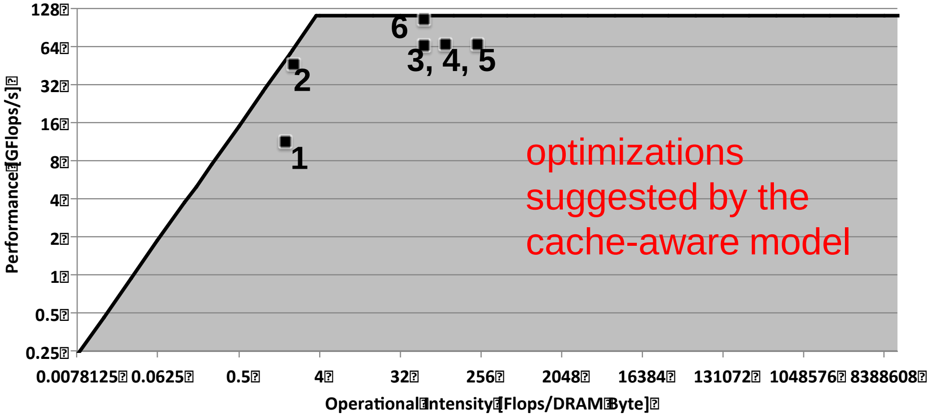
Practical Example: Dense Matrix Multiplication

Cache-aware Roofline Model



- OPT 6 achieves **near theoretical performance**

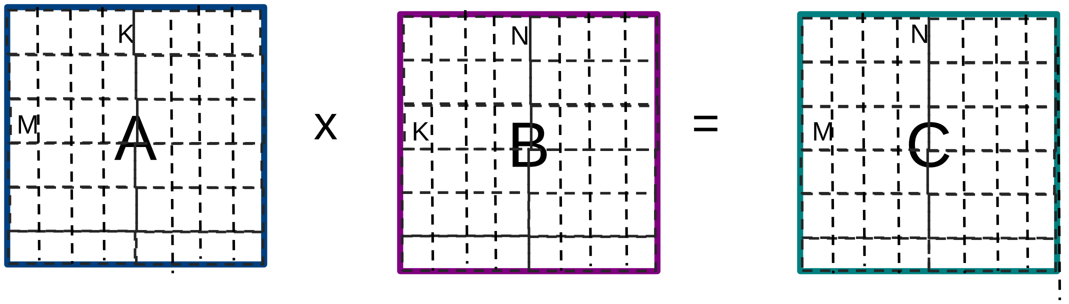
Original Roofline Model



optimizations suggested by the cache-aware model

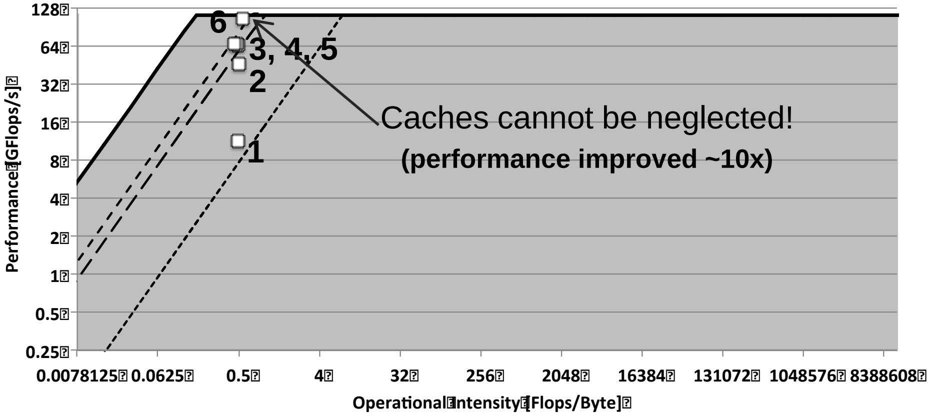
- moves to the compute bound region (shift in operational intensity)

```
// matrix multiplication example
// VER 1: Row major matrices
// OPT 2: Transpose B matrix
// OPT 3: Blocking for L3
// OPT 4: Blocking for L2
// OPT 5: Blocking for L1
// OPT 6: Highly optimized (MKL)
```



Practical Example: Dense Matrix Multiplication

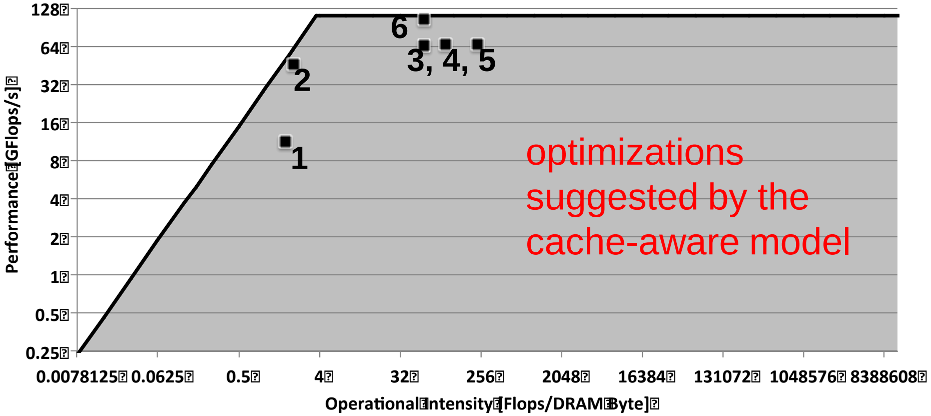
Cache-aware Roofline Model



Caches cannot be neglected!

(performance improved ~10x)

Original Roofline Model

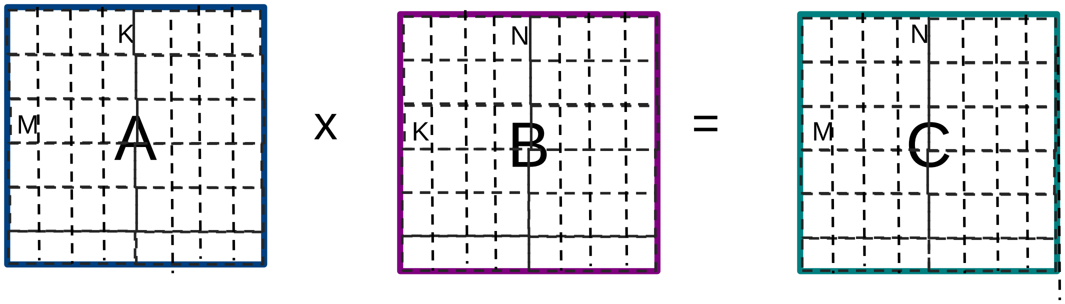


optimizations suggested by the cache-aware model

- OPT 6 achieves **near theoretical performance**

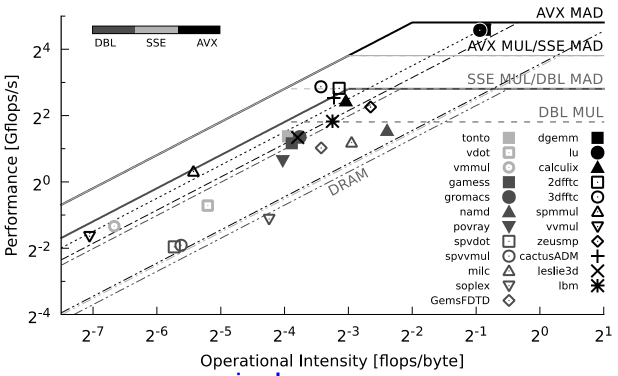
- moves to the compute bound region (shift in operational intensity)

```
// matrix multiplication example
// VER 1: Row major matrices
// OPT 2: Transpose B matrix
// OPT 3: Blocking for L3
// OPT 4: Blocking for L2
// OPT 5: Blocking for L1
// OPT 6: Highly optimized (MKL)
```

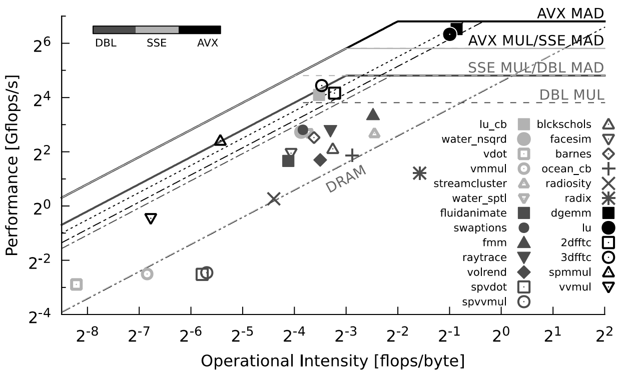


Cache-aware Roofline Model: Use Cases

Application Characterization

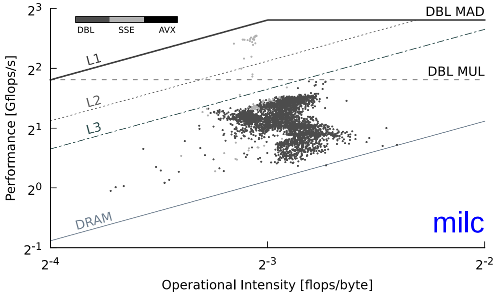


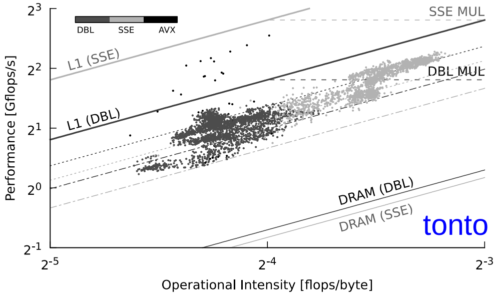
single core

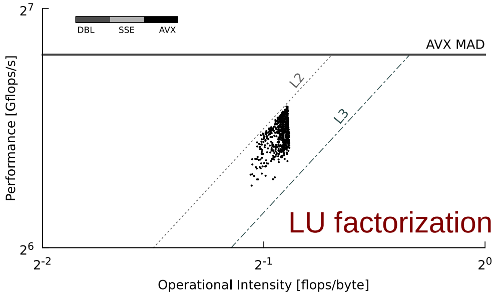


quad-core

Online Monitoring







* *Antão, D., Taniça, L., Ilić, A., Pratas, F., Tomás, P., and Sousa, L., "Monitoring Performance and Power for Application Characterization with Cache-aware Roofline Model", PPAM'13*

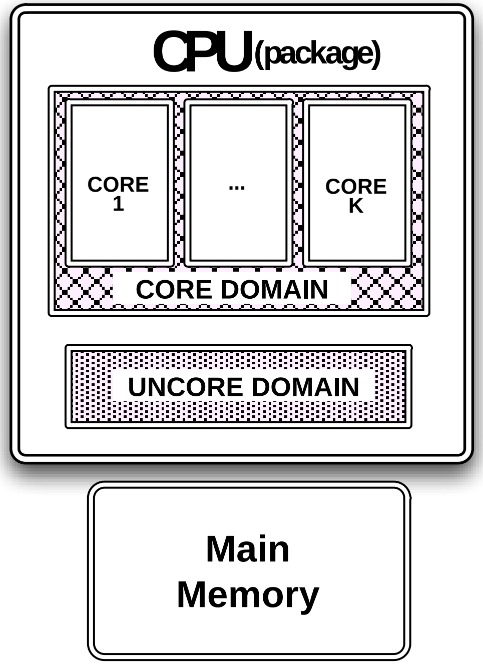
Outline

THE CACHE-AWARE ROOFLINE MODEL:

- PERFORMANCE
- POWER*
- EFFICIENCY

- *Ilić, A., Pratas, F. and Sousa, L., "Beyond the Roofline: Power, Energy and Efficiency Modeling for Multicores" (submitted)*

Power Roofline Model

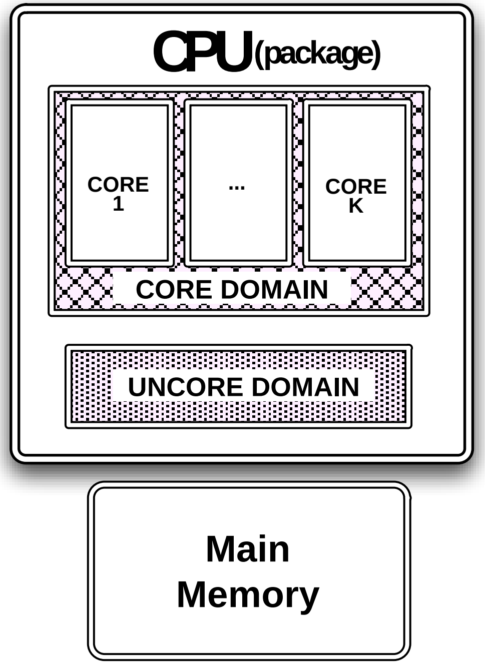


POWER MODELING FOR 3 DIFFERENT DOMAINS (RAPL-BASED):

1. POWER OF CORES (P_C)
   - consumed by components within the cores
2. UNCORE POWER (P_U)
   - consumed by all other (non-processing) parts of the chip, e.g., off-chip memory controller
3. PACKAGE POWER (P_P)
   - the power of the complete processor chip

Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



Power Roofline Model relates **Power Consumption** with Operational Intensity (I=f/b)

⇒ Average Power Consumption must be considered

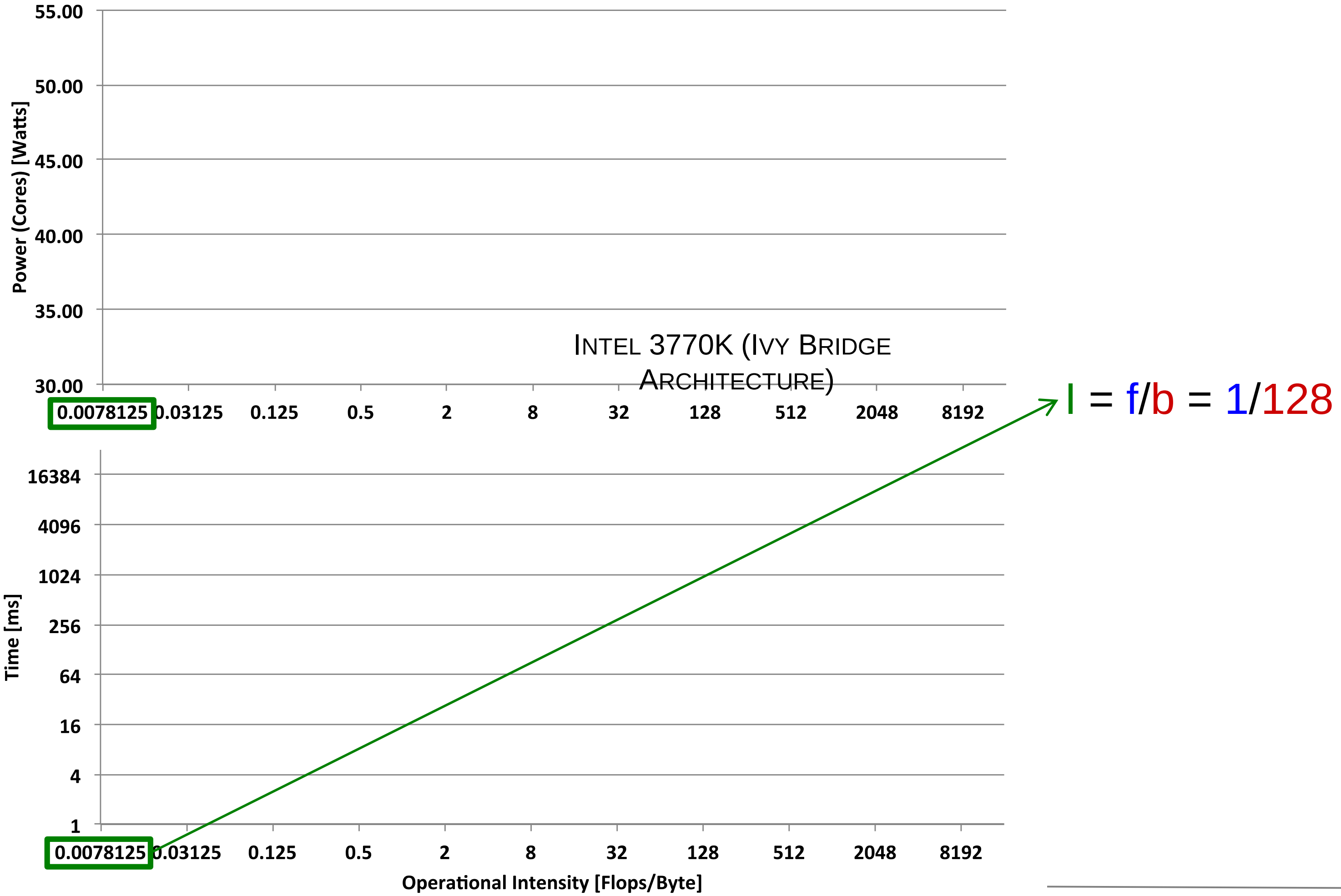
- during the *time interval, T(I)*, in which the (Roofline) performance is obtained!

⇒ Power Contributions of both memory operations and FP operations vary with two factors:

1. The number of executed operations
2. The contribution of each during the time interval **T(I)**

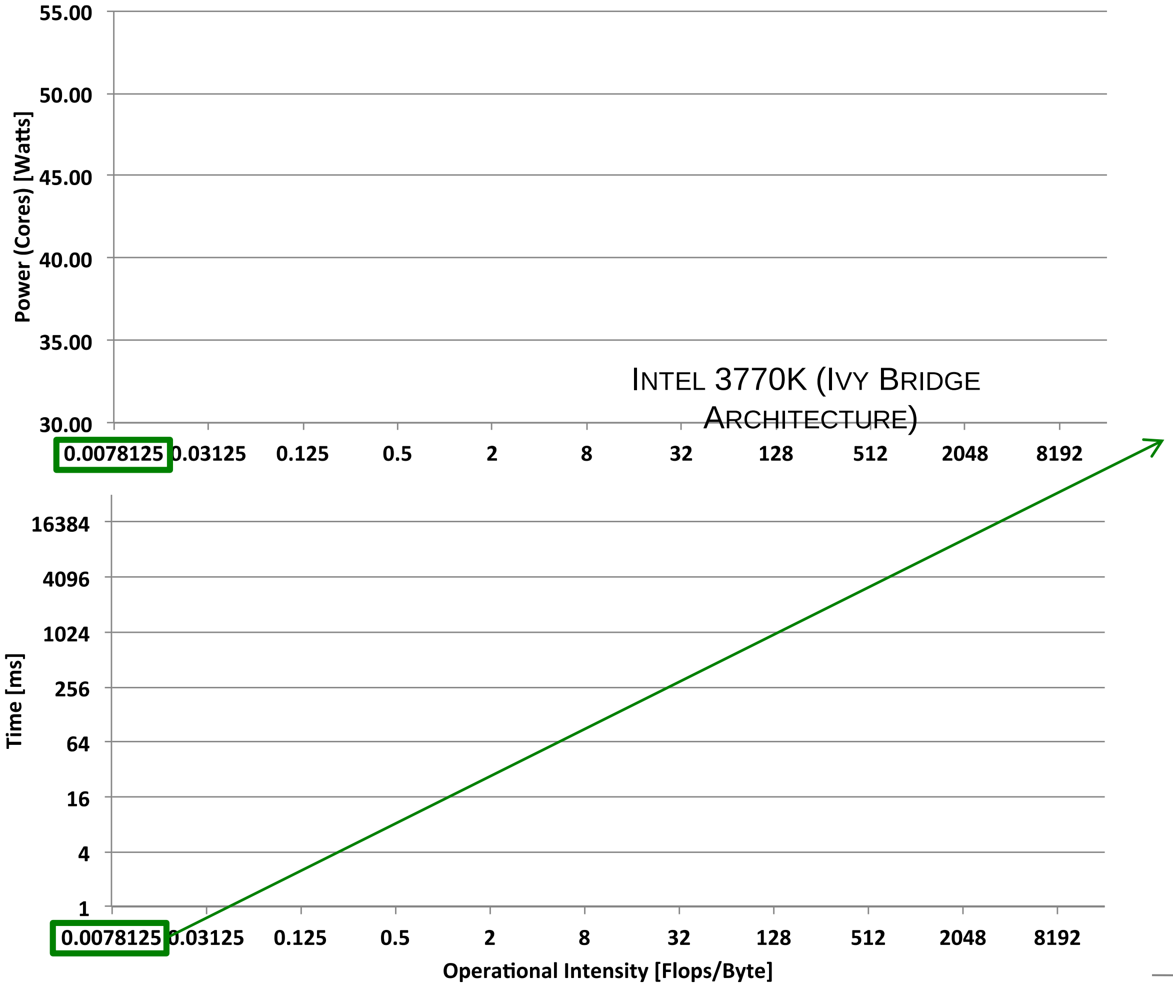
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



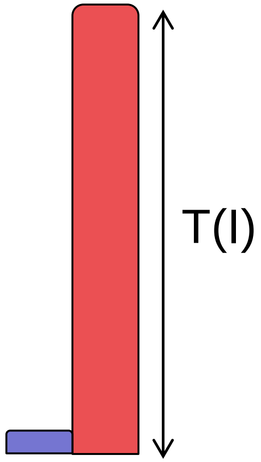
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



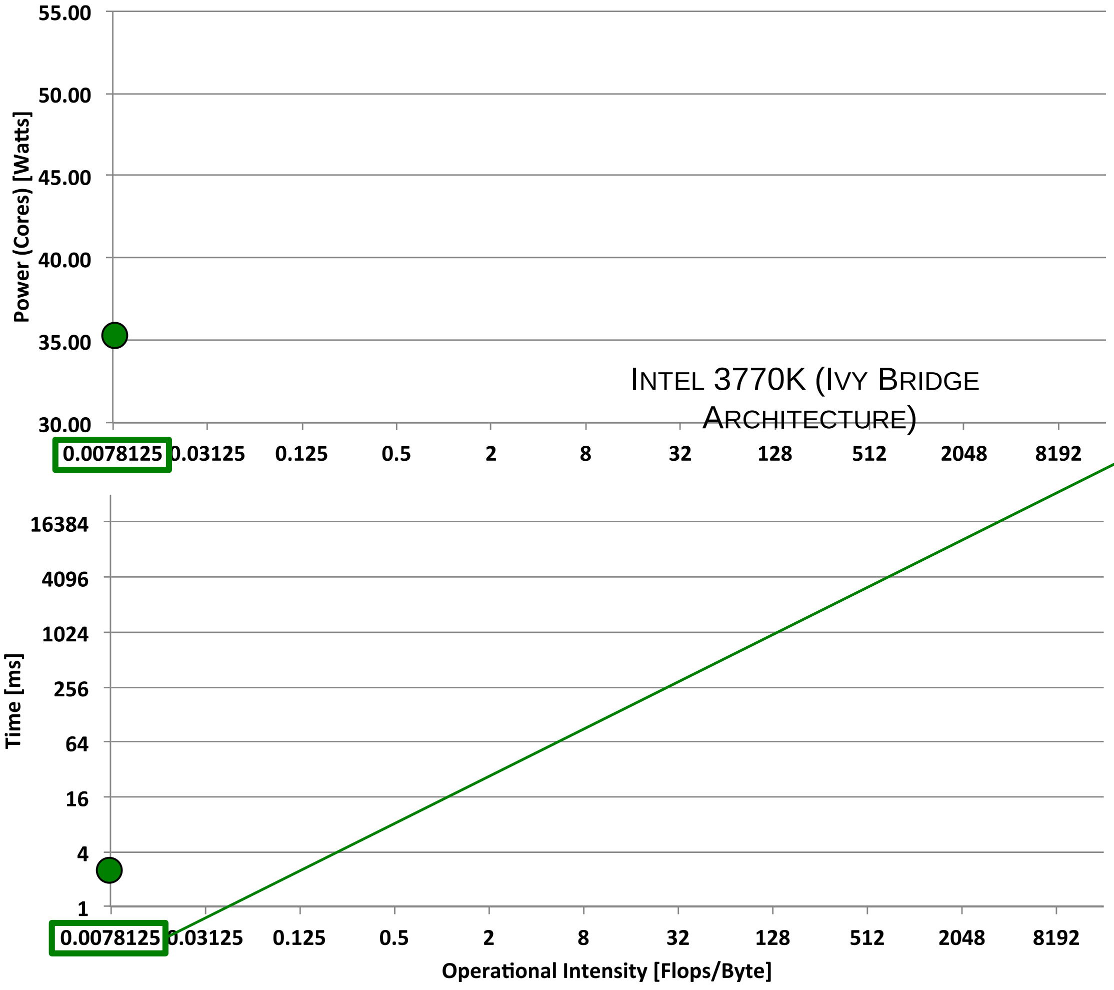
Intel 3770K (Ivy Bridge Architecture)

$I = f/b = 1/128$

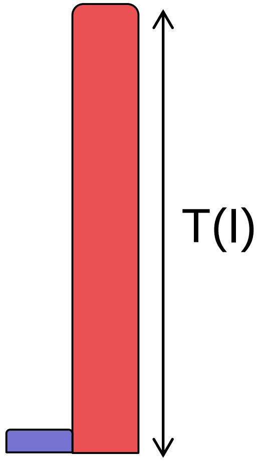


Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)

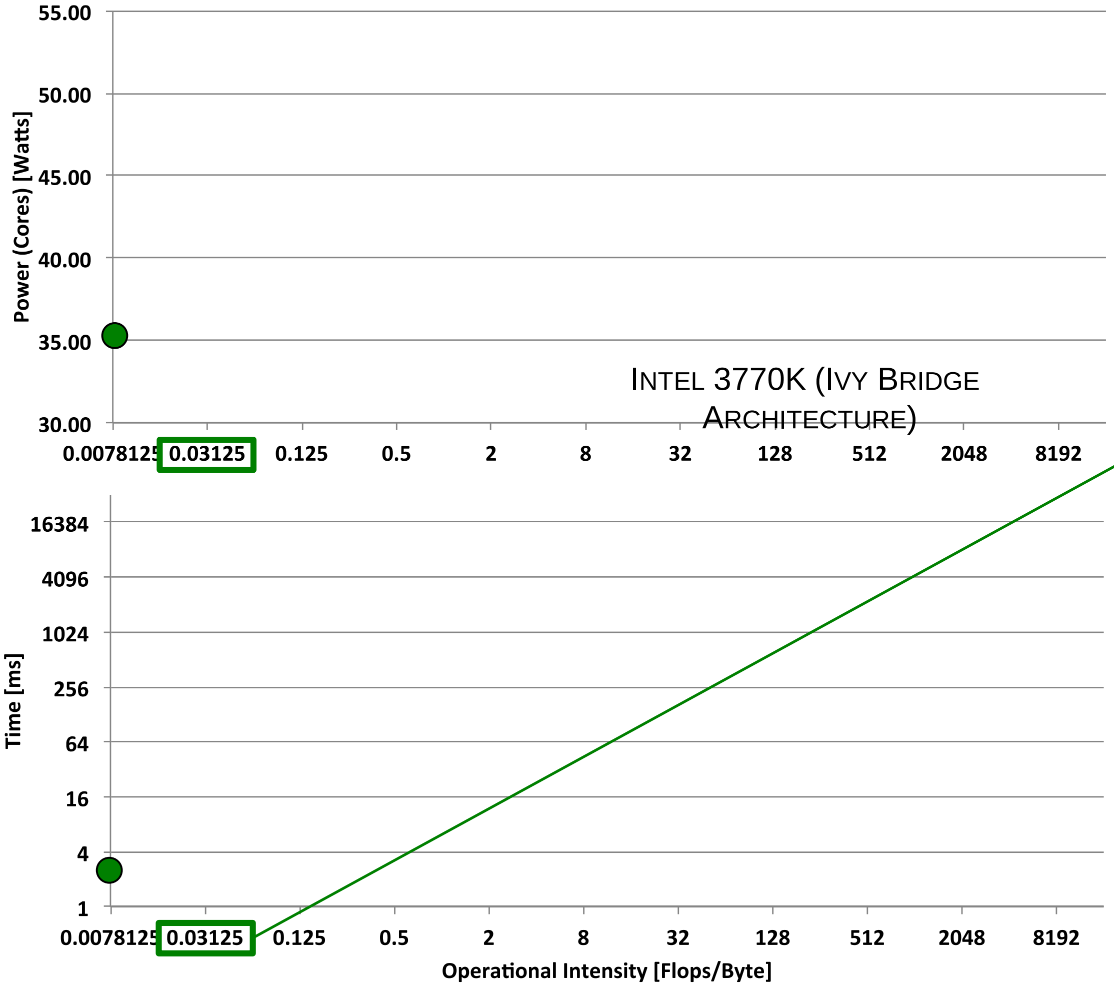


$I = f/b = 1/128$



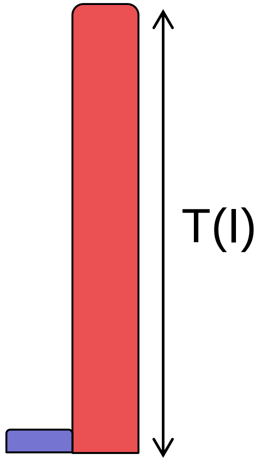
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



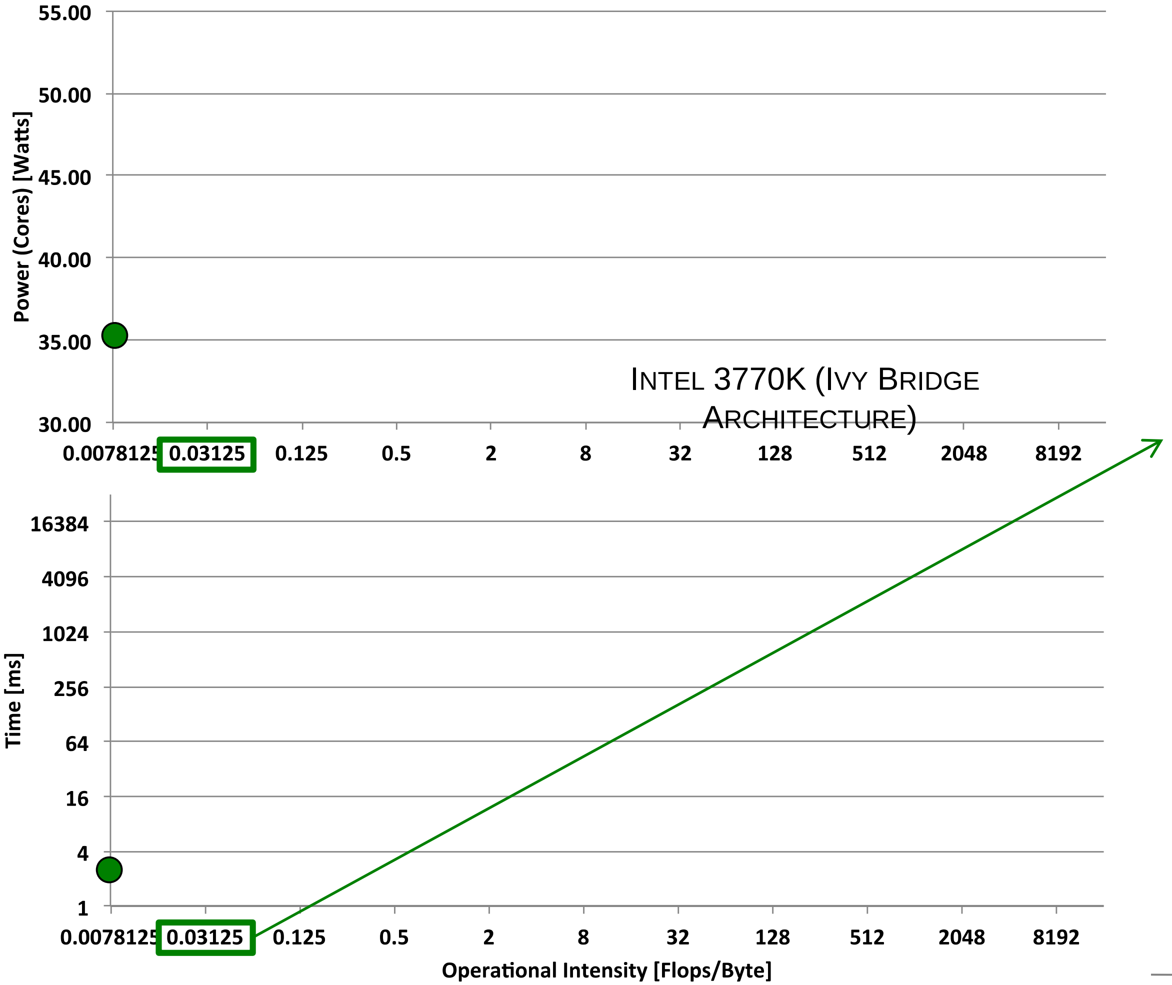
Intel 3770K (Ivy Bridge Architecture)

$I = f/b = 4/128$



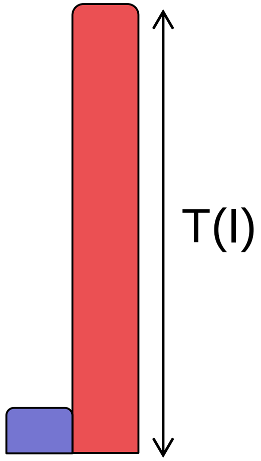
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



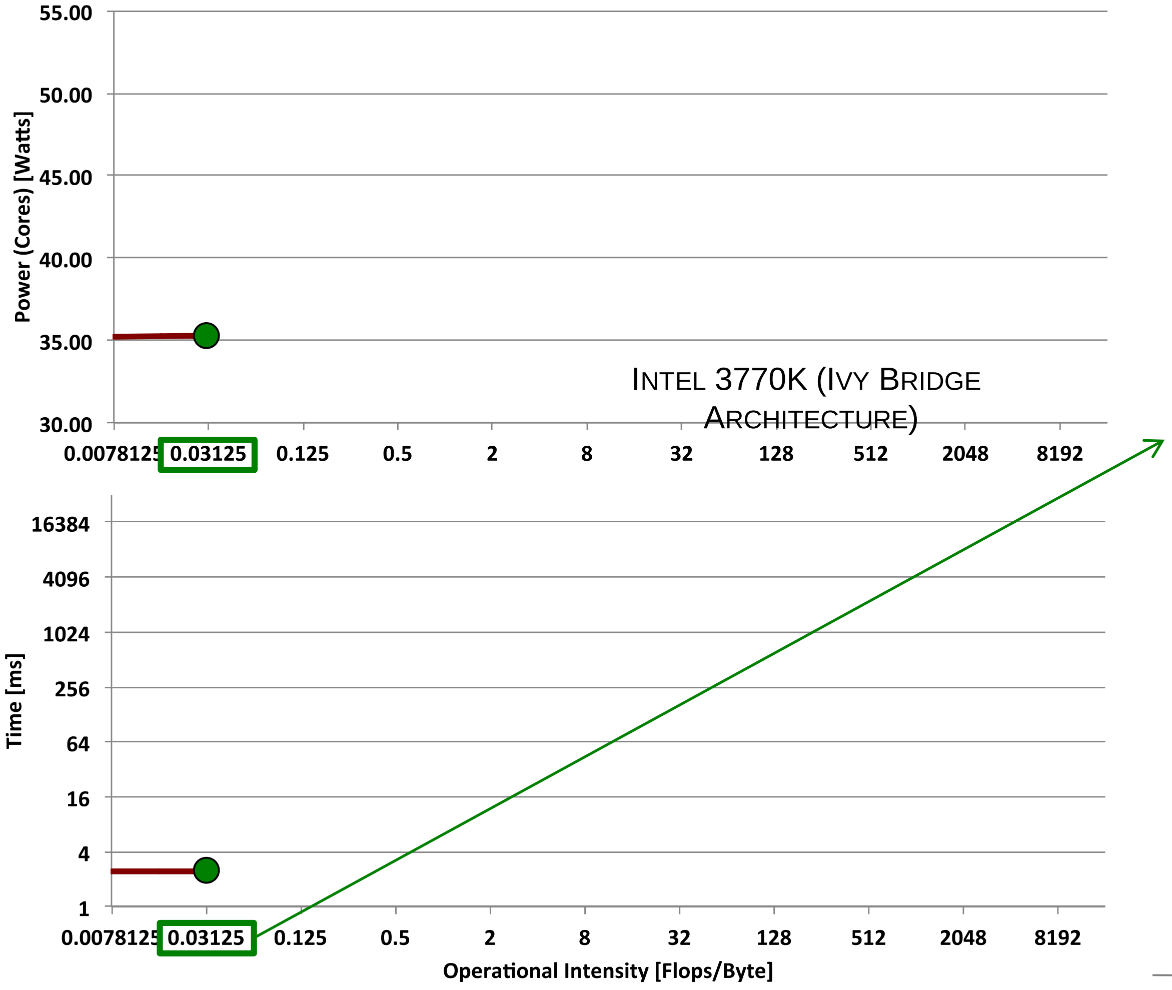
Intel 3770K (Ivy Bridge Architecture)

$I = f/b = 4/128$



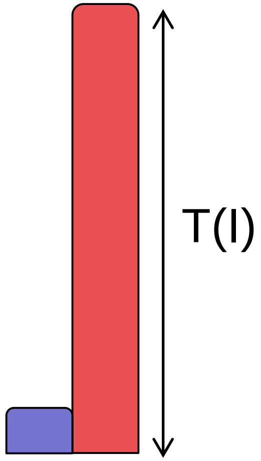
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



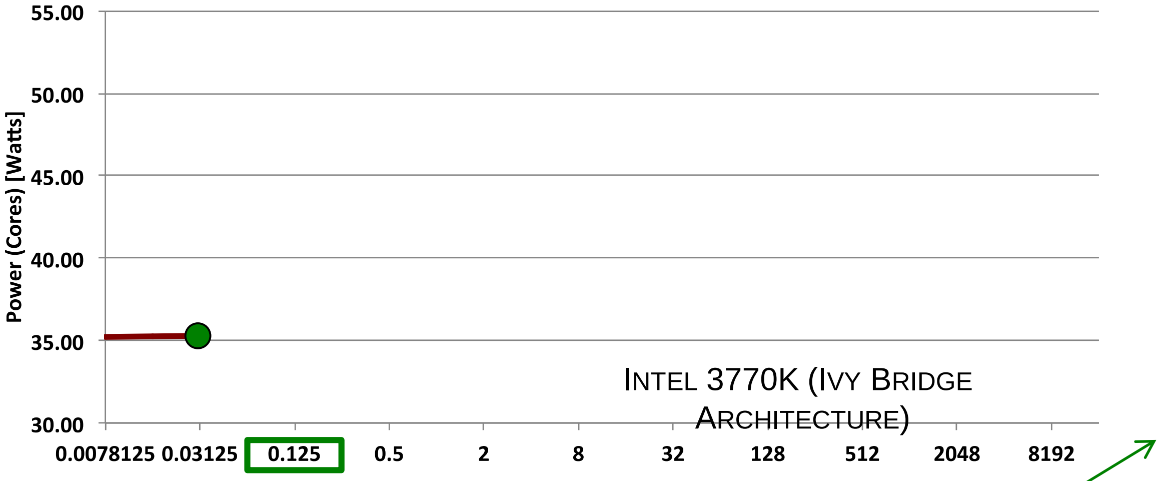
Intel 3770K (Ivy Bridge Architecture)

$I = f/b = 4/128$

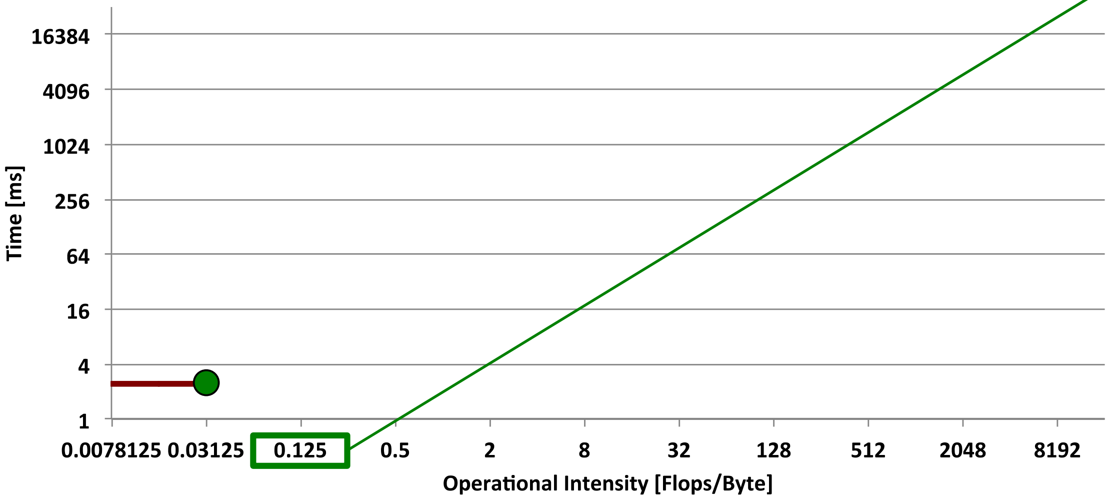


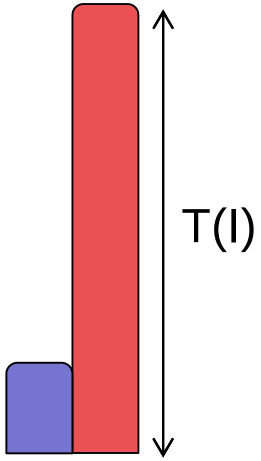
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



$I = f/b = 16/128$



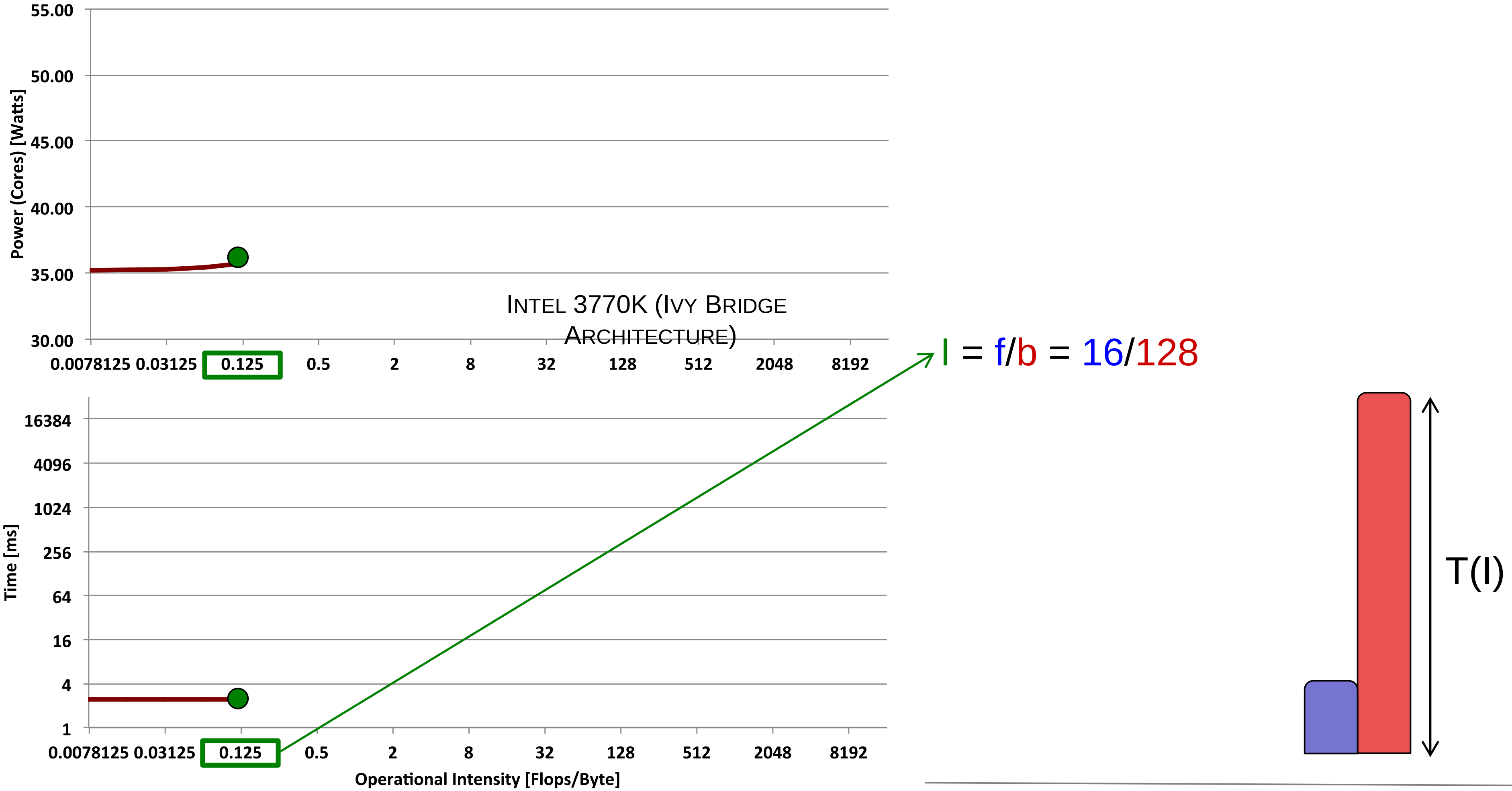


Power Roofline Model

1. Power of Cores (P_C)

2. Uncore Power (P_U)

3. Package Power (P_P)

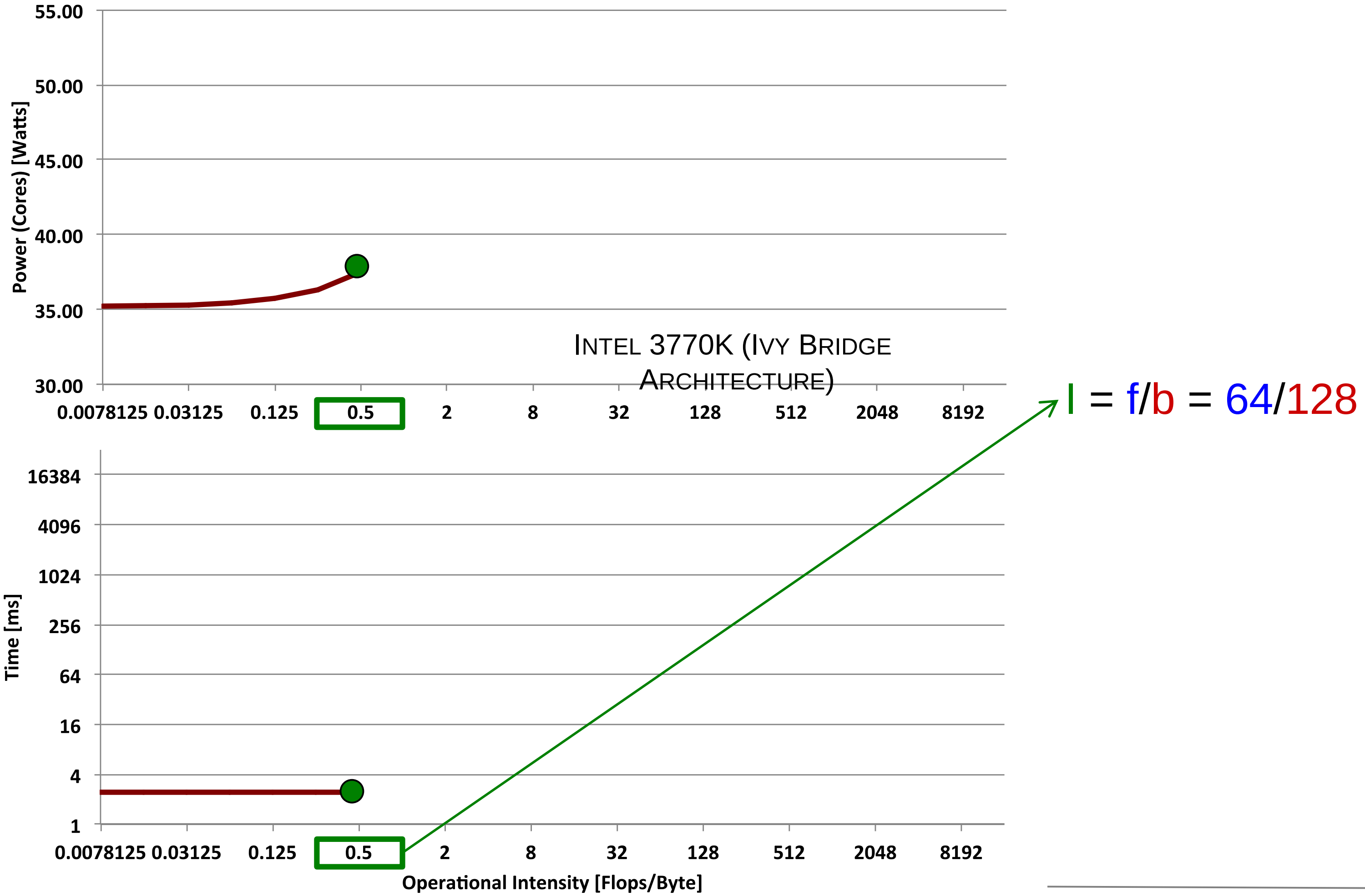


Intel 3770K (Ivy Bridge Architecture)

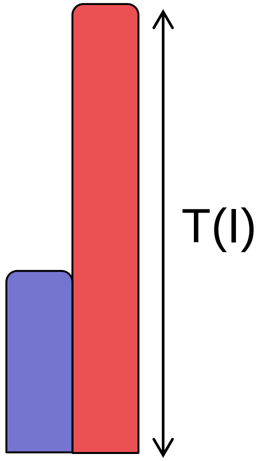
I = f/b = 16/128

Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)

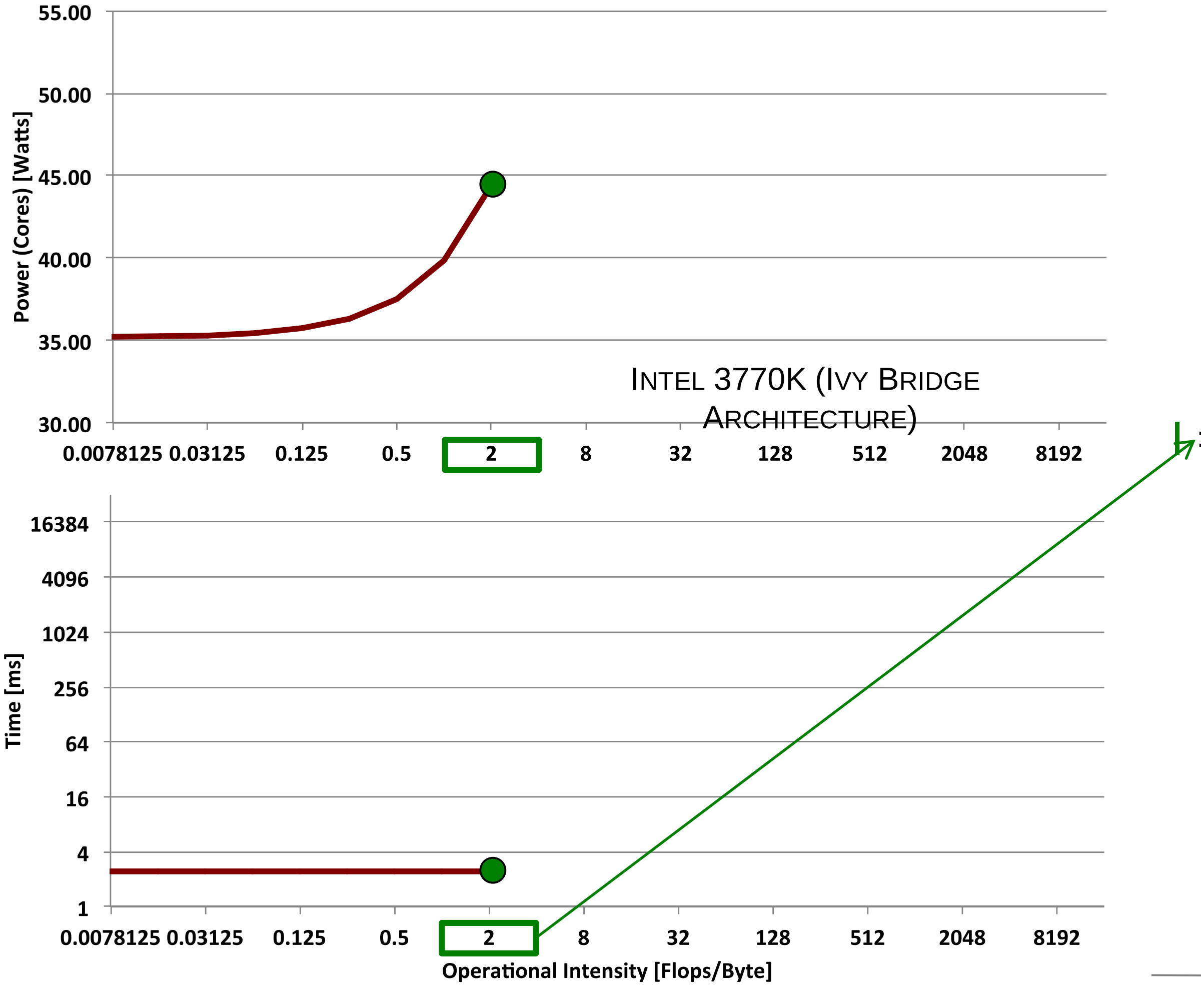


$I = f/b = 64/128$



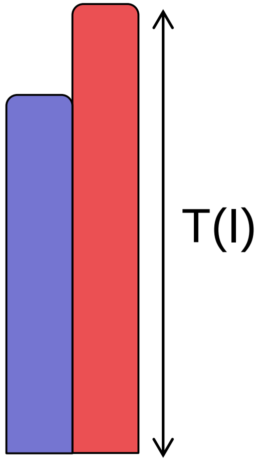
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



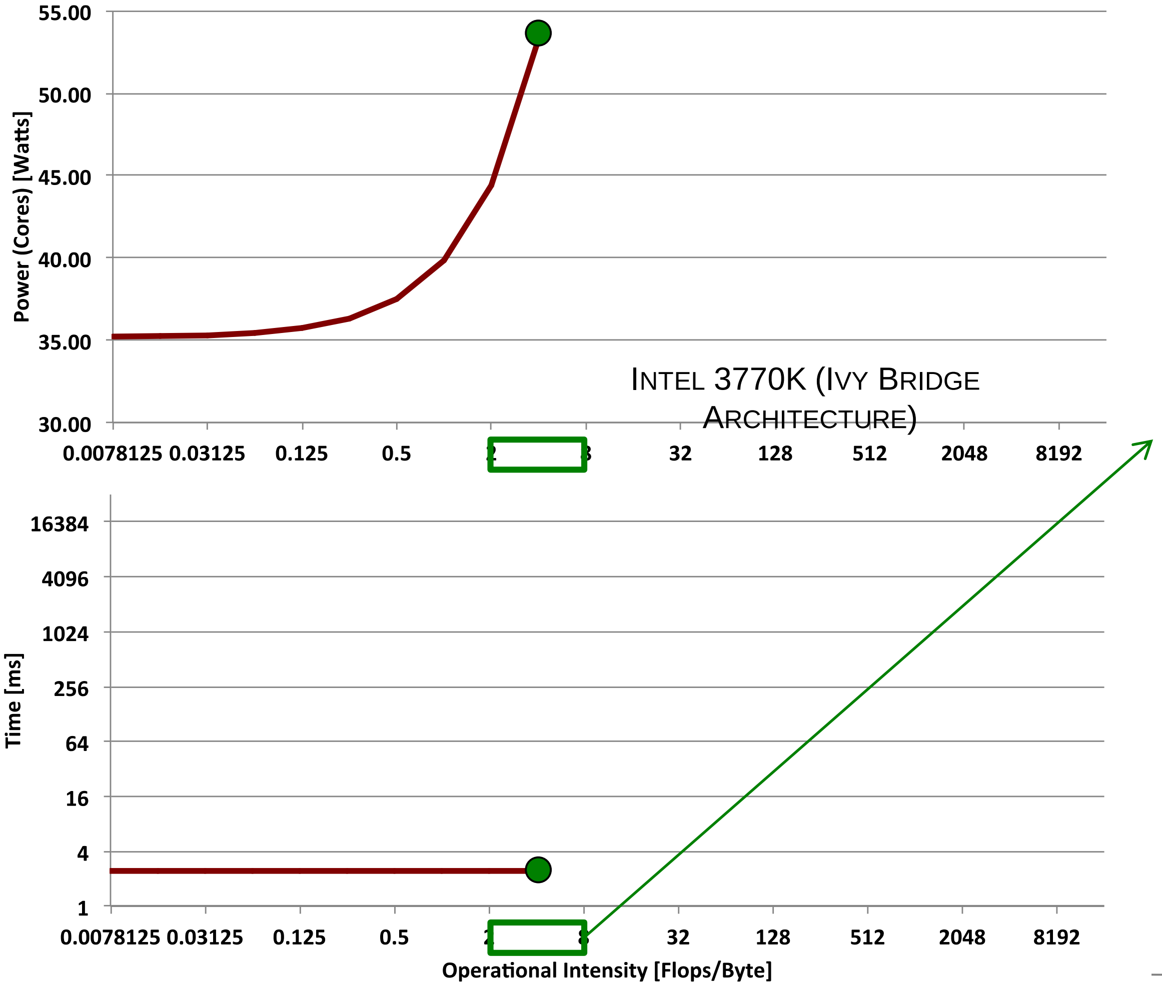
Intel 3770K (Ivy Bridge Architecture)

$I = f/b = 256/128$



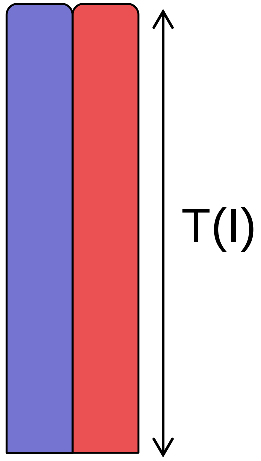
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



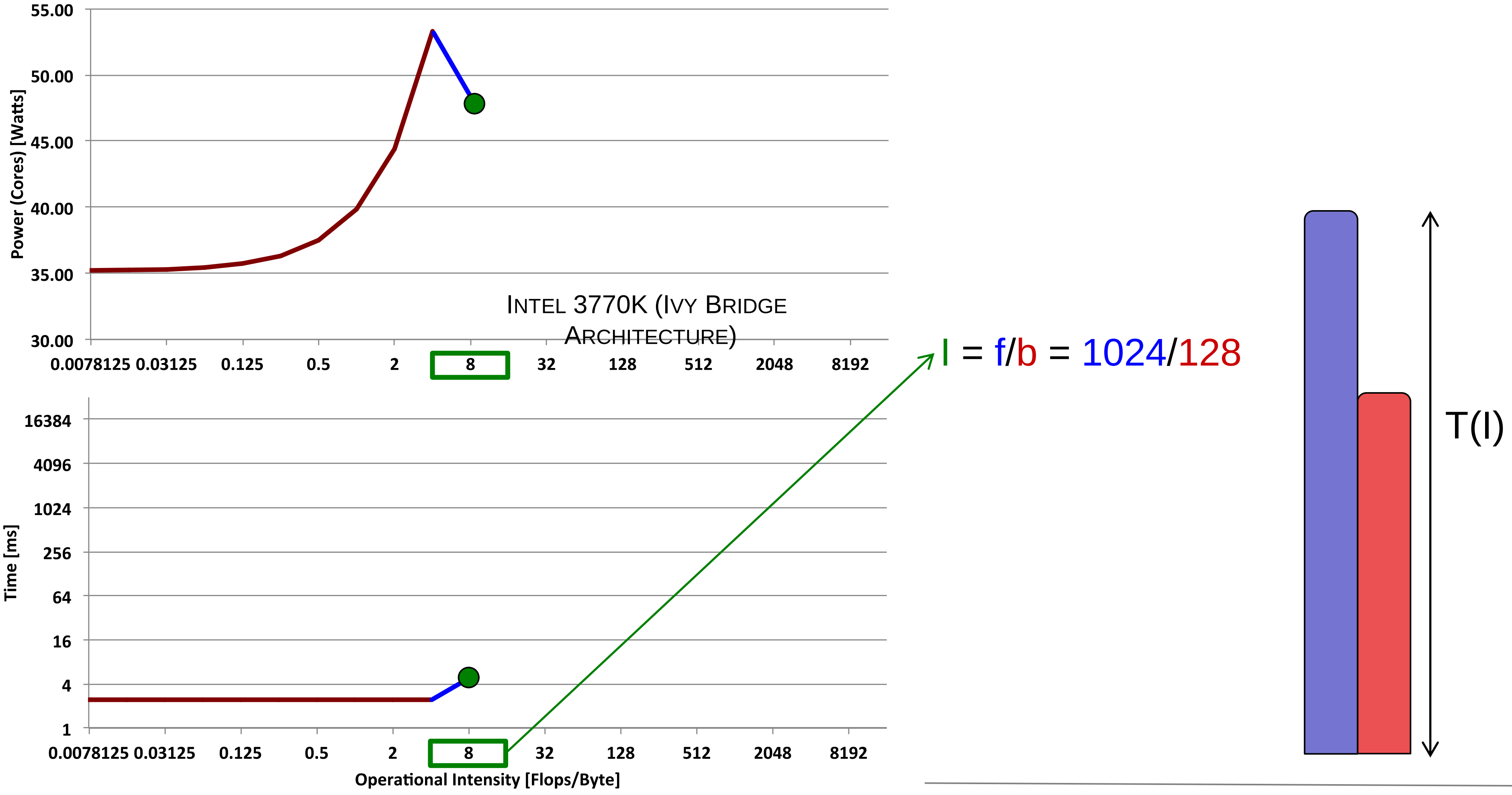
Maximum Overlap = Maximum Pow

$I = f/b \approx 512/128$



Power Roofline Model

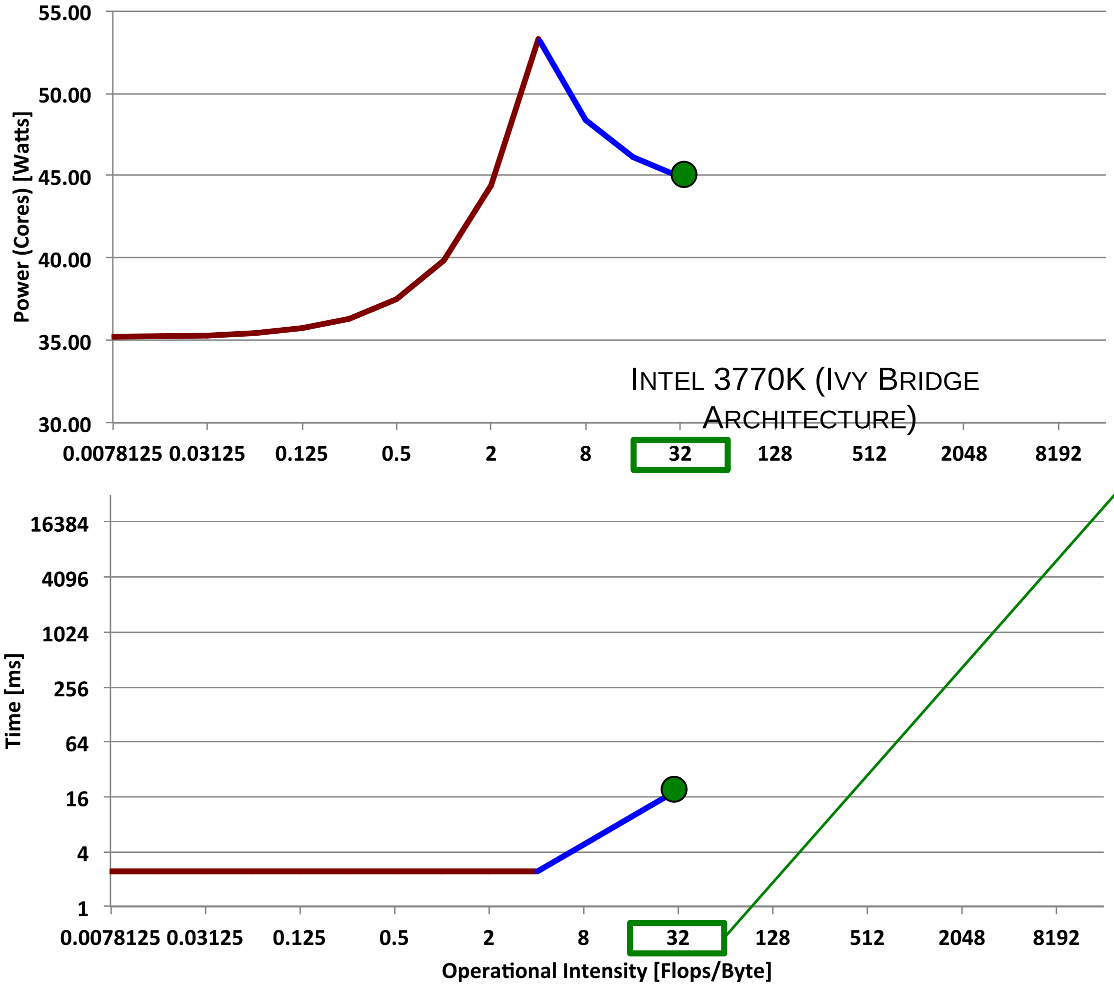
1. Power of Cores (P_C) | 2. Uncore Power (P_U) | 3. Package Power (P_P)



$I = f/b = 1024/128$

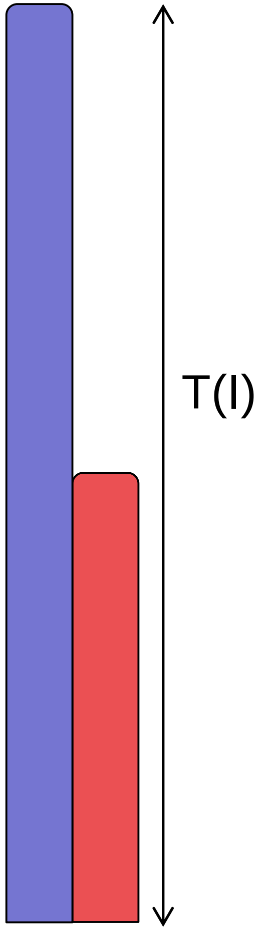
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



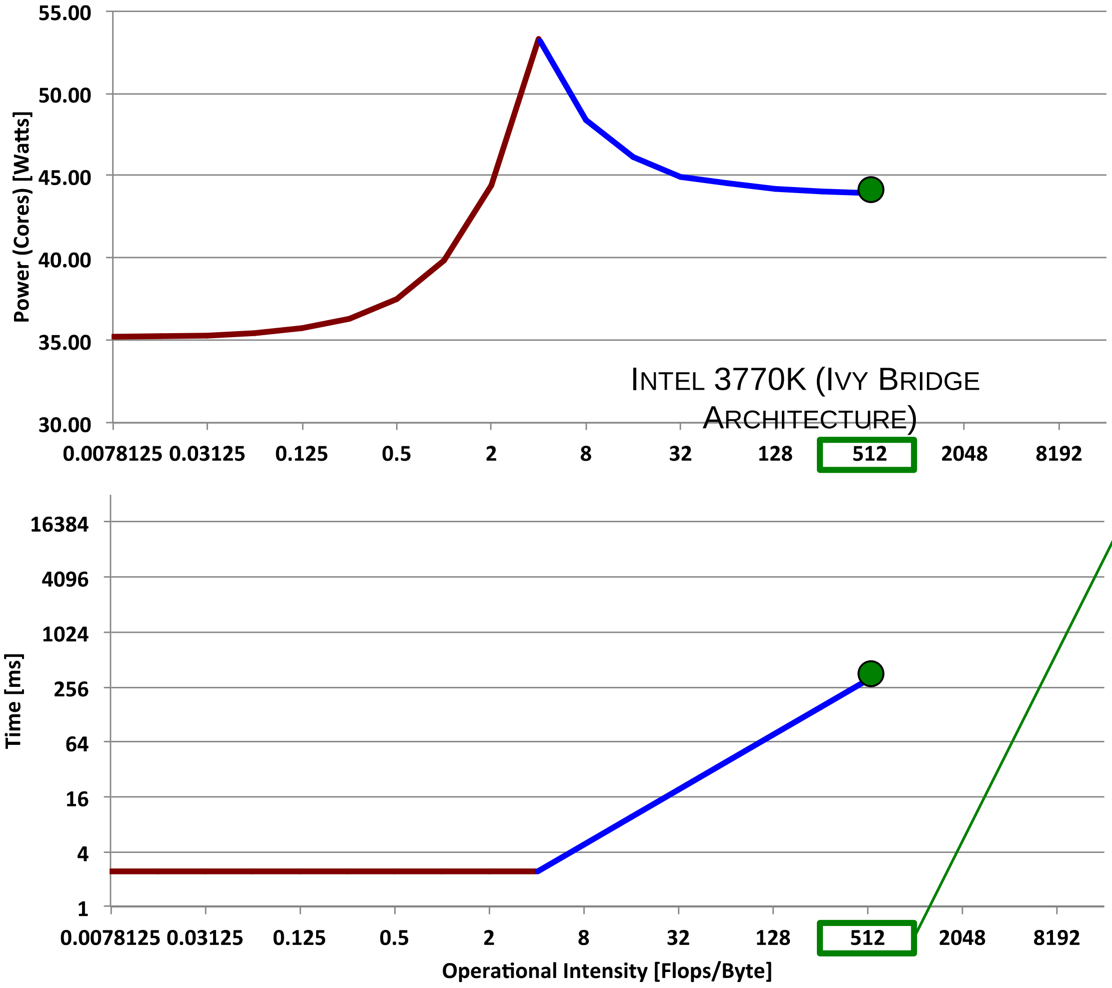
Intel 3770K (Ivy Bridge Architecture)

$I = f/b = 4096/128$



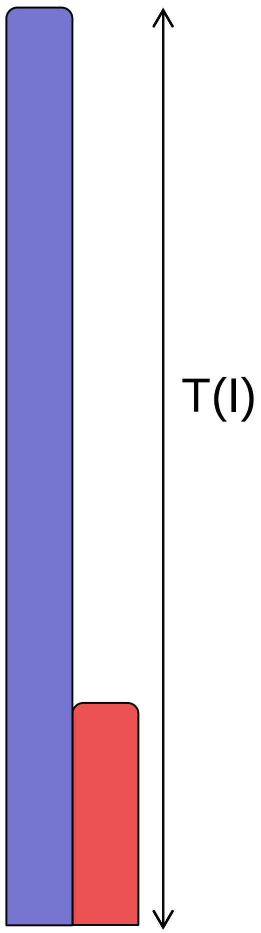
Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)



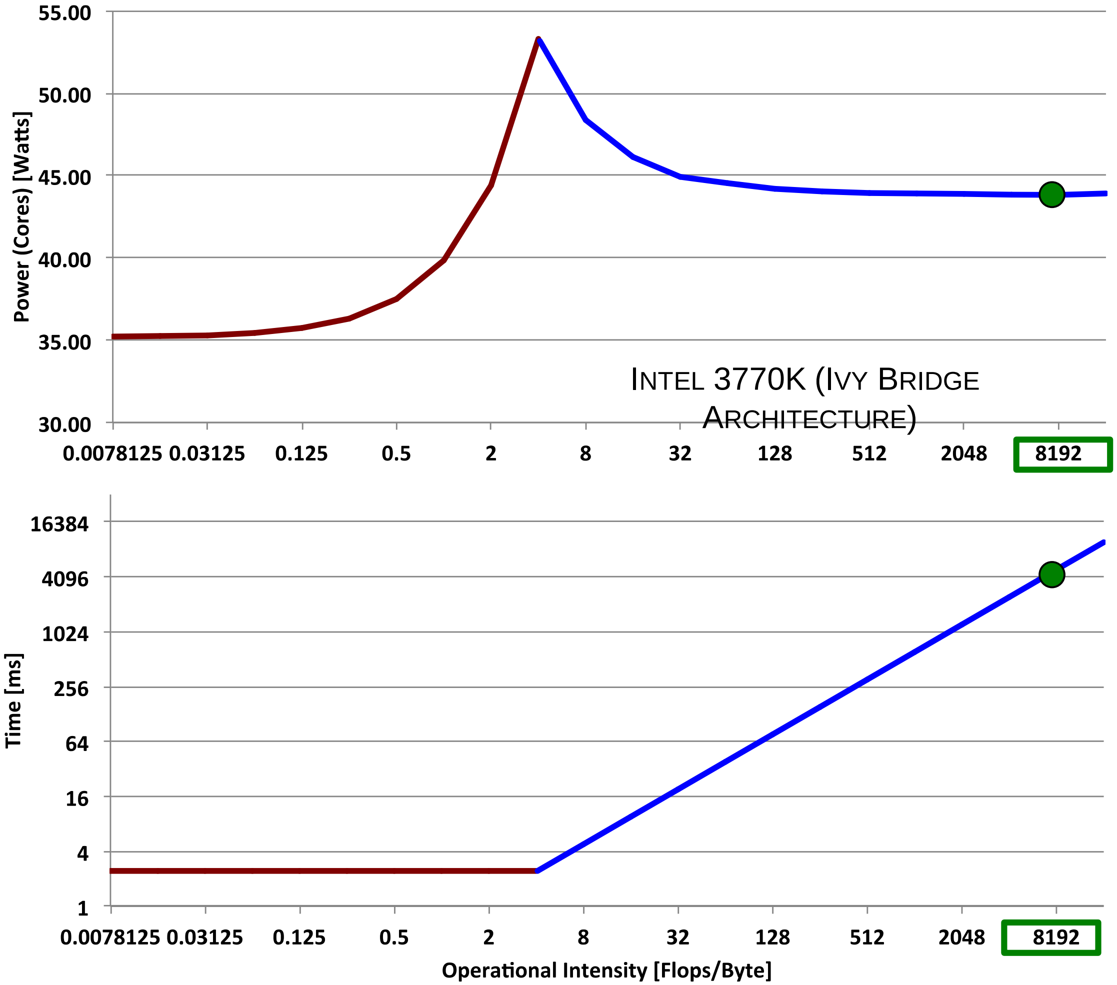
Intel 3770K (Ivy Bridge Architecture)

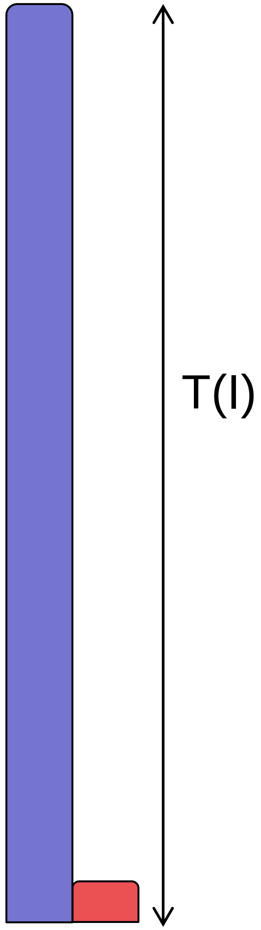
$I = f/b = 65536/128$



Power Roofline Model

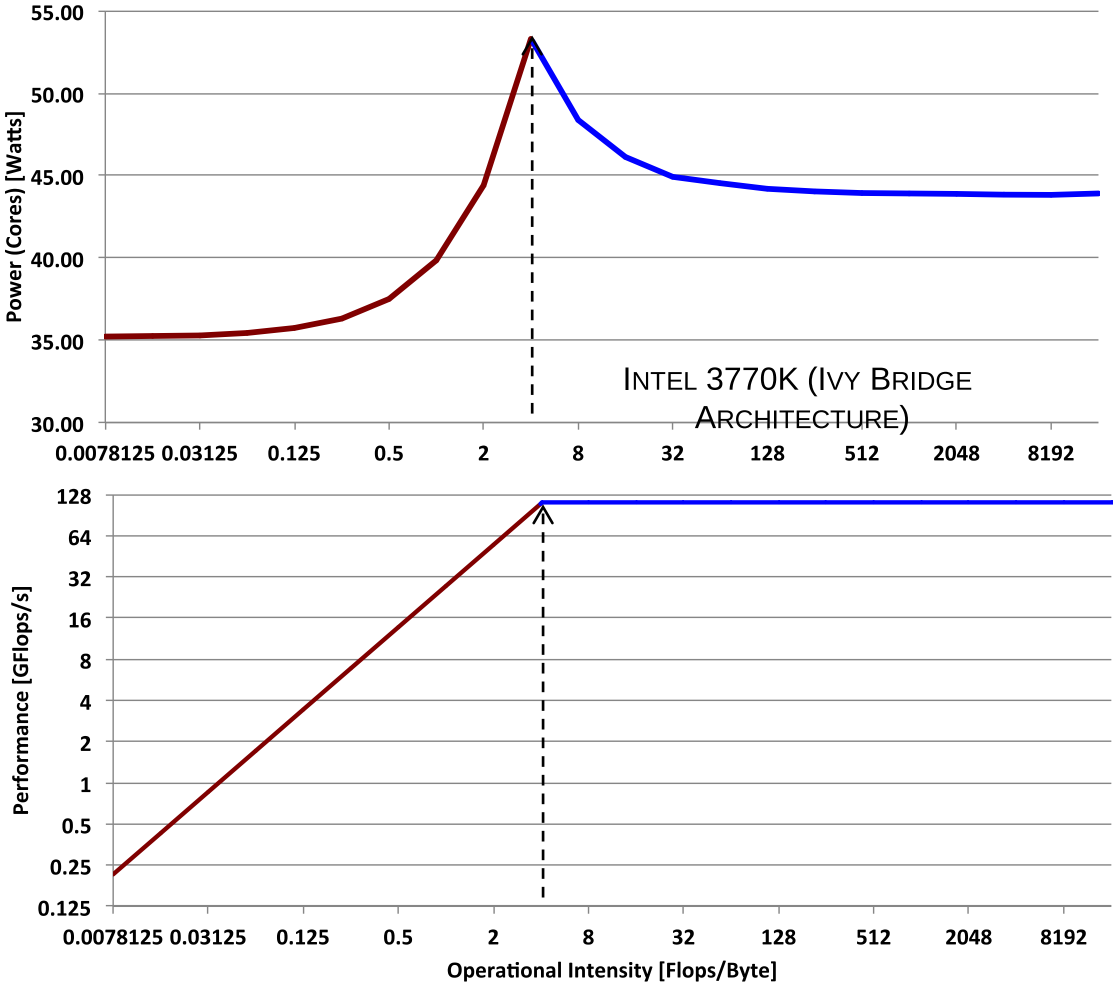
1. POWER OF CORES (P_C) 2. UNCORE POWER (P_U) 3. PACKAGE POWER (P_P)





Power Roofline Model

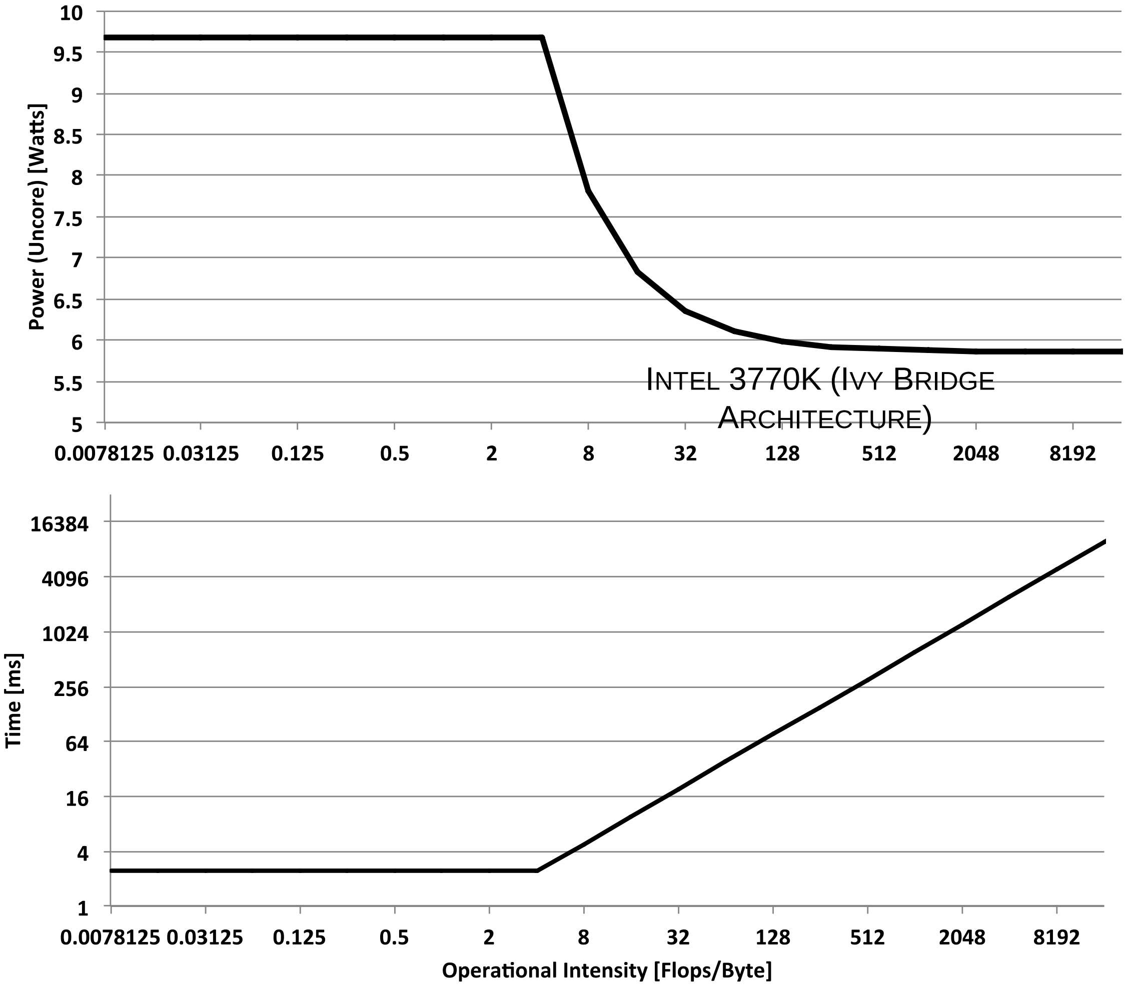
1. Power of Cores (P$_C$) 2. Uncore Power (P$_U$) 3. Package Power (P$_P$)



The most "desirable" performance point is the worst in the power domain

Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)

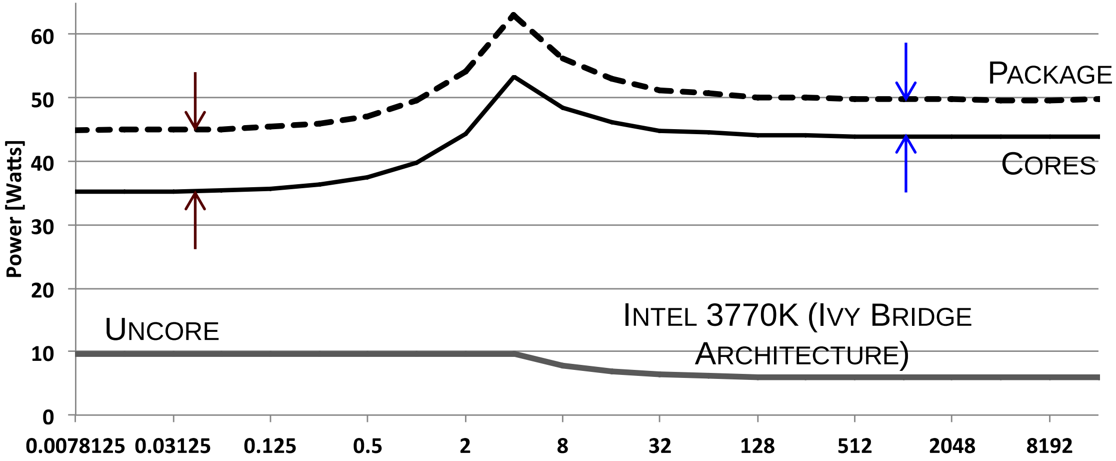


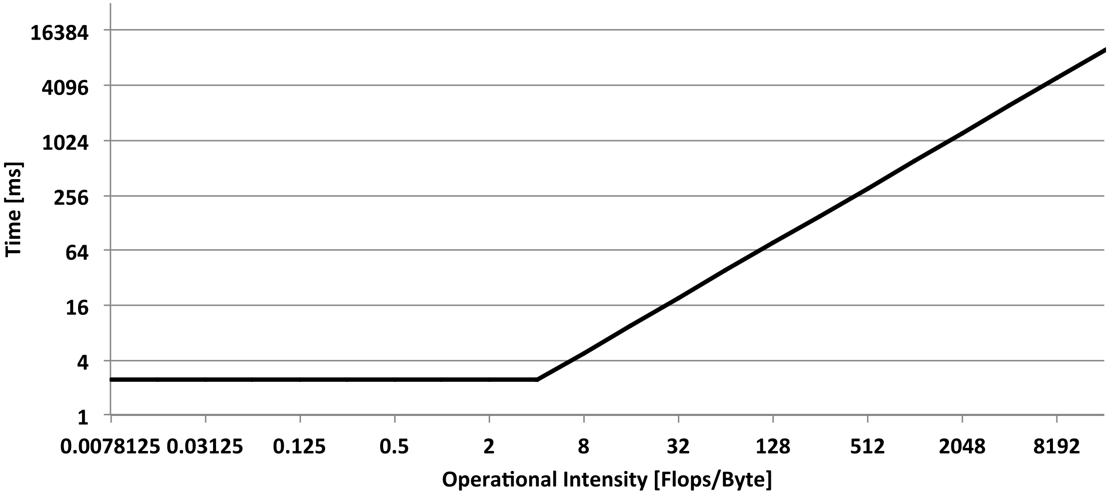
Intel 3770K (Ivy Bridge Architecture)

The impact of the off-chip memory controller power reduces with the number of memory operations and their contribution when compared to the FP operations

Power Roofline Model

1. Power of Cores (P_C) 2. Uncore Power (P_U) 3. Package Power (P_P)

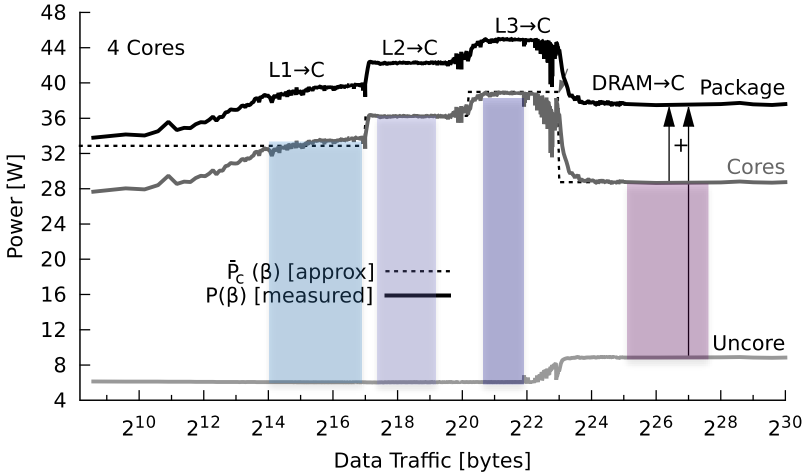


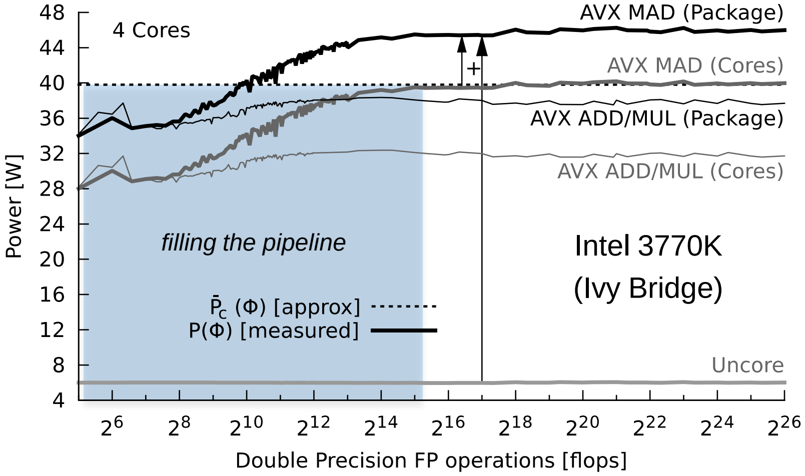


The Package Power depends on superposition with the Power of Cores, the Uncore Power, and the other components on the chip

Power Cache-aware Roofline Model

- Different power domains: *Cores* + *Uncore* = *Package*



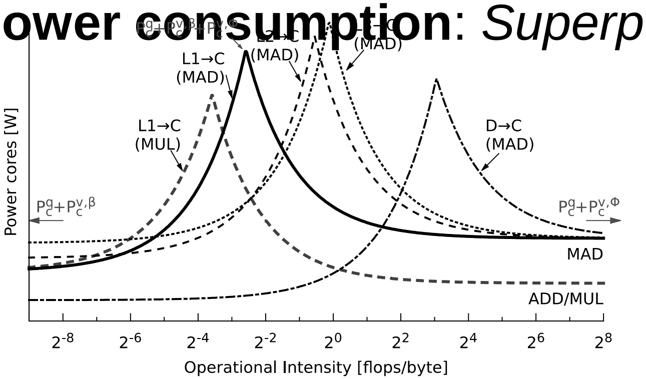


Intel 3770K
(Ivy Bridge)

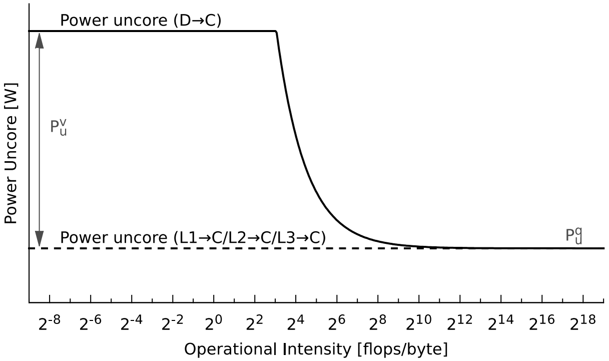
Power Cache-aware Roofline Model

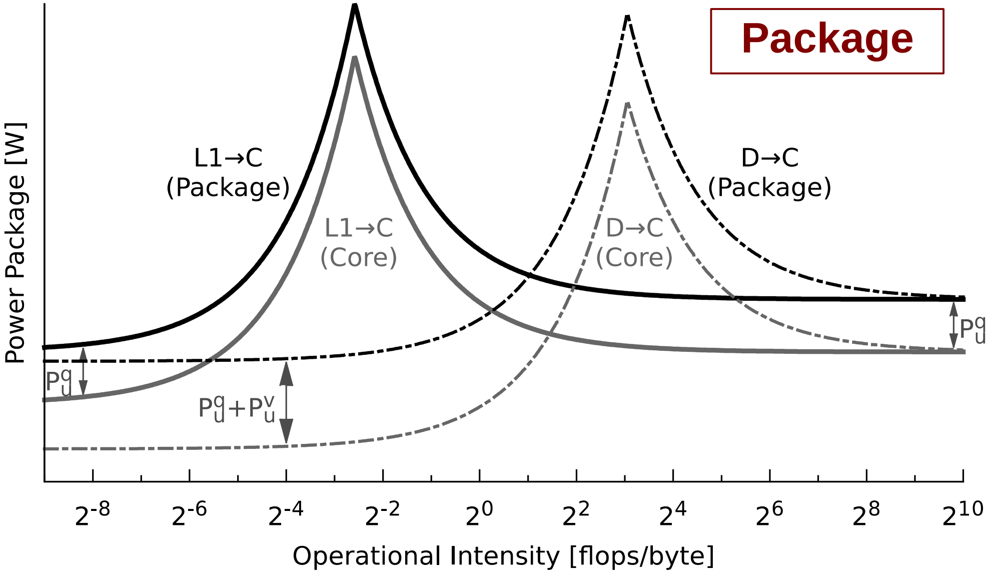
- **Performance**: Computations (*flops*) and communication (*bytes*) overlap in time
- **Power consumption**: *Superposed* contributions of *flops* and *bytes*

Cores



Uncore

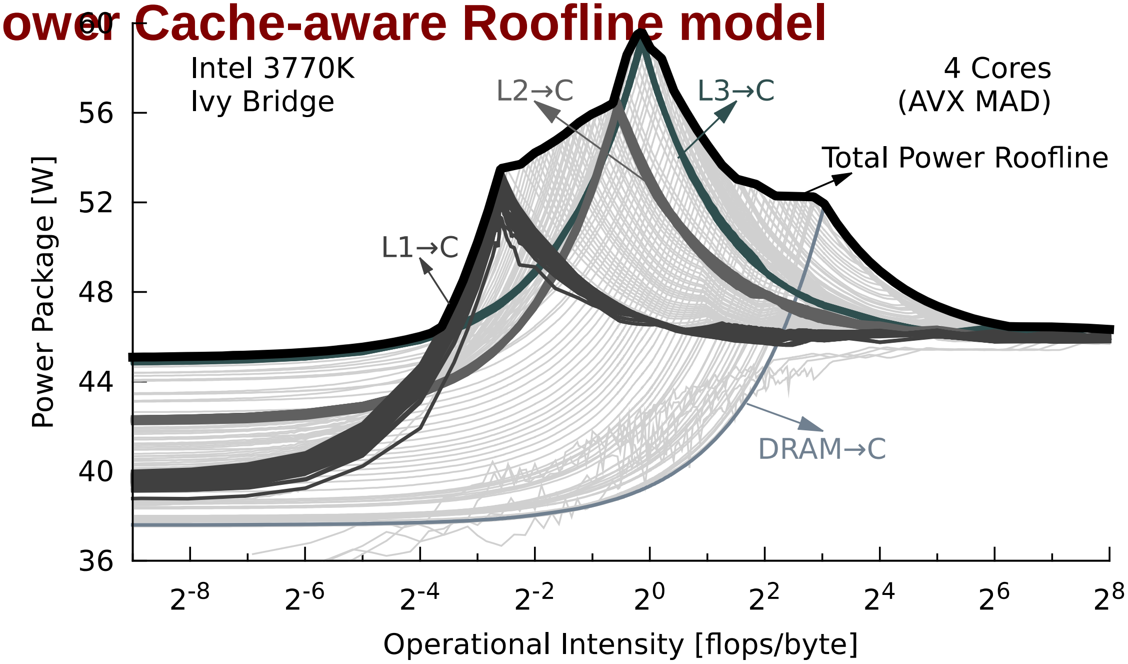




Package

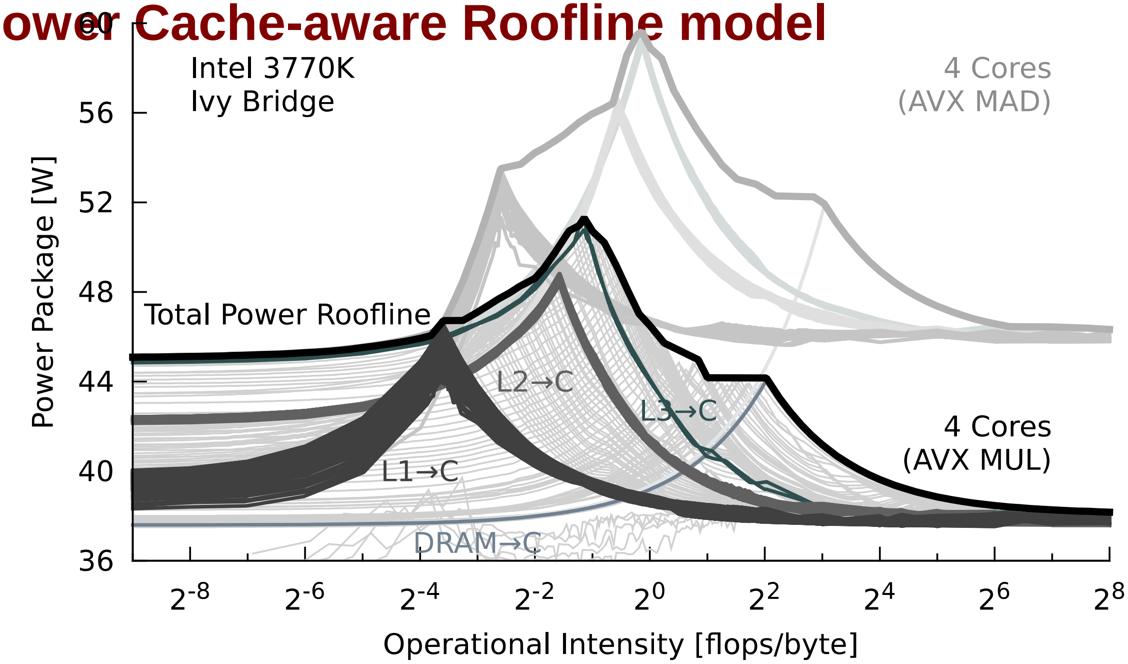
Power Cache-aware Roofline Model

- **Performance**: Computations (*flops*) and communication (*bytes*) overlap in time
- **Power consumption**: Superposed contributions of *flops* and *bytes*
- **Total Power Cache-aware Roofline model**



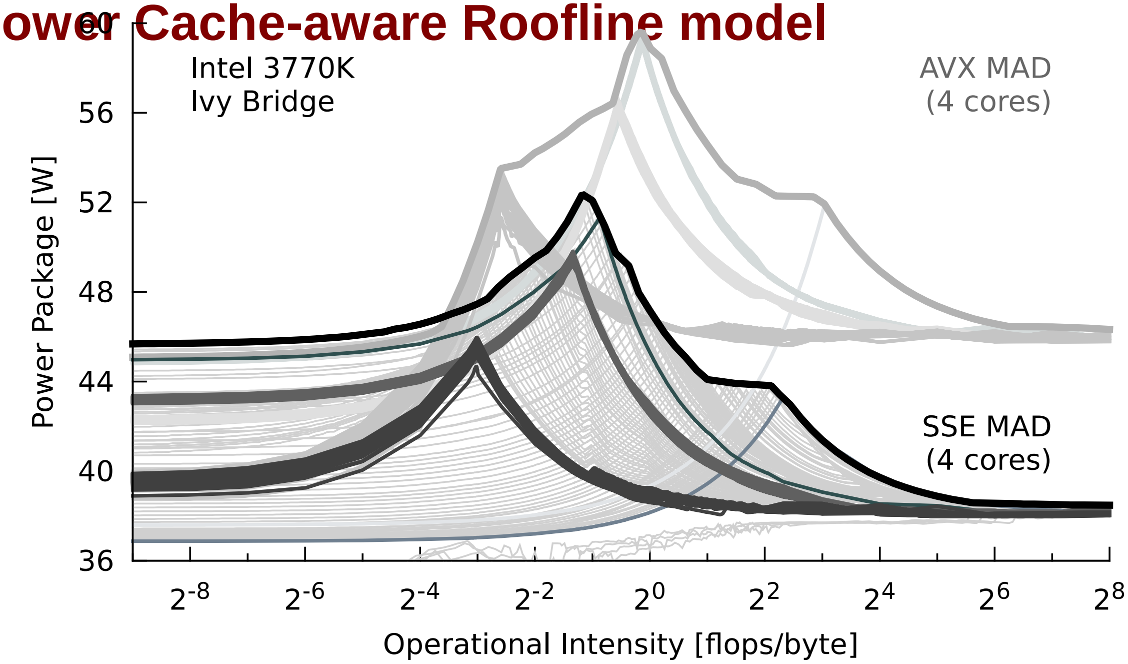
Power Cache-aware Roofline Model

- **Performance**: Computations (*flops*) and communication (*bytes*) overlap in time
- **Power consumption**: Superposed contributions of *flops* and *bytes*
- **Total Power Cache-aware Roofline model**



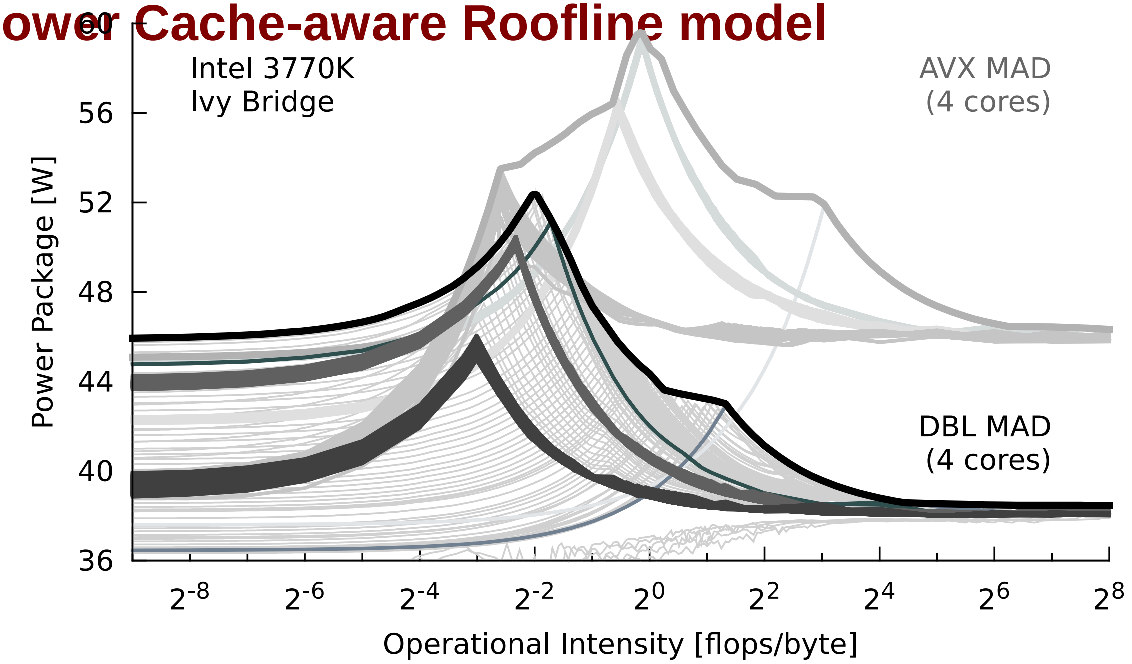
Power Cache-aware Roofline Model

- **Performance**: Computations (*flops*) and communication (*bytes*) overlap in time
- **Power consumption**: Superposed contributions of *flops* and *bytes*
- **Total Power Cache-aware Roofline model**



Power Cache-aware Roofline Model

- **Performance**: Computations (*flops*) and communication (*bytes*) overlap in time
- **Power consumption**: Superposed contributions of *flops* and *bytes*
- **Total Power Cache-aware Roofline model**



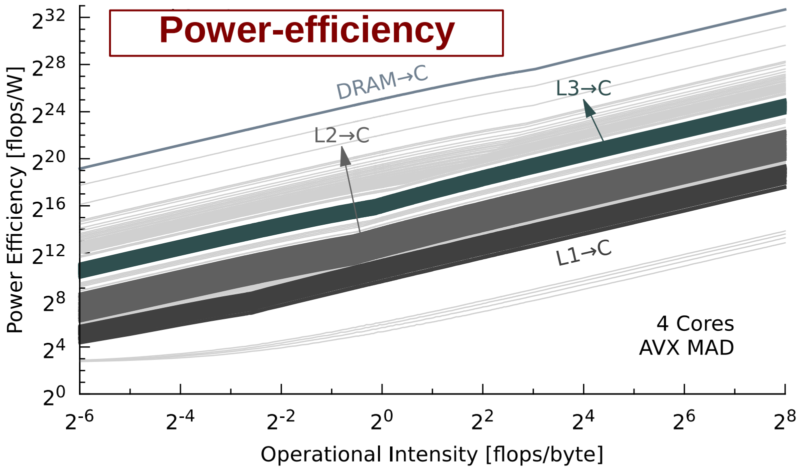
Outline

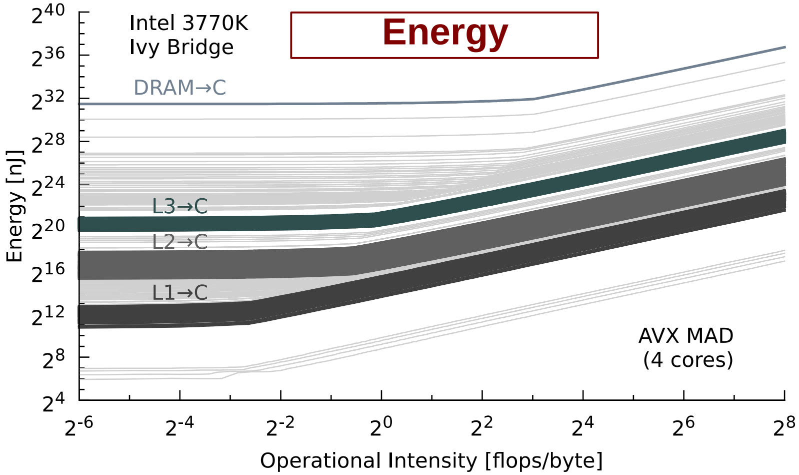
THE CACHE-AWARE ROOFLINE MODEL:

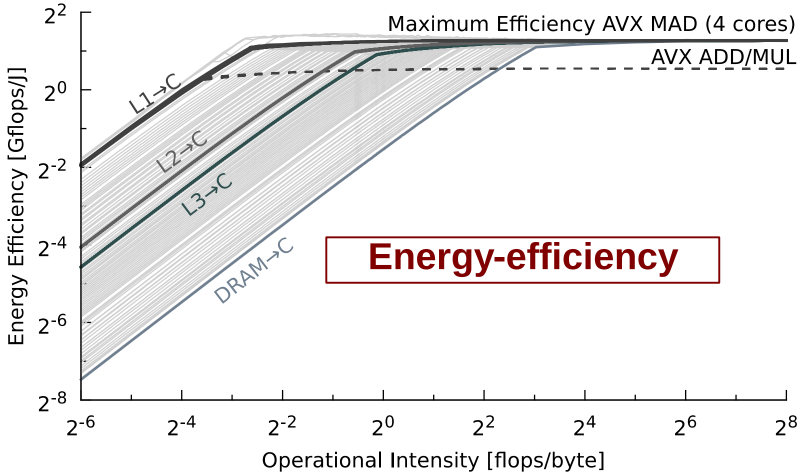
- PERFORMANCE
- POWER
- EFFICIENCY*

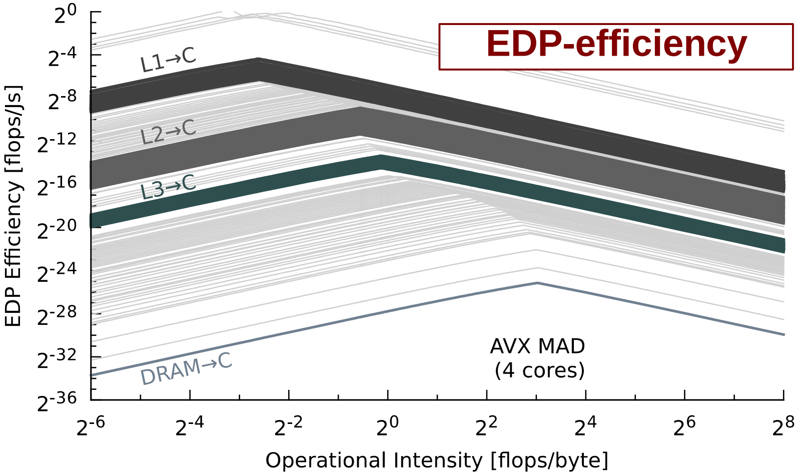
* *Ilić, A., Pratas, F. and Sousa, L., “Beyond the Roofline: Power, Energy and Efficiency Modeling for Multicores”*

Efficiency Cache-Aware Roofline Model



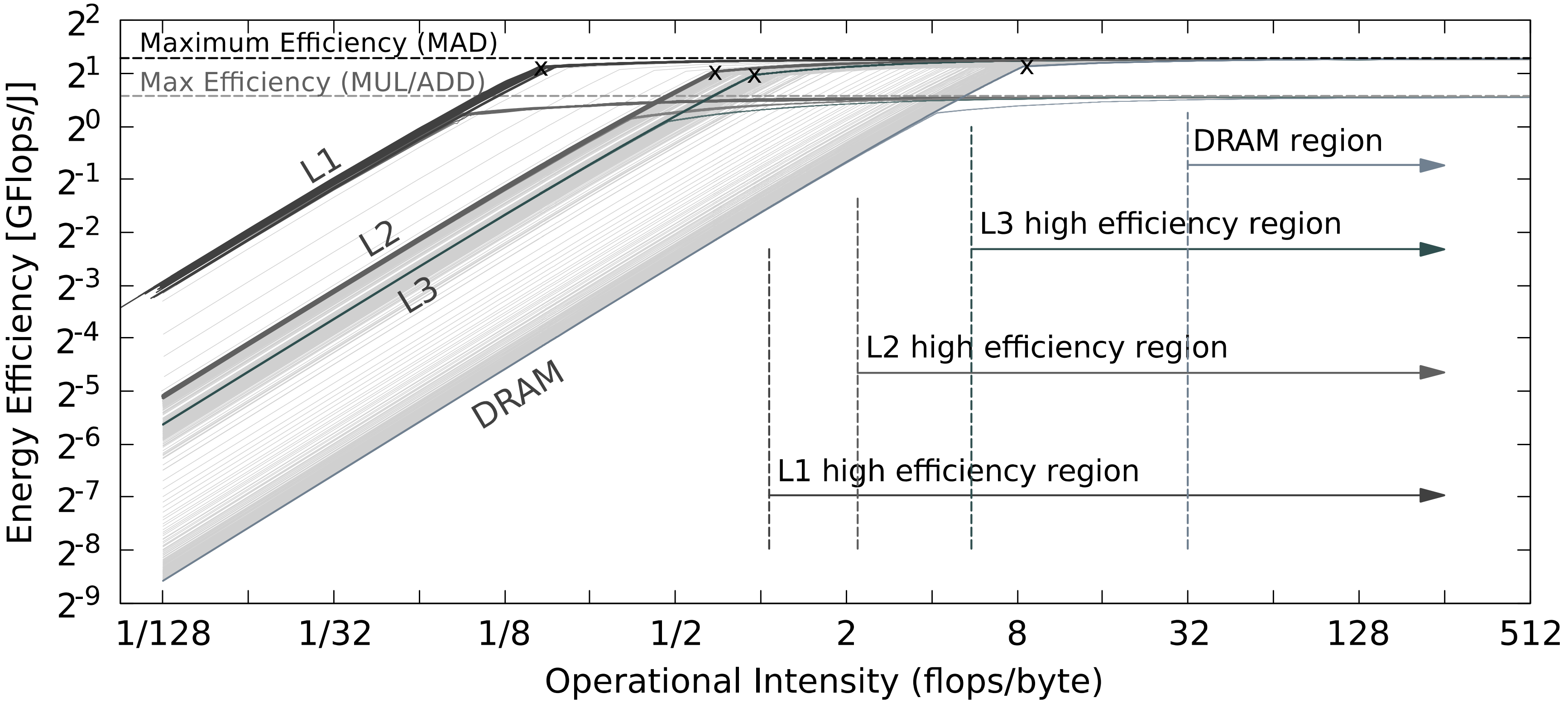






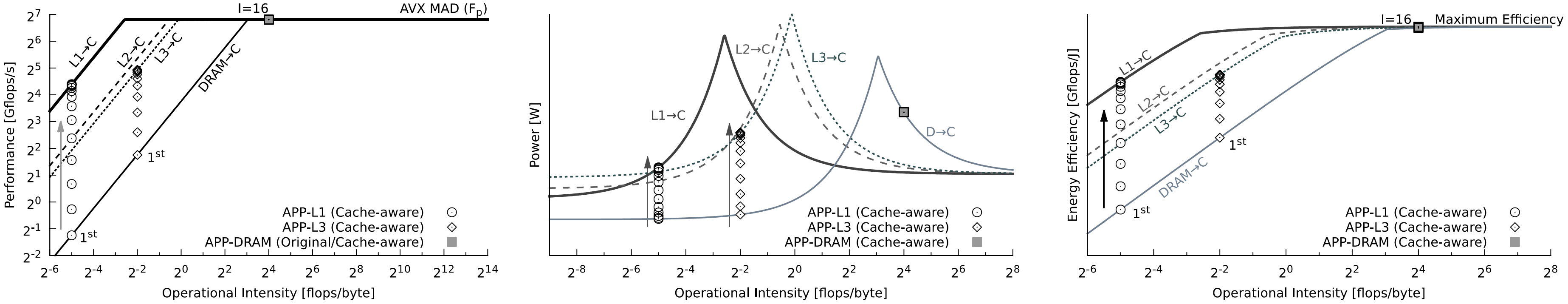
Efficiency Cache-Aware Roofline Model

Energy-efficiency



Application Behavior

Cache-aware Roofline Models*

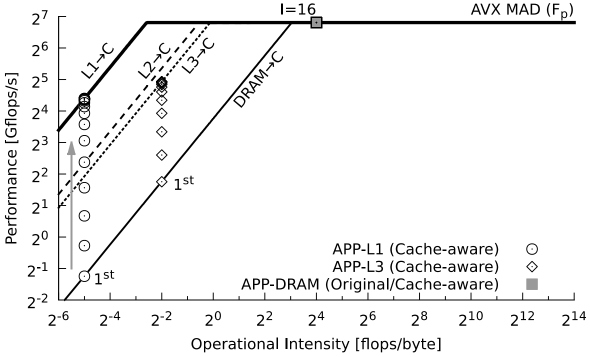


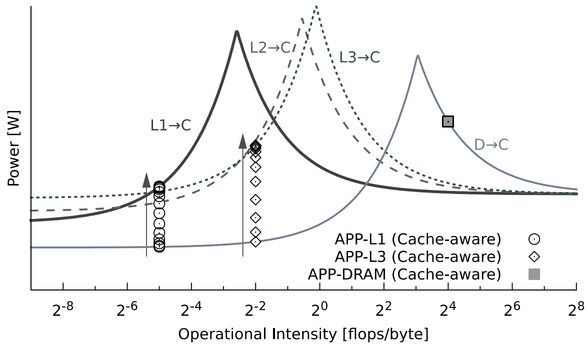
** Ilić, A., Pratas, F. and Sousa, L., "Cache-aware Roofline model: Upgrading the loft", IEEE Computer Architecture Letters (2013)*

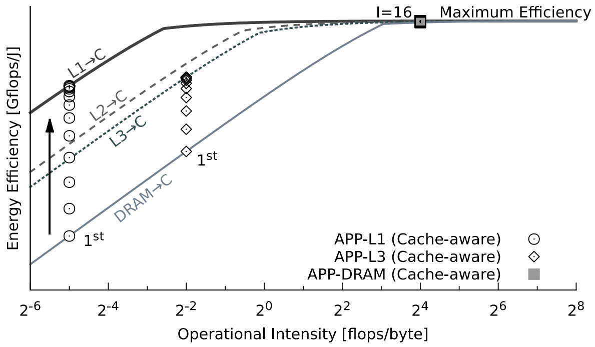
** Ilić, A., Pratas, F. and Sousa, L., "Beyond the Roofline: Power, Energy and Efficiency Modeling for Multicores" (submitted)*

Application Behavior

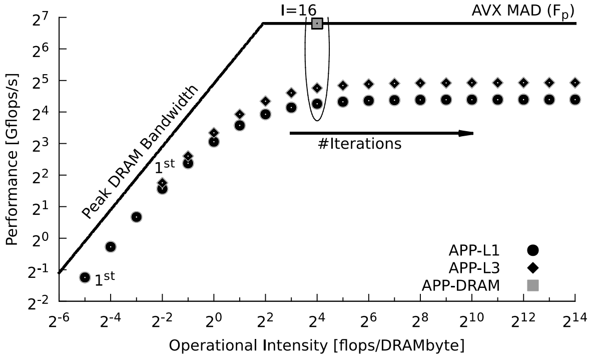
Cache-aware Roofline Models*

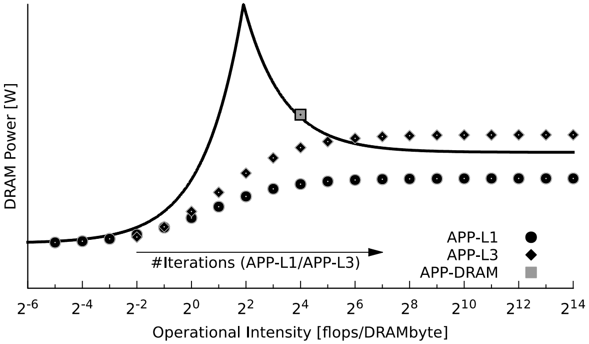


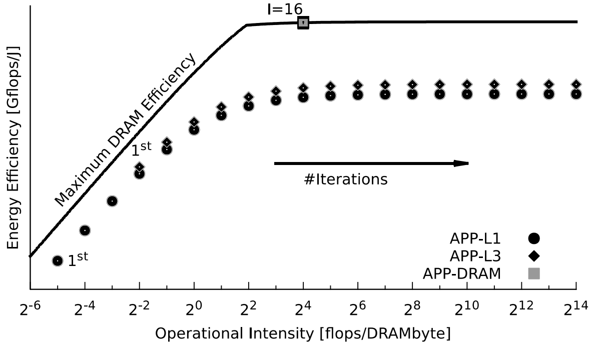




Original Roofline Models**







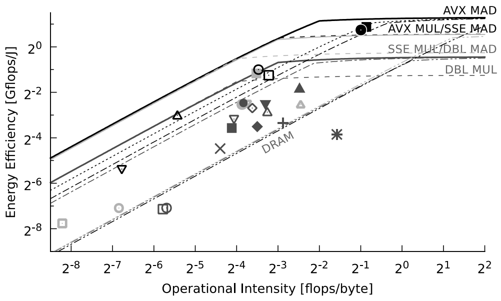
** A. Ilić, F. Pratas, and L. Sousa, “Cache-aware Roofline model: Upgrading the loft”, IEEE Computer Architecture Letters (2013)*

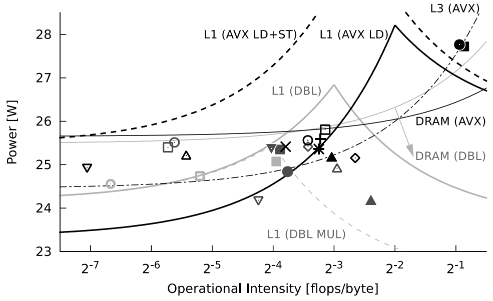
*** S. Williams, et.al. “Roofline: An insightful visual performance model for multicore architectures”, Comm. of the ACM (2009)*

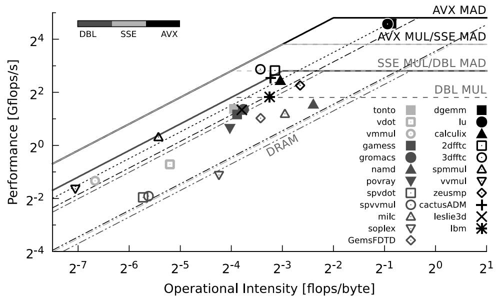
*** J. Choi, D. Bedard, R. Fowler, and R. Vuduc. “A roofline model of energy”, IPDPS (2013/2014)*

Cache-aware Roofline Model: Use Cases

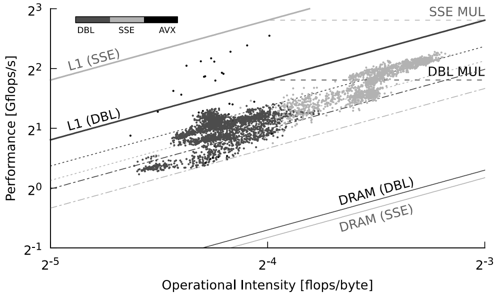
Application Characterization

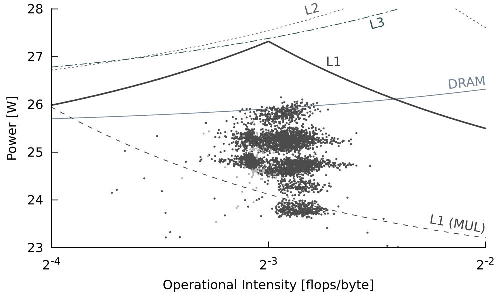


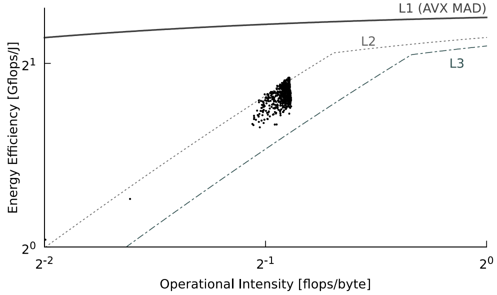




Online Monitoring







* *Ilić, A., Pratas, F. and Sousa, L., "Beyond the Roofline: Power, Energy and Efficiency Modeling for Multicores" (submitted)*

* *Antão, D., Taniça, L., Ilić, A., Pratas, F., Tomás, P., and Sousa, L., "Monitoring Performance and Power for Application Characterization with Cache-aware Roofline Model", PPAM'13*

Conclusions

BUNCH OF CACHE-AWARE ROOFLINE MODELS (EXPERIMENTALLY VERIFIED)

- **(Total) Performance**
- **(Total) Power Roofline Models**: for several domains, i.e., power of cores, uncore power and complete package power
- **Energy Roofline Model**: Time + Power Domains
- **Energy-Efficiency Roofline Model**: Performance + Power Domains
- **EDP-based Roofline Model**: Performance + Energy Domains

ALL MODELS OBTAINED WITHIN A SINGLE TEST PROCEDURE

- THE SAME TIME NEEDED AS FOR CONSTRUCTING THE PERFORMANCE ROOFLINE MODEL

FUTURE WORK

- INTEGRATION OF THE PERFORMANCE CARM IN INTEL TOOLS
- GPUS, ARMS, COMPLETE SYSTEM …

Questions?

Thank you!

Further readings:

A. Ilic, F. Pratas, and L. Sousa, ***"Cache-aware Roofline model: Upgrading the loft",*** IEEE Computer Architecture Letters, CAL (2013)

A. Ilic, F. Pratas, and L. Sousa, ***"CARM: Cache-Aware Performance, Power and Energy-Efficiency Roofline Modeling",*** Intel CATC (2015)

L. Taniça, A. Ilic, P. Tomás, and L. Sousa, ***"SchedMon: A Performance and Energy Monitoring Tool for Modern Multi-cores",*** MuCoCoS/Euro-Par (2014)

D. Antão, L. Taniça, A. Ilic, F. Pratas, P. Tomás, and L. Sousa***, "Monitoring Performance and Power for Application Characterization with Cache-aware Roofline Model",*** PPAM (2013)

A. Ilic, F. Pratas, and L. Sousa, ***"Beyond the Roofline: Power, Energy and Efficiency Modeling for Multicores"*** *(#$%&)*

inesc id lisboa