Coping with Complexity: CPUs, GPUs and Real-world Applications

Leonel Sousa,
Frederico Pratas, Svetislav Momcilovic and Aleksandar Ilic

9th Scheduling for Large Scale Systems Workshop
Lyon, France

July 2014

Motivation

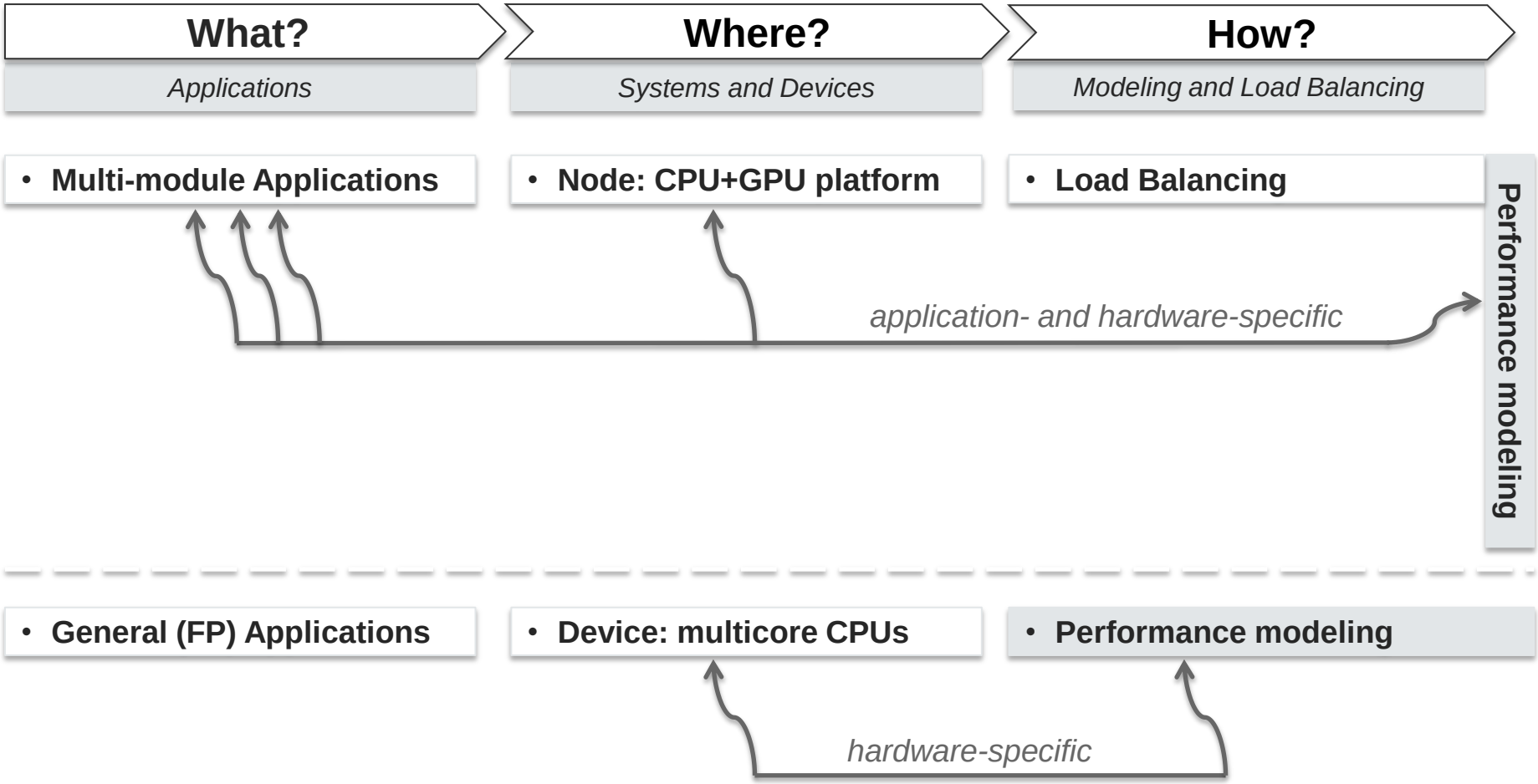
- **Commodity computers = Heterogeneous systems**
  - Multi-core General Purpose Processors (CPUs)
  - Graphics Processing Units (GPUs)
  - Special accelerators, co-processors, FPGAs, mobile and wearable systems
- **Significant computing power**
  - Not yet fully exploited for efficient collaborative computing
- **Heterogeneity makes it really difficult!**
  - Applications, devices, interconnects, systems…
  - Performance modeling and load balancing for efficient computing

Outline

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*
• **Multi-module Applications**	• **Node: CPU+GPU platform**	• **Load Balancing**
• **General (FP) Applications**	• **Device: multicore CPUs**	• **Performance modeling**

Performance modeling

Outline


What?
Where?
How?
Applications
Systems and Devices
Modeling and Load Balancing
• Multi-module Applications
• Node: CPU+GPU platform
• Load Balancing
application- and hardware-specific
Performance modeling
• General (FP) Applications
• Device: multicore CPUs
• Performance modeling
hardware-specific

Outline

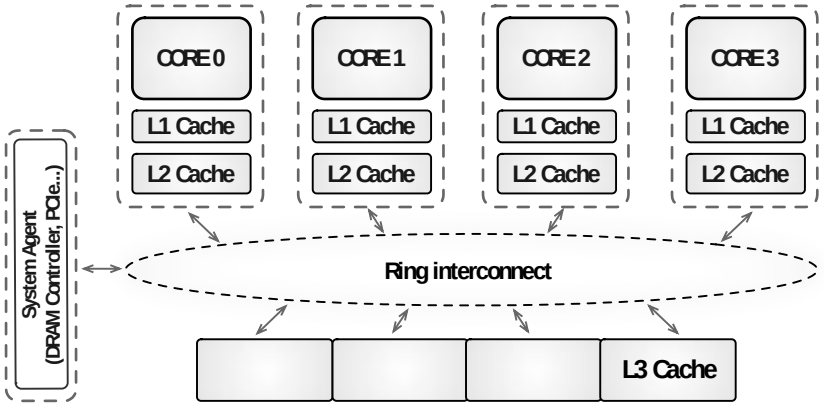
What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*
• Multi-module Applications	• **Node: CPU+GPU platform**	• Load Balancing
• General (FP) Applications	• Device: multicore CPUs	• Performance modeling

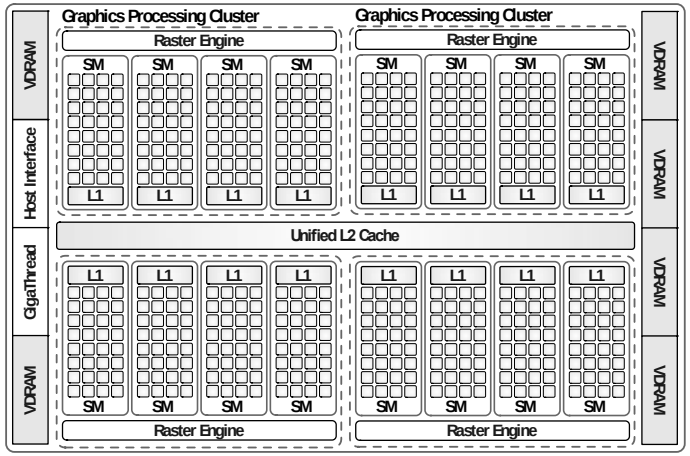
Performance modeling

Node: CPU+GPU platform

What? | **Where?** | How?

Applications | *Systems and Devices* | *Modeling and Load Balancing*



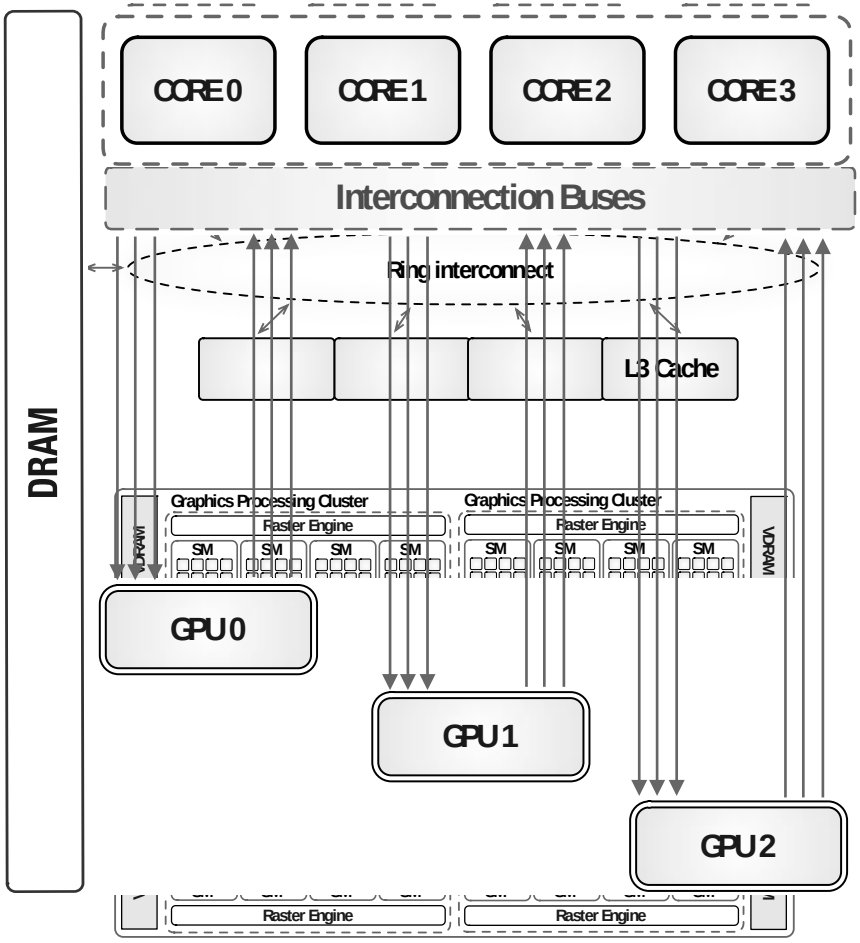


- **Multi-core CPU** (Master)
  - Replication of identical cores
  - Memory hierarchy: private and shared caches
  - Programming: OpenMP, Pthreads, OpenCL
- **GPUs/Accelerators** (distant workers)
  - Large number of “simple” cores
  - Complex memory hierarchy: global/local/shared
  - Programming: CUDA, OpenCL

Node: CPU+GPU platform

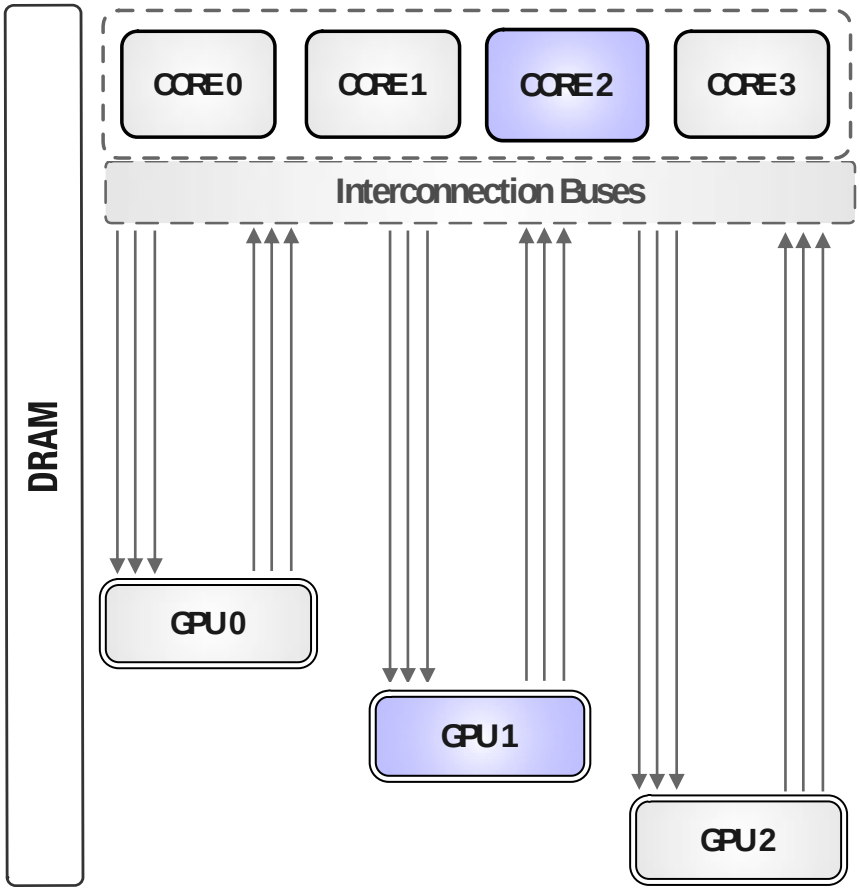
What? | **Where?** | How?

Applications | *Systems and Devices* | *Modeling and Load Balancing*

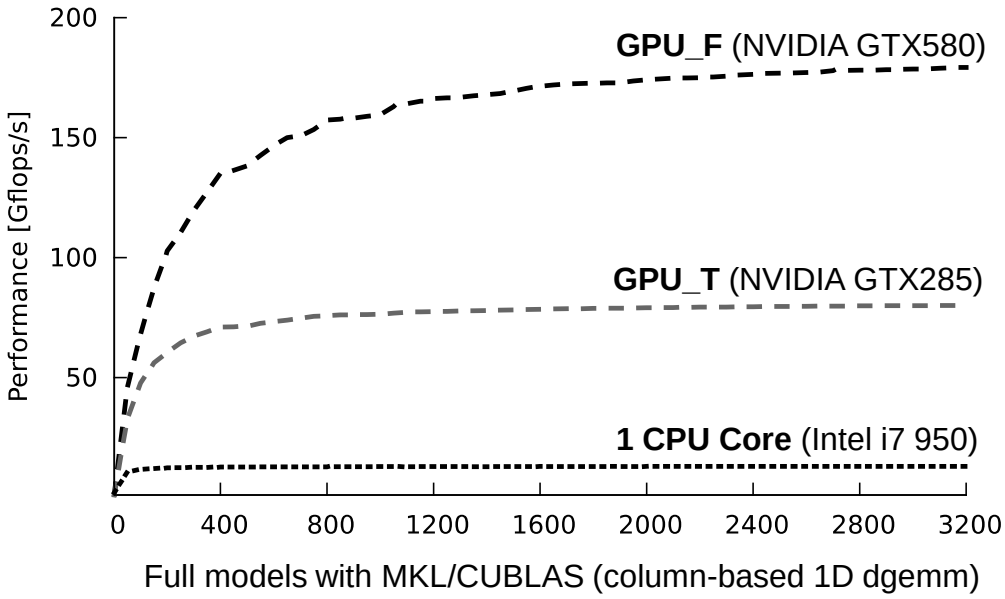


- **Multi-core CPU** (Master)
  - Replication of identical cores
  - Memory hierarchy: private and shared caches
  - Programming: OpenMP, PThreads, OpenCL
- **GPUs/Accelerators** (distant workers)
  - Large number of “simple” cores
  - Complex memory hierarchy: global/local/shared
  - Programming: CUDA, OpenCL
  - Configuration: Maxeler data-flow engines
- **Collaborative CPU+GPU execution**
  - Architectural diversity and programmability
  - Code parallelization on a per device basis
  - Integration into a single unified environment (OpenCL, StarPU, StarSs, CHPS, …)

Node: CPU+GPU platform



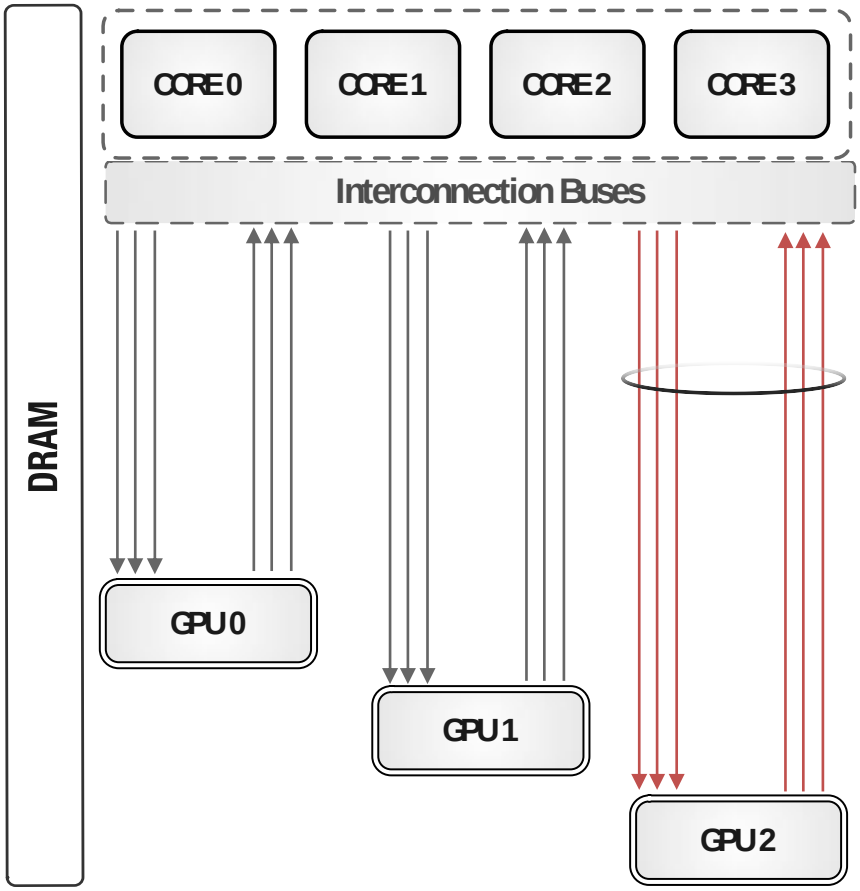
- **GPU vs. CPU performance:**
  - GPU usually much faster, but not for all problems
  - Performance might differ by orders of magnitude
  - Accurate performance modeling is required!



Node: CPU+GPU platform

What? | **Where?** | How?

Applications | *Systems and Devices* | *Modeling and Load Balancing*

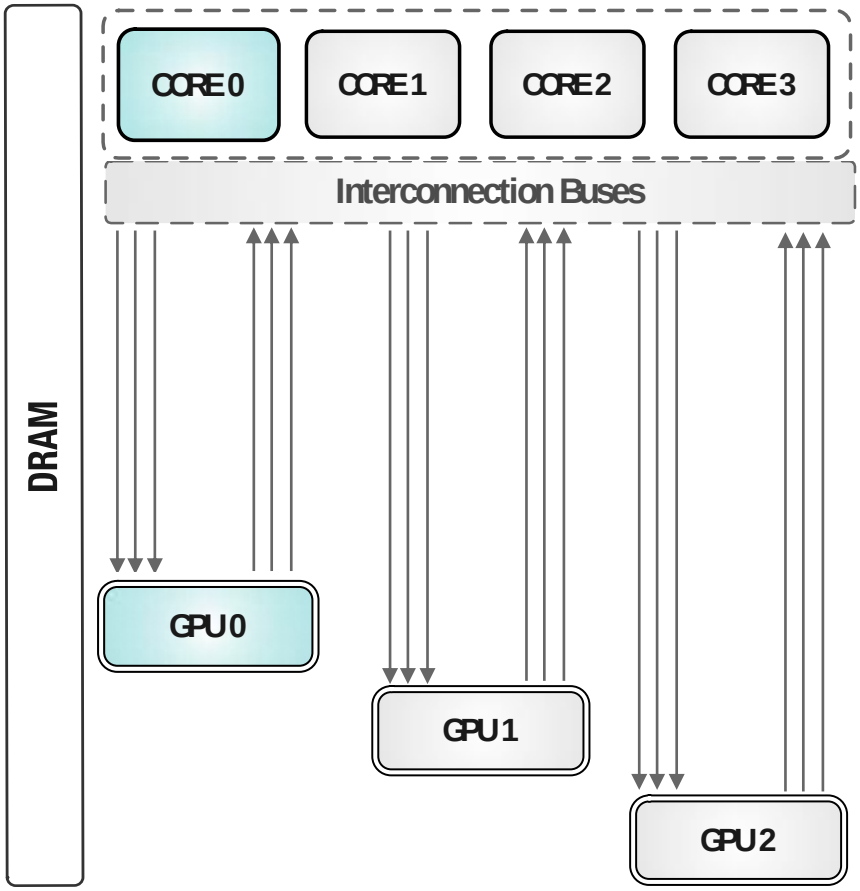


- **GPU vs. CPU performance:**
  - GPU usually much faster, but not for all problems
  - Performance might differ by orders of magnitude
- **GPUs are connected via PCI Express**
  - Bidirectional lines
  - Asymmetric bandwidth (in different directions)

Node: CPU+GPU platform

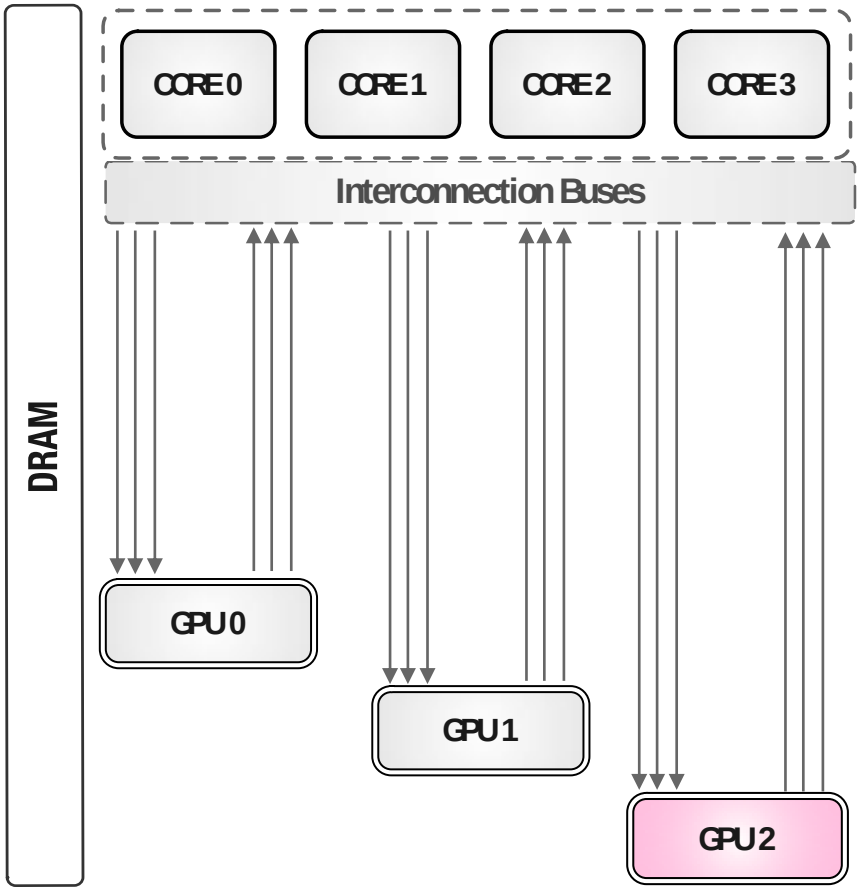
What? | **Where?** | How?

Applications | *Systems and Devices* | *Modeling and Load Balancing*



- **GPU vs. CPU performance:**
  - GPU usually much faster, but not for all problems
  - Performance might differ by orders of magnitude
- **GPUs are connected via PCI Express**
  - Bidirectional lines
  - Asymmetric bandwidth (in different directions)
- **GPUs are co-processors**
  - CPU Core/Thread initiates all data-transfers and GPU kernel calls
  - Core is usually completely devoted (underused)

Node: CPU+GPU platform



- **GPU vs. CPU performance:**
  - GPU usually much faster, but not for all problems
  - Performance might differ by orders of magnitude
- **GPUs are connected via PCI Express**
  - Bidirectional lines
  - Asymmetric bandwidth (in different directions)
- **GPUs are co-processors**
  - CPU Core/Thread initiates all data-transfers and GPU kernel calls
  - Core is usually completely devoted (underused)
- **GPUs** do not benefit from paging
  - **Limited** global **memory!**

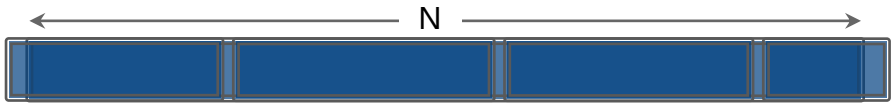
Outline

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*
• **Multi-module Applications** - Divisible Load Applications - H.264/AVC Video Encoding (inter-prediction mode)	• **Node: CPU+GPU platform**	• Load Balancing
• General (FP) Applications	• Device: multicore CPUs	• Performance modeling

Performance modeling

Divisible Load Processing

- **Discretely Divisible Load (DDL) Applications**
  - Computations divisible into pieces of arbitrary sizes (integers)
  - Fractions independently processed in parallel with no precedence constraints

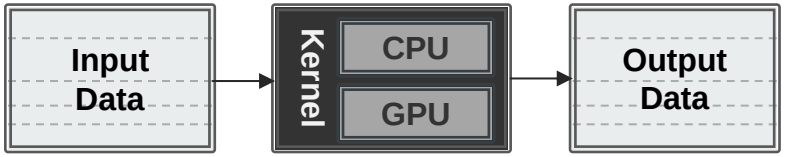


- **Applicable to a wide range of scientific problems**
  - Linear algebra, digital signal and image processing, database applications …
- **State of the art approaches in Heterogeneous Distributed Computing**
  - Assume symmetric bandwidth and an one-port model for communication links
  - Limited memory: only input load size is considered; exceeding load simply redistributed
  - Computation/communication time is not always a linear/affine function of the #chunks
  - Single-level load balancing solutions

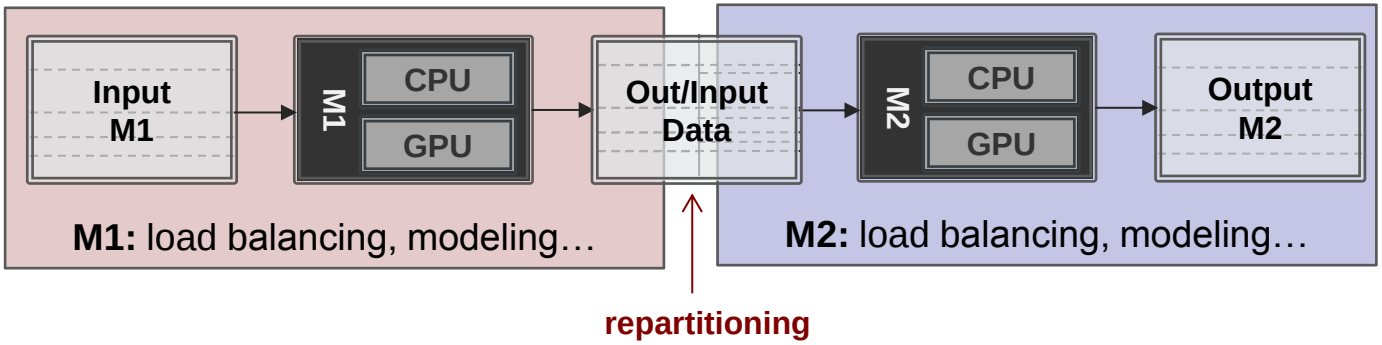
Divisible Load Processing

What? *Applications* | Where? *Systems and Devices* | How? *Modeling and Load Balancing*

- **Single-Module Applications**



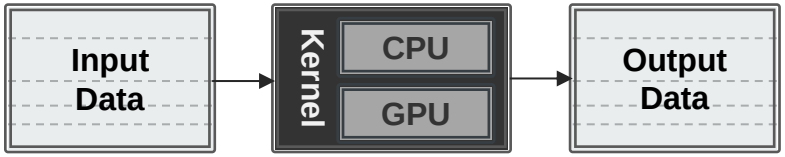
- **Multi-module Applications**



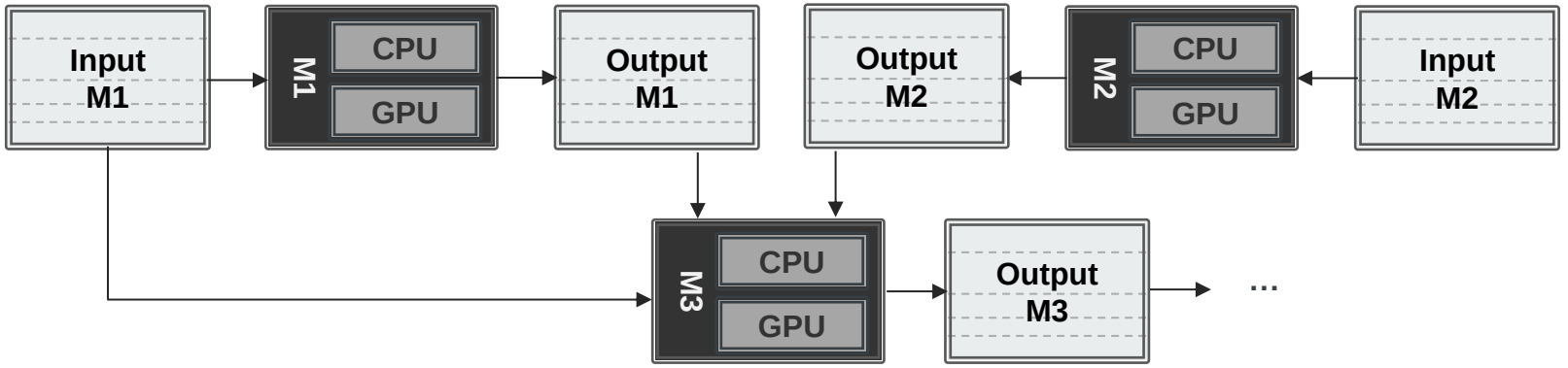
Divisible Load Processing

What? *Applications* | Where? *Systems and Devices* | How? *Modeling and Load Balancing*

- **Single-Module Applications**



- **Multi-module Applications**

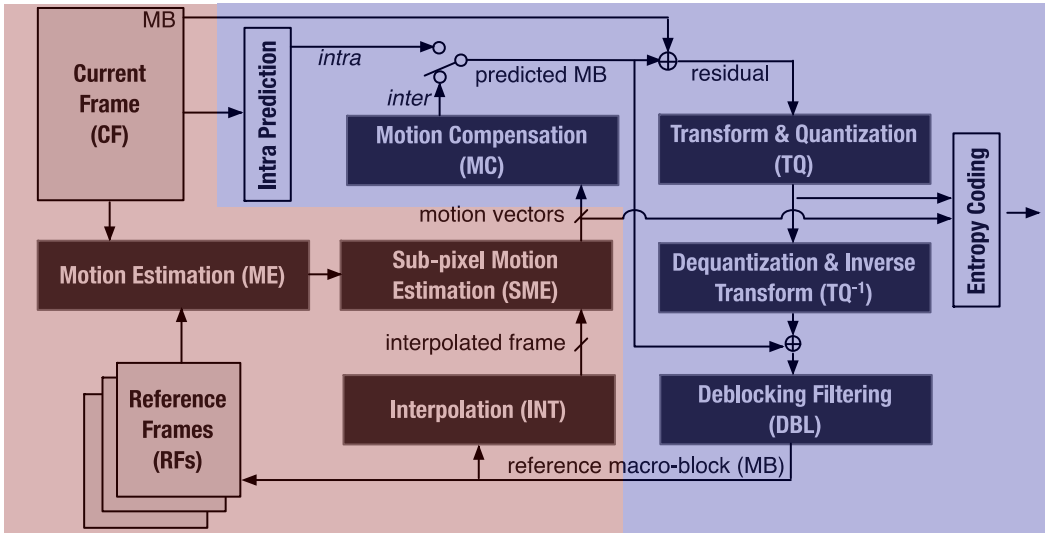


  - Data-dependencies, multiple input/output buffers, shared access to data buffers

Divisible Load Processing

What? *Applications* | Where? *Systems and Devices* | How? *Modeling and Load Balancing*

- **H.264/AVC Video Encoding**

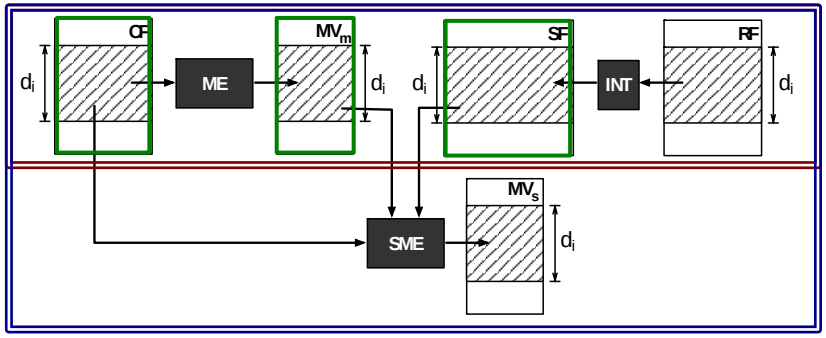


R* modules

- max. 6% on GPU (8.5% CPU)
- Dijkstra algorithm

ME+INT+SME

- min. 94% on GPU (92% CPU)
- Load balancing and modeling



- **Adaptive real-time** video encoding for **HD** sequences:
  - **Multi-module** load balancing
  - **Simultaneous** inter-prediction load balancing
  - **Communication minimization** (shared data buffers)

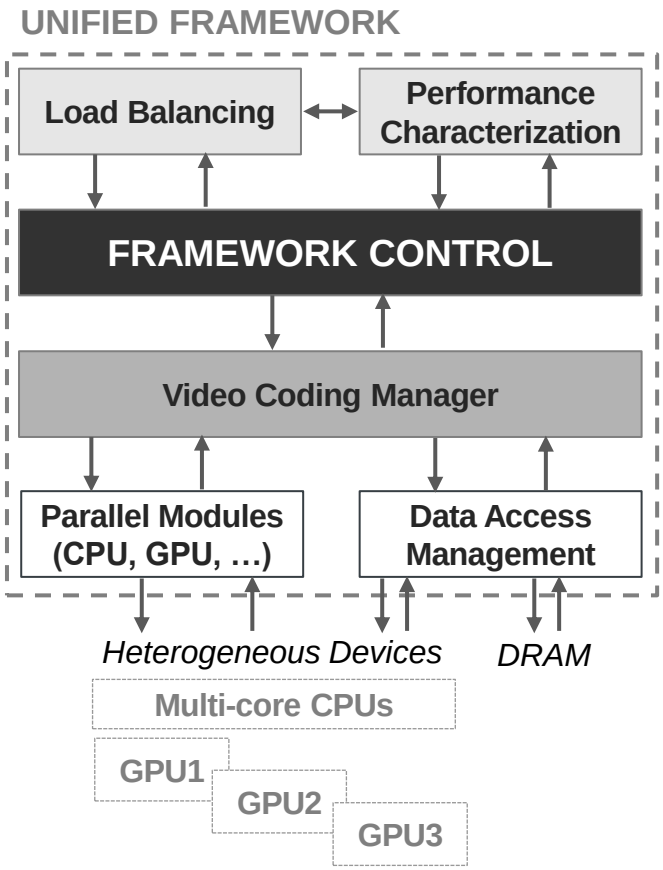
Outline

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*
• **Multi-module Applications** - Divisible Load Applications - H.264/AVC Video Encoding (inter-prediction mode)	• **Node: CPU+GPU platform**	• **Load Balancing** **- FEVES -** **F**ramework for **E**fficient parallel **V**ideo **E**ncoding on heterogeneous **S**ystems
• General (FP) Applications	• Device: multicore CPUs	• Performance modeling

Performance modeling

FEVES: General Layout

What? — *Applications* | Where? — *Systems and Devices* | **How?** — *Modeling and Load Balancing*



- **FEVES: Unified CPU+GPU encoding framework**
  - for collaborative inter-loop video encoding (extendable)
  - organized in several functional blocks

➢ **Framework control** provides the key functionality

- interacts with other blocks

➢ **Video Coding Manager** orchestrates collaborative execution

- invokes respective implementations of **Parallel Modules**
- automatic **Data Access Management** between DRAM and local memories

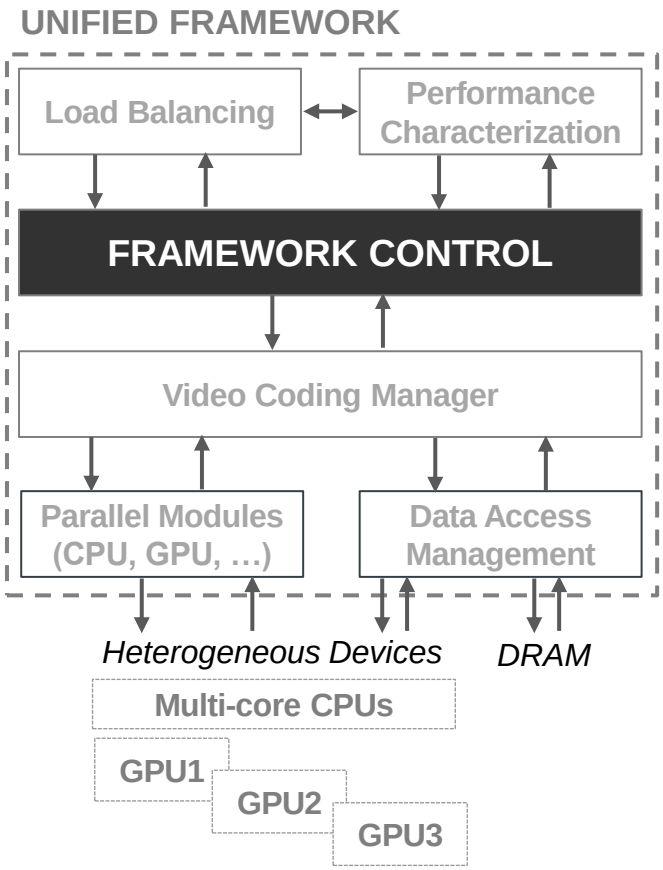
➢ **Load Balancing** with online **Performance Characterization**

- provides multi-module workload distributions for collaborative processing

** A. Ilic, S. Momcilovic, N. Roma and L. Sousa., "FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems", in **ICPP-2014***

FEVES: Framework Control

What? — Applications | Where? — Systems and Devices | **How?** — *Modeling and Load Balancing*



- **Framework Control**

Initialization

① **Detect** available devices (number, type, capabilities)
② Instantiate respective **Parallel Modules** (CPU+GPU)
③ Configure **Video Coding and Data Access Manager**
④ Equidistant **partitioning** for ME, INT and SME
⑤ Execute and **record** execution/transfer time
⑥ Initial **Performance characterization** for each device/module speeds and asymmetric bandwidth of PCIe links

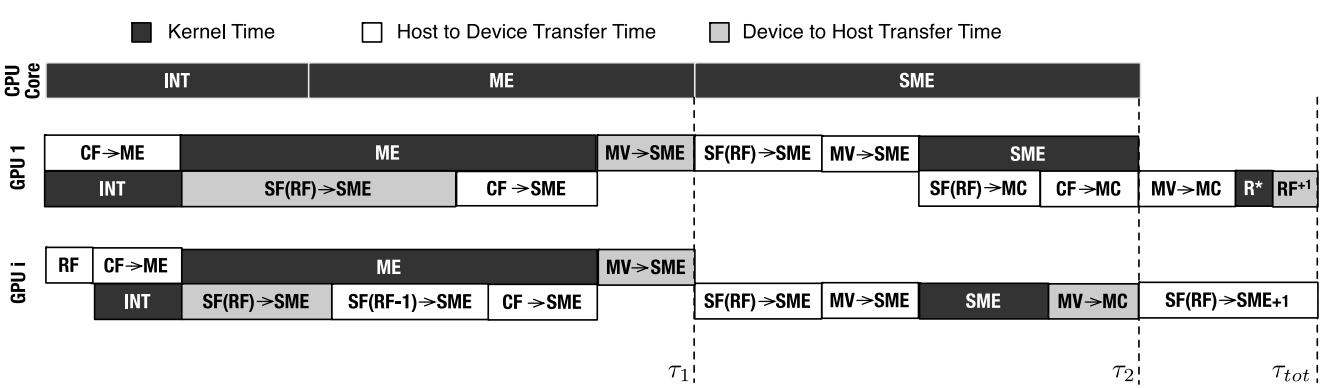
Iterative phase

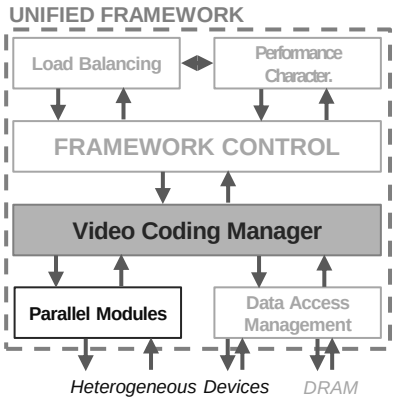
for each *frame* **do**
① Determine load distributions with **Load Balancing** based on **Performance Characterization**
② Execute modules with **Video Coding Manager, Data Access Management** and **Parallel Modules**
③ Record execution and transfer times and update **Performance characterization**

** A. Ilic, S. Momcilovic, N. Roma and L. Sousa., "FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems", in **ICPP-2014***

FEVES: Video Coding Manager

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*





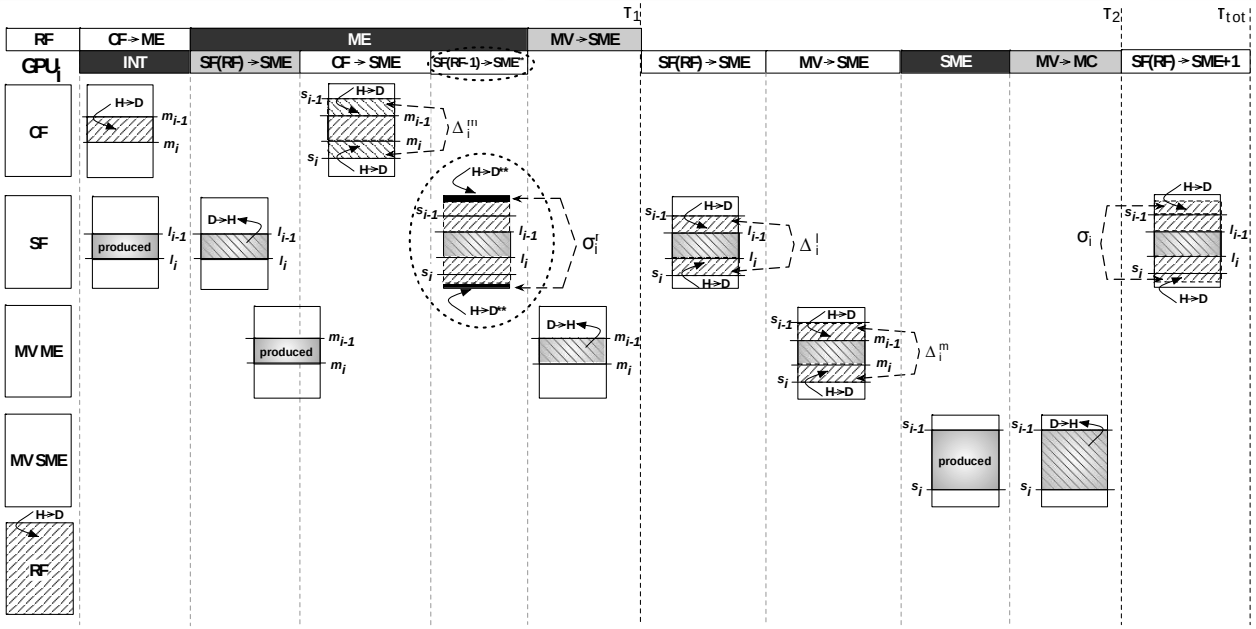
- **Video Coding Manager** orchestrates collaborative CPU+GPU video encoding
  - **automatically configured** according to detected device capabilities *(initialization phase)*, e.g., the amount of supported concurrency between computation and communication for GPU devices
  - invokes highly optimized CPU and GPU implementations for the **Library of Parallel Modules** (*SSE/AVX, Fermi/Kepler…*)
  - allows automatic **Data Access Management** between DRAM and local memories

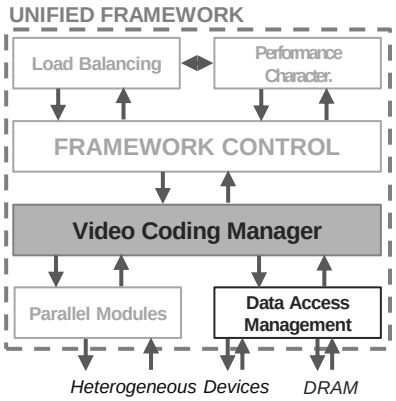
- **Collaborative Video Encoding** orchestration
  - Module **executions** and respective **data transfers** are invoked in a **predefined order** to ensure correctness of encoding
  - In respect to inherent data-dependencies in H.264/AVC encoding several **synchronization points** are defined:
    - t_1 – reflects the dependency of SME module on the outputs of ME and INT modules
    - t_2 – marks the completion of SME module and beginning of R* processing
    - t_{tot} – encoding of a current frame is completed (R* modules executed on single fastest device, e.g., GPU1)

** A. Ilic, S. Momcilovic, N. Roma and L. Sousa., "FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems", in **ICPP-2014***

FEVES: Data Access Management

What? *Applications* → **Where?** *Systems and Devices* → **How?** *Modeling and Load Balancing*





- **Data Access Management** for automatic data transfers and device memory management
  - functionality strictly depends on the decisions from the **Load Balancing** block (load distributions)
  - simultaneously tracks the state of **several input/output buffers**:
    - current frame (CF), interpolated sub-frame (SF), motion vectors from ME (MV ME) and SME (MV SME), reference frame (RF)
  - determines on the **size** of data transfers, their **order**, and exact **position** within the respective buffer
  - provides **communication minimization** when several modules access to the same shared buffer

** A. Ilic, S. Momcilovic, N. Roma and L. Sousa., "FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems", in **ICPP-2014***

FEVES: Load Balancing

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

Input: $N, n_w, n_c, T_1^{R*}, K_i^m, K_i^l, K_i^s$

Input: $K_1^{rfdh}, K_i^{cfhd}, K_i^{rfhd}, K_i^{sfhd}, K_i^{sfdh}, K_i^{mvhd}, K_i^{mvdh}, \sigma_i^{r-1}$

Output: $m=\{m_i\}, l=\{l_i\}, s=\{s_i\}, \sigma=\{\sigma_i\}, \sigma^r=\{\sigma_i^r\}$

Objective: minimize τ_{tot}

Performance Characterization (updated at runtime)

$$\sum_{i=1}^{n_w+n_c} m_i = N, \quad \sum_{i=1}^{n_w+n_c} l_i = N, \quad \sum_{i=1}^{n_w+n_c} s_i = N \qquad (1)$$

CPU Core

$$\forall i \in \{n_w+1, \ldots, n_w+n_c\}:$$

$$l_i K_i^l + m_i K_i^m \le \tau_1 \qquad (2)$$

$$\tau_1 + s_i K_i^s \le \tau_2 \qquad (3)$$

GPU_1 — performs R* modules

$$m_1 K_1^{cfhd} + m_1 K_1^m + m_1 K_1^{mvdh} \le \tau_1 \qquad (4)$$

$$l_1 K_1^l + l_1 K_1^{sfdh} + \Delta_1^m K_1^{cfhd} + m_1 K_1^{mvdh} \le \tau_1 \qquad (5)$$

$$m_1 K_1^{cfhd} + l_1 K_1^{sfdh} + \Delta_1^m K_1^{cfhd} + m_1 K_1^{mvdh} \le \tau_1 \qquad (6)$$

$$\tau_1 + \Delta_1^l K_1^{sfhd} + \Delta_1^m K_1^{mvhd} + s_1 K_1^s \le \tau_2 \qquad (7)$$

$$\tau_1 + \Delta_1^l K_1^{sfhd} + \Delta_1^m K_1^{mvhd} + (N - l_1 - \Delta_1^l) K_1^{sfhd} + (N - m_1 - \Delta_1^m) K_1^{cfhd} \le \tau_2 \qquad (8)$$

$$\tau_2 + (N - s_1) K_1^{mvhd} + T_1^{R*} + N K_1^{rfdh} \le \tau_{tot} \qquad (9)$$

GPU_i — accelerator/ distant worker

$$\forall i \in \{2, \ldots, n_w\}:$$

$$N K_i^{rfhd} + m_i K_i^{cfhd} + m_i K_i^m + m_i K_i^{mvdh} \le \tau_1 \qquad (10)$$

$$N K_i^{rfhd} + l_i K_i^l + l_i K_i^{sfdh} + \sigma_i^{r-1} K_i^{sfhd} + \Delta_i^m K_i^{cfhd} + m_i K_i^{mvdh} \le \tau_1 \qquad (11)$$

$$N K_i^{rfhd} + m_i K_i^{cfhd} + l_i K_i^{sfdh} + \sigma_i^{r-1} K_i^{sfhd} + \Delta_i^m K_i^{cfhd} + m_i K_i^{mvdh} \le \tau_1 \qquad (12)$$

$$\tau_1 + \Delta_i^l K_i^{sfhd} + \Delta_i^m K_i^{mvhd} + s_i K_i^s + s_i K_i^{mvdh} \le \tau_2 \qquad (13)$$

$$\sigma_i = \text{MIN}(N - l_i - \Delta_i^l, (\tau_{tot} - \tau_2)/K_i^{sfhd}) \qquad (14)$$

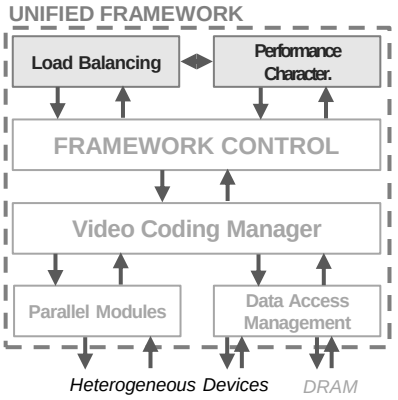
$$\sigma_i^r = N - l_i - \Delta_i^l - \sigma_i \qquad (15)$$

communication minimization

$$\forall i \in \{1, \ldots, n_w\}:$$

$$\Delta_i^m = \text{MS\_BOUNDS}(m, s) \qquad (16)$$

$$\Delta_i^l = \text{LS\_BOUNDS}(l, s) \qquad (17)$$



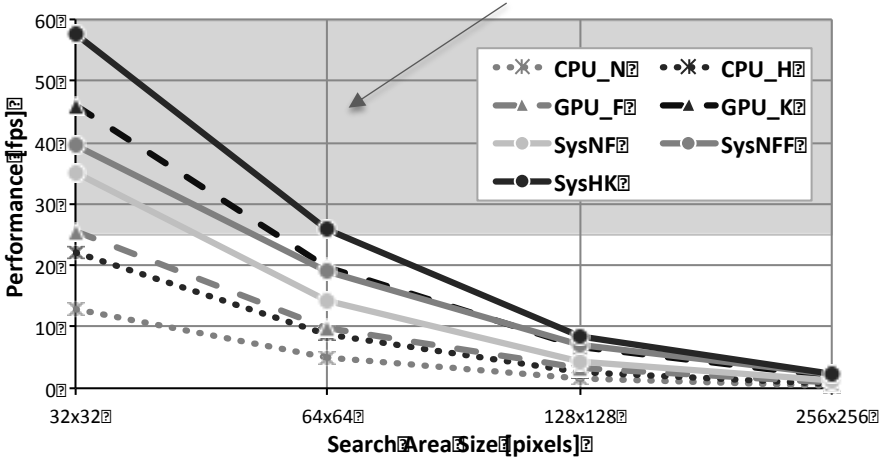
- **Load Balancing** based on *linear programming* to determine:
  - **cross-device load distributions** for ME, INT and SME modules
  - amount of data transfers across different devices for shared buffers
  - **communication minimization**
  - **minimizes total** collaborative CPU+GPU video encoding **time**

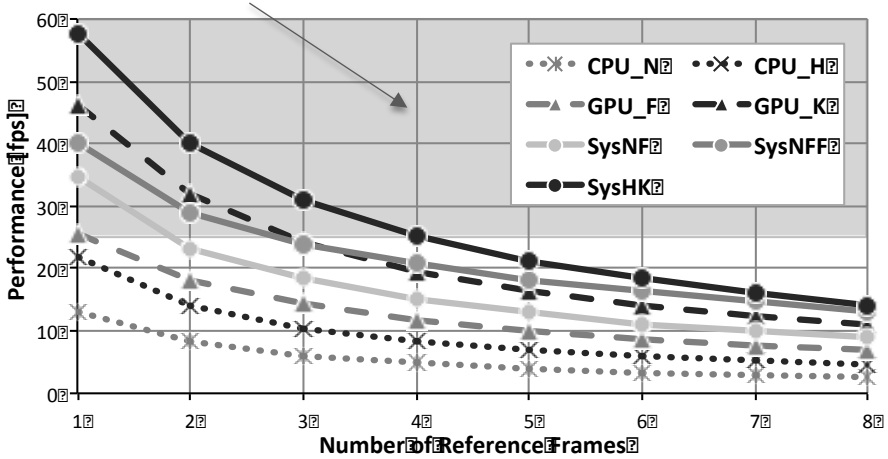
** A. Ilic, S. Momcilovic, N. Roma and L. Sousa., "FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems", in **ICPP-2014***

FEVES: Experimental results

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

Real-time video encoding for full HD (1080p) video sequences





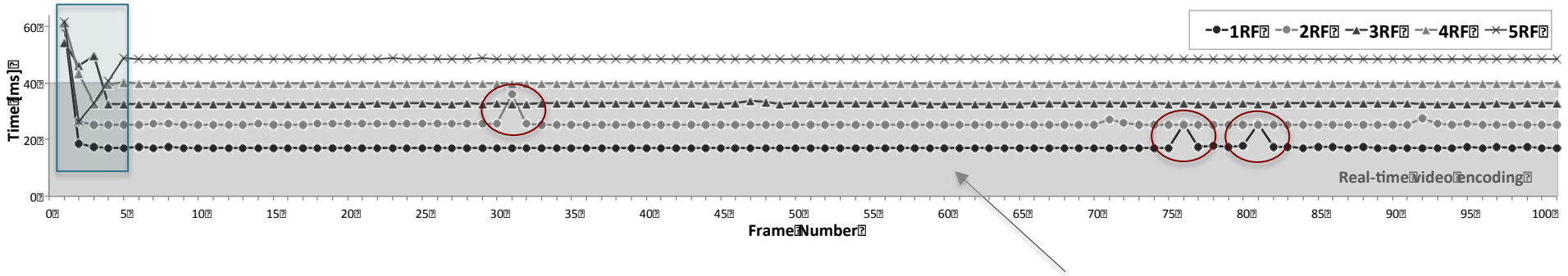
- **Scalable** over both search area (SA) size and the number of reference frames (RF)
- Highly optimized parallel modules (**CPU_H 1.7x faster** than **CPU_N**; **GPU_K 2x** than **GPU_F**)
- **Real-time encoding** on SysHK**: for 64x64 SA** size (1 RF) and **up to 4 RFs** for 32x32 SA
- Average **speedup** on SysNFF: **5x** vs. CPU_N and **2.2x** vs. GPU_F

Devices		Heterogeneous Systems	
CPU_N	Intel Nehalem i7 950	**SysNF**	CPU_N + GPU_F
CPU_H	Intel Haswell i7 4770K	**SysNFF**	CPU_N + 2xGPU_F
GPU_F	NVIDIA Fermi GTX580	**SysHK**	CPU_H + GPU_K
GPU_K	NVIDIA Kepler GTX780Ti		

** A. Ilic, S. Momcilovic, N. Roma and L. Sousa., "FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems", in **ICPP-2014***

FEVES: Experimental results

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*



Real-time video encoding for 1080p "Rolling Tomatoes" sequence (first 100 frames)

- **Real-time encoding** for **up to 4 RFs** for 32x32 SA on **SysHK (**Intel i7 4770K + NVIDIA GTX780Ti)
- **Load Balancing** capable of efficiently coping with **increasing problem complexity**
- Dynamic **Performance Characterization** allows **adaptation** to the current state of the platform

** A. Ilic, S. Momcilovic, N. Roma and L. Sousa., "FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems", in **ICPP-2014***

Outline

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*
• **Multi-module Applications** - Divisible Load Applications - H.264/AVC Video Encoding (inter-prediction mode)	• **Node: CPU+GPU platform**	• **Load Balancing** **- FEVES -** **F**ramework for **E**fficient parallel **V**ideo **E**ncoding on heterogeneous **S**ystems
• **General (FP) Applications**	• **Device: multicore CPUs**	• **Performance modeling**

Performance modeling

Outline

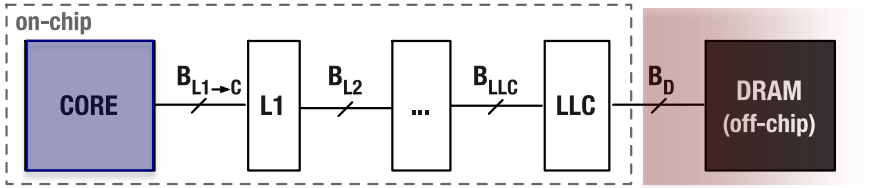
What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*
• **Multi-module Applications** - Divisible Load Applications - H.264/AVC Video Encoding (inter-prediction mode)	• **Node: CPU+GPU platform**	• **Load Balancing** **- FEVES -** **F**ramework for **E**fficient parallel **V**ideo **E**ncoding on heterogeneous **S**ystems
• **General (FP) Applications**	• **Device: multicore CPUs**	• **Cache-aware Roofline Model** - Performance and Total Performance

Performance modeling

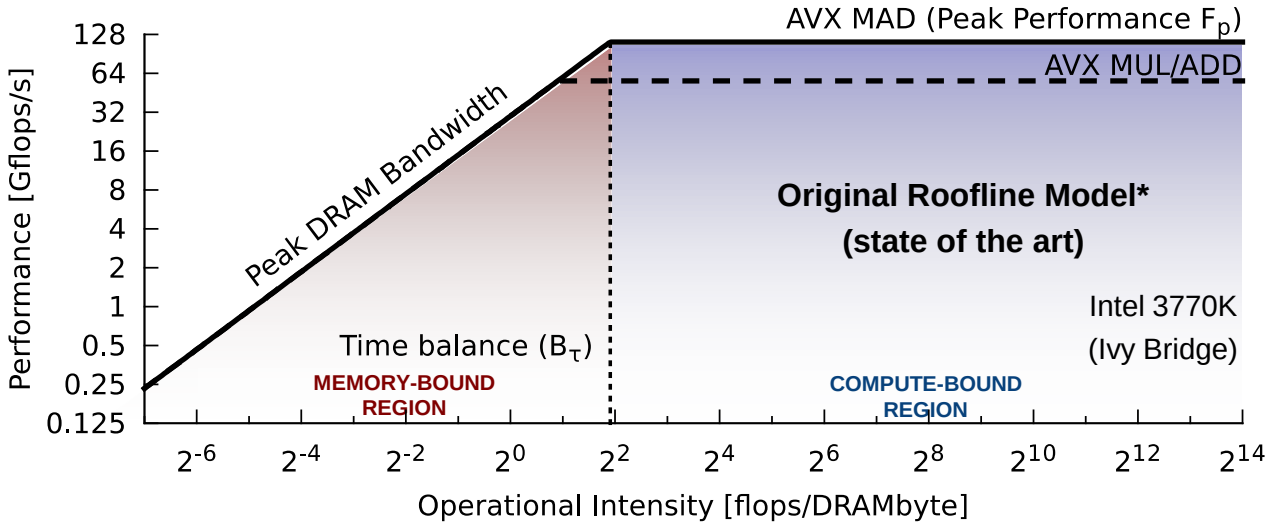
Original Roofline Model

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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



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

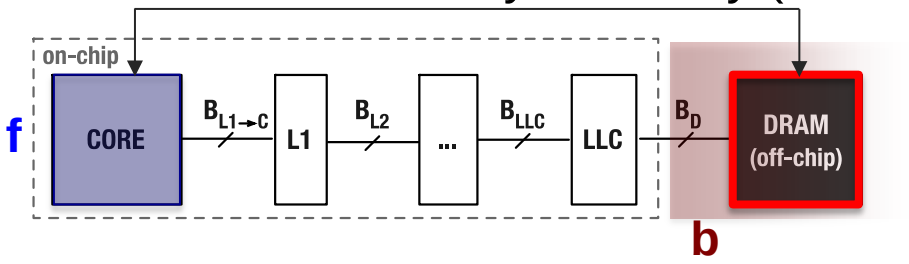


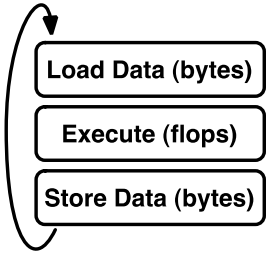
** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

Original Roofline Model: Hands On

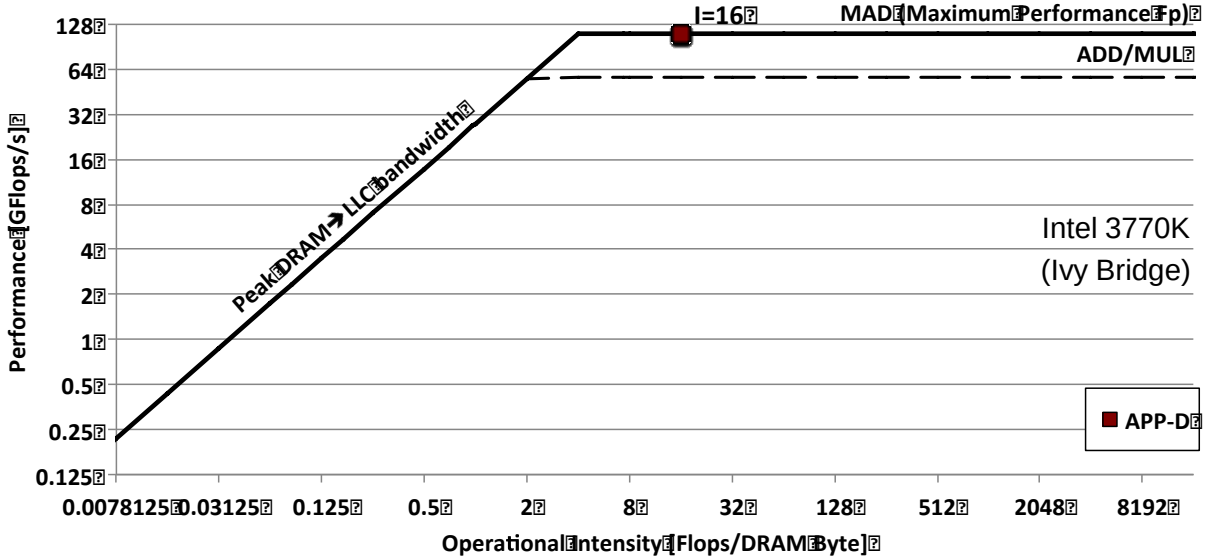
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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





APP-D (data traffic from DRAM)



$I=(\Sigma f_i)/(\Sigma b_i)$

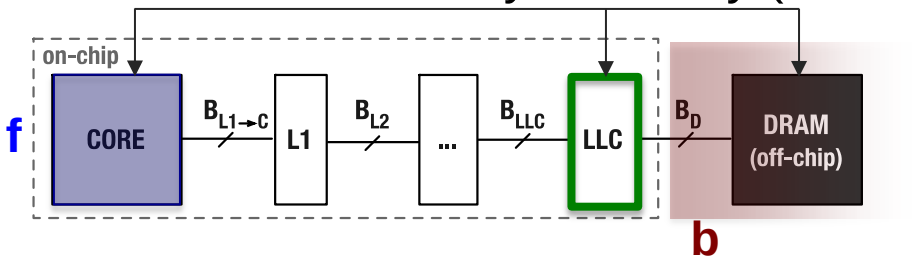
I is constant

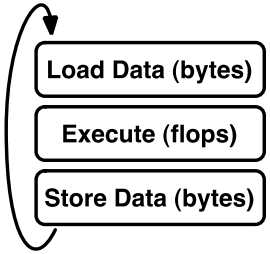
** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

Original Roofline Model: Hands On

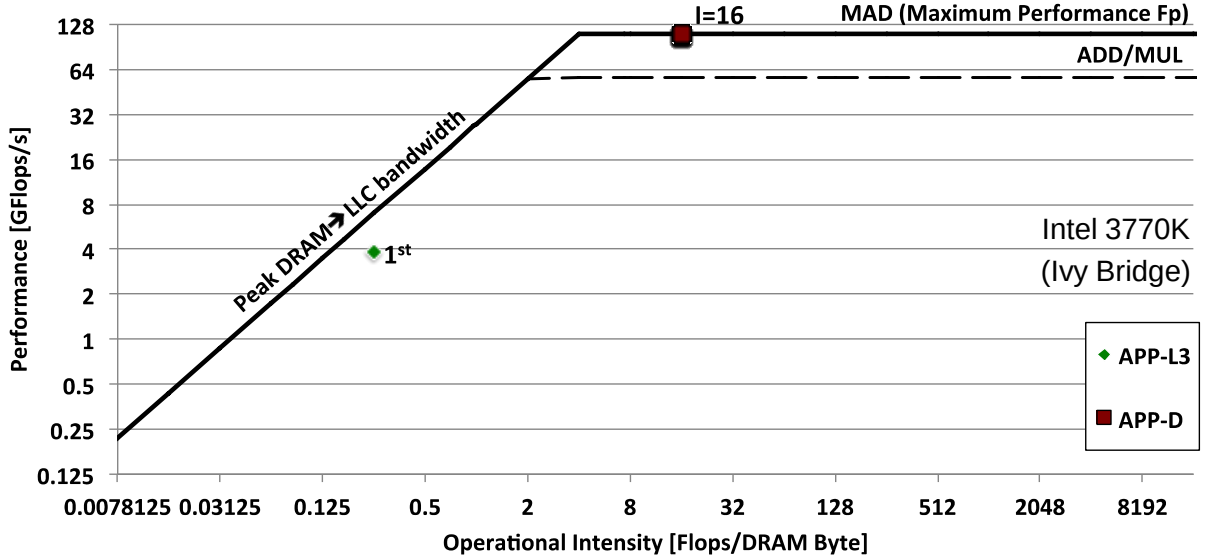
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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





APP-L3 (data fits in L3)



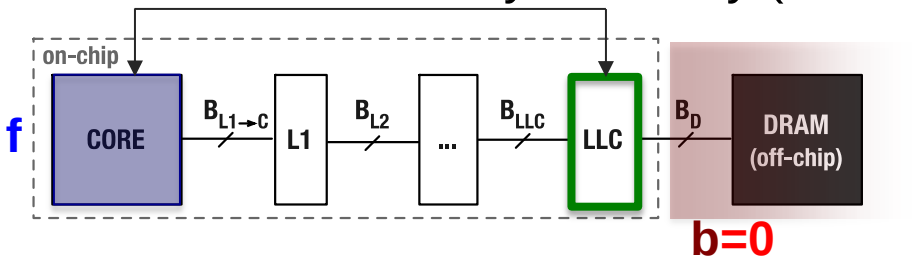
$I_1 = f_1 / b_1$

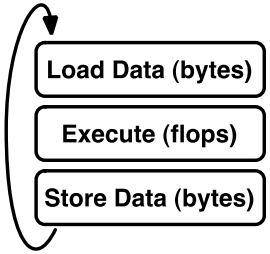
** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

Original Roofline Model: Hands On

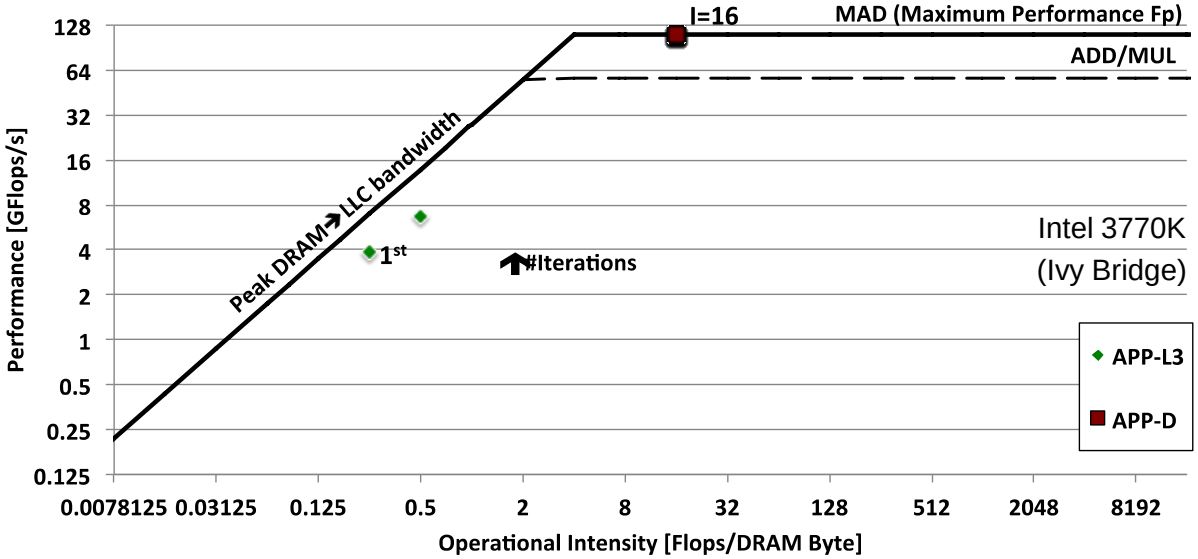
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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





APP-L3 (data fits in L3)



$I_1=f_1/b_1$

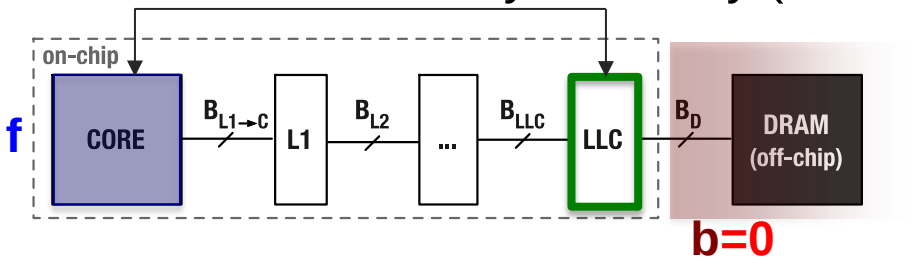
$I_2=(f_1+f_2)/b_1$

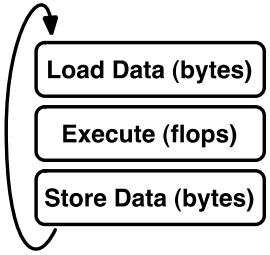
** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

Original Roofline Model: Hands On

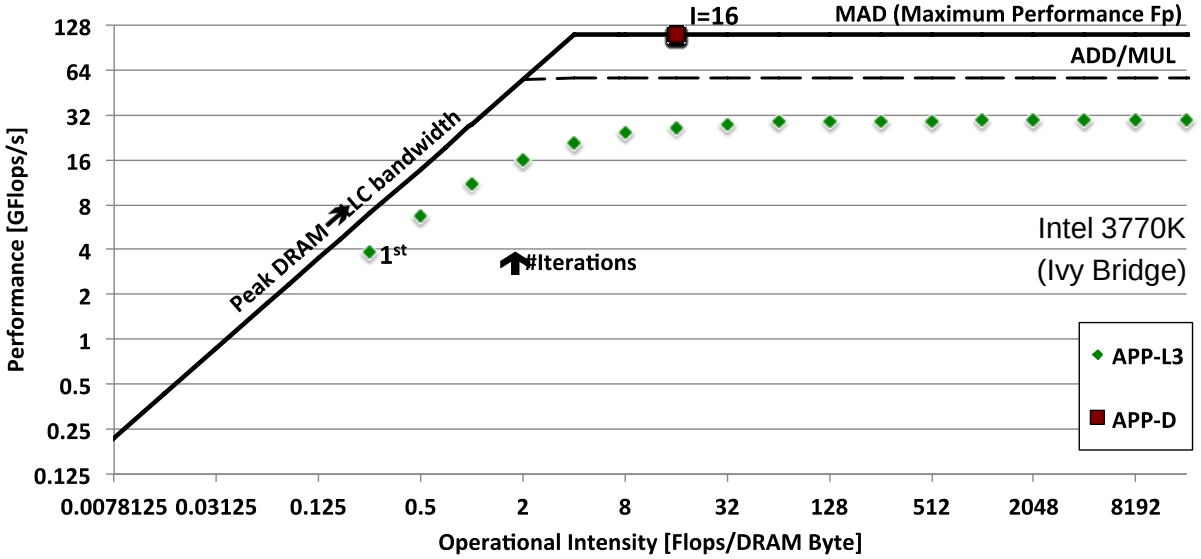
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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





APP-L3 (data fits in L3)



$I_1 = f_1/b_1$

$I_2 = (f_1+f_2)/b_1$

$I_i = (\Sigma f_i)/b_1$

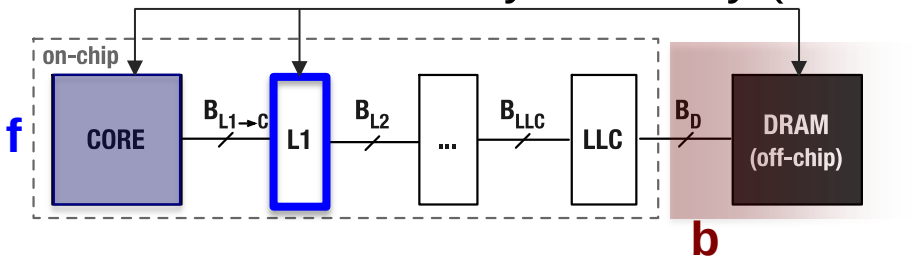
I is variable

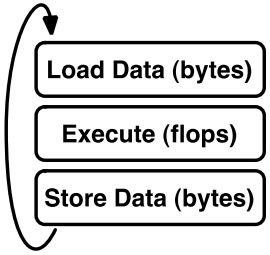
** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

Original Roofline Model: Hands On

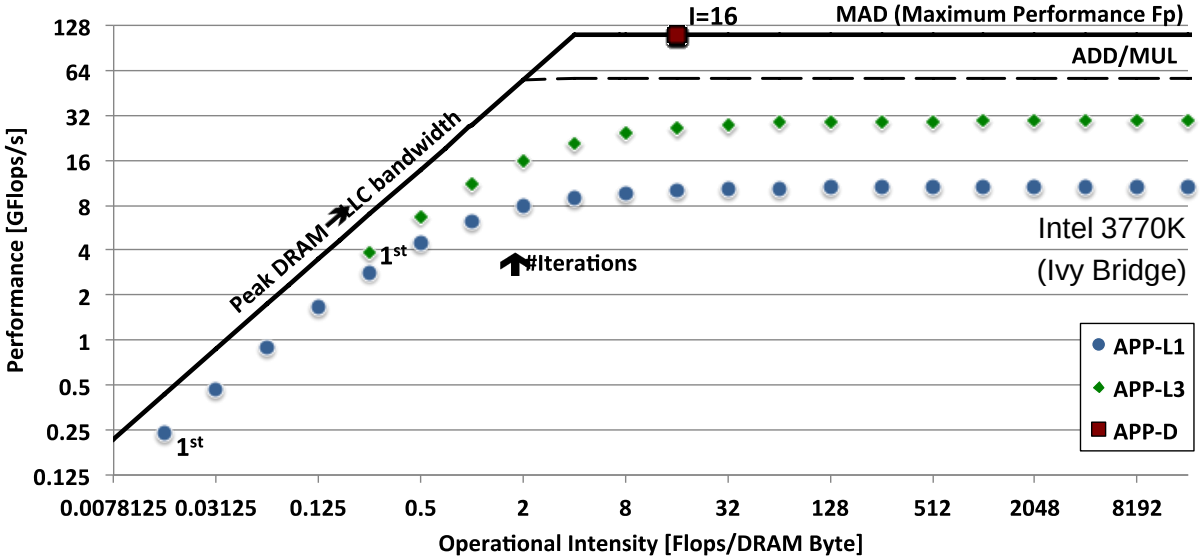
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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





APP-L1 (data fits in L1)



$I_1=f_1/b_1$

$I_2=(f_1+f_2)/b_1$

$I_i=(\Sigma f_i)/b_1$

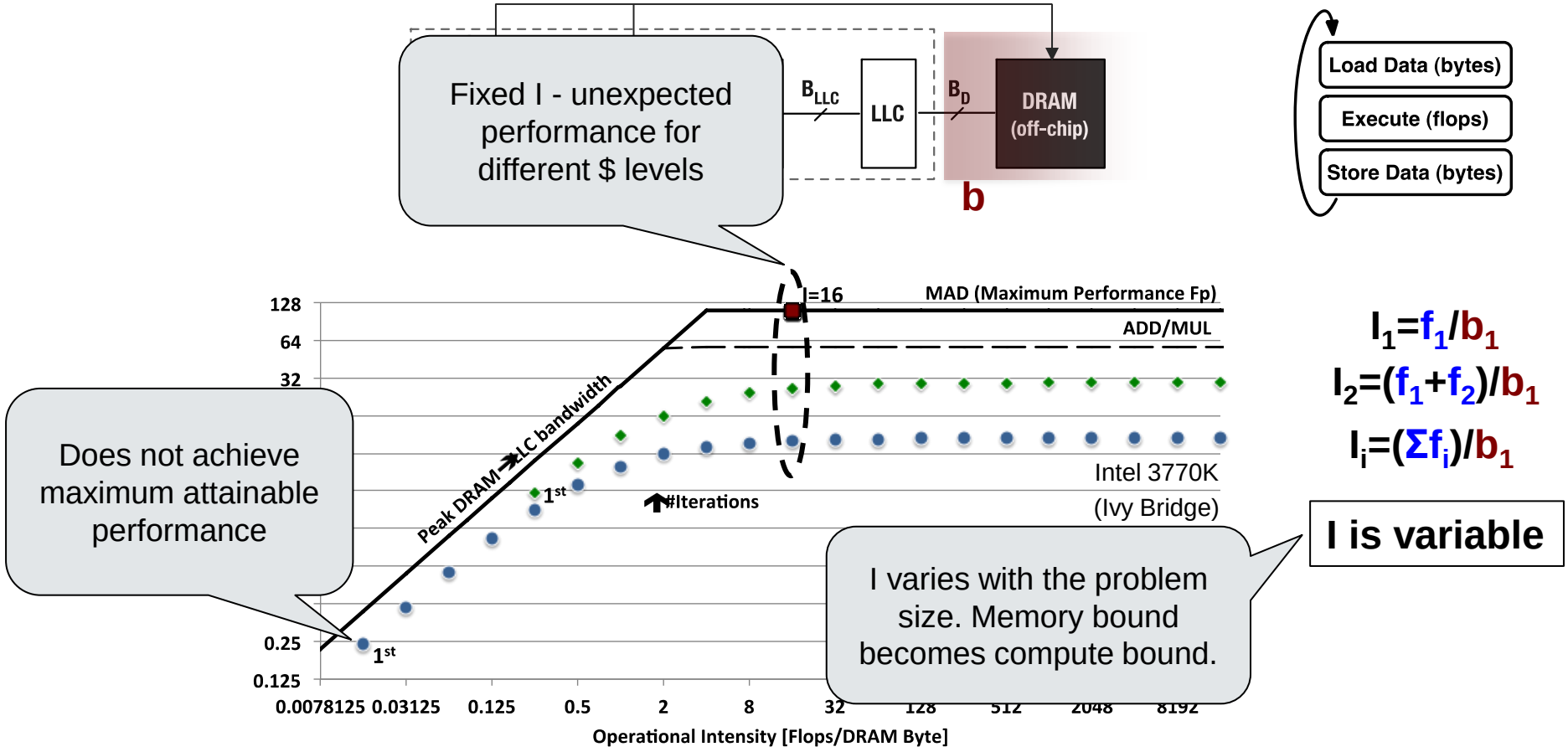
I is variable

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

Original Roofline Model: Hands On

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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

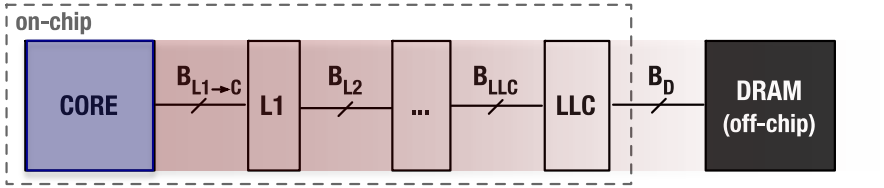


** Williams, S., Waterman, A. and Patterson, D., "Roofline: An insightful visual performance model for multicore architectures", Communications of the ACM (2009)*

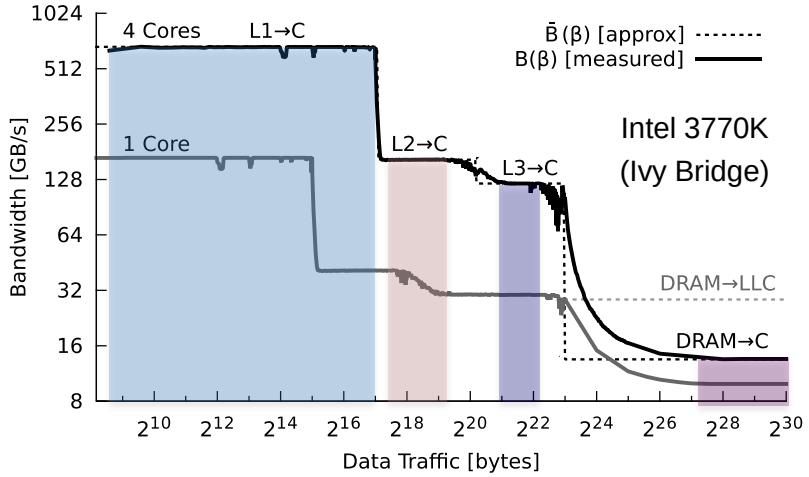
Cache-aware Roofline Model

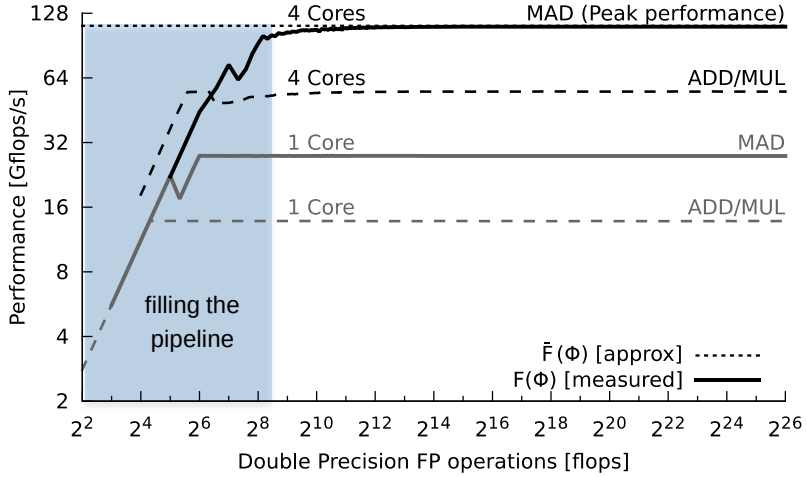
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

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



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

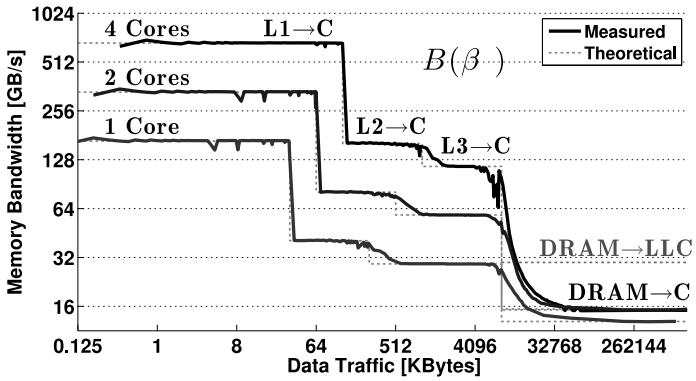




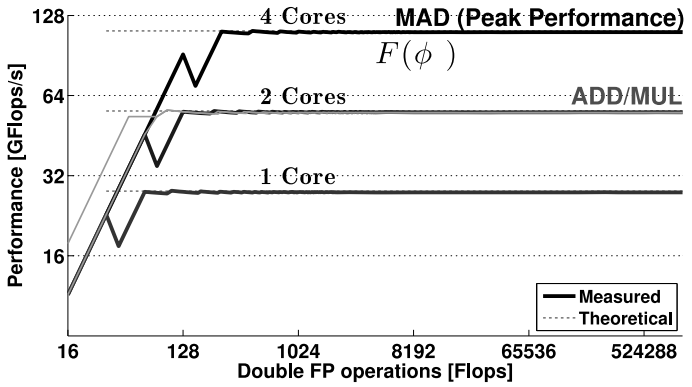
Cache-aware Roofline Model

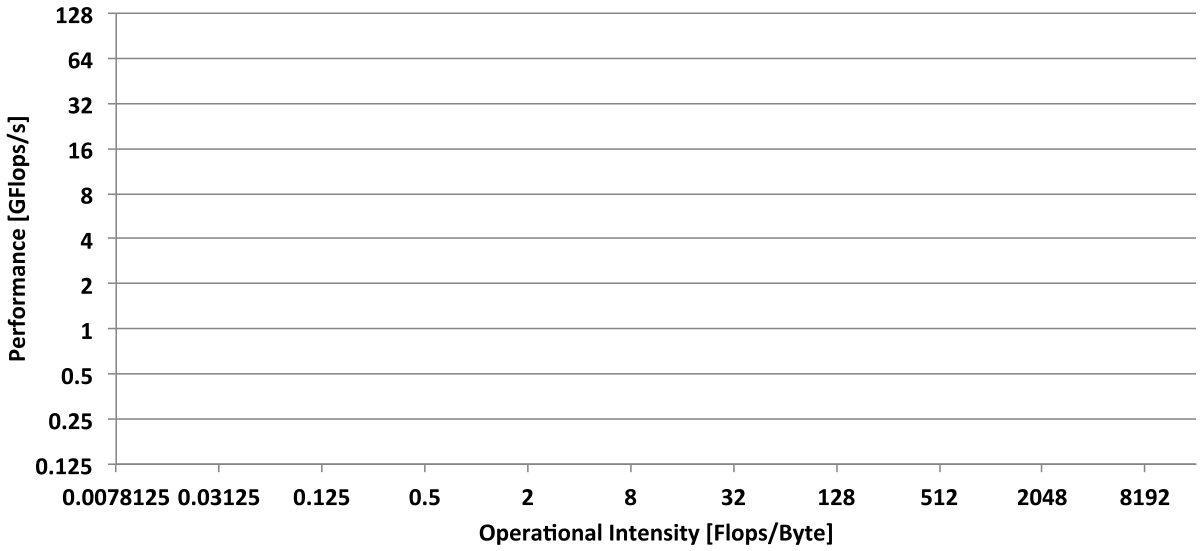
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

Memory bandwidth variation



Performance variation

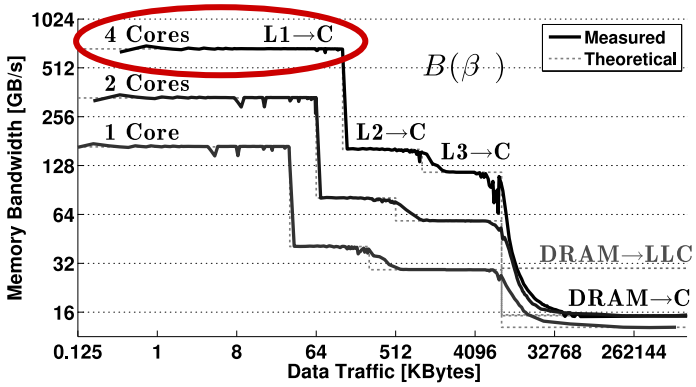




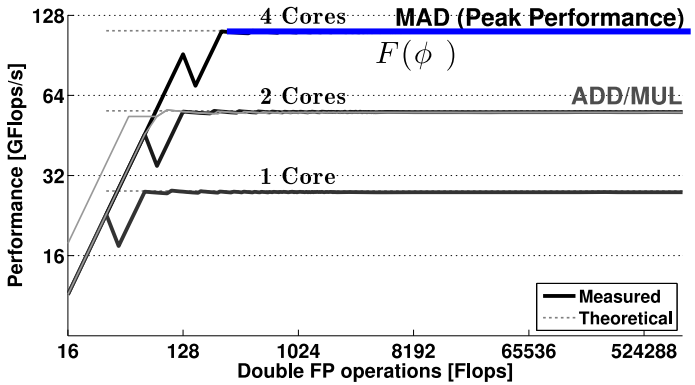
Cache-aware Roofline Model

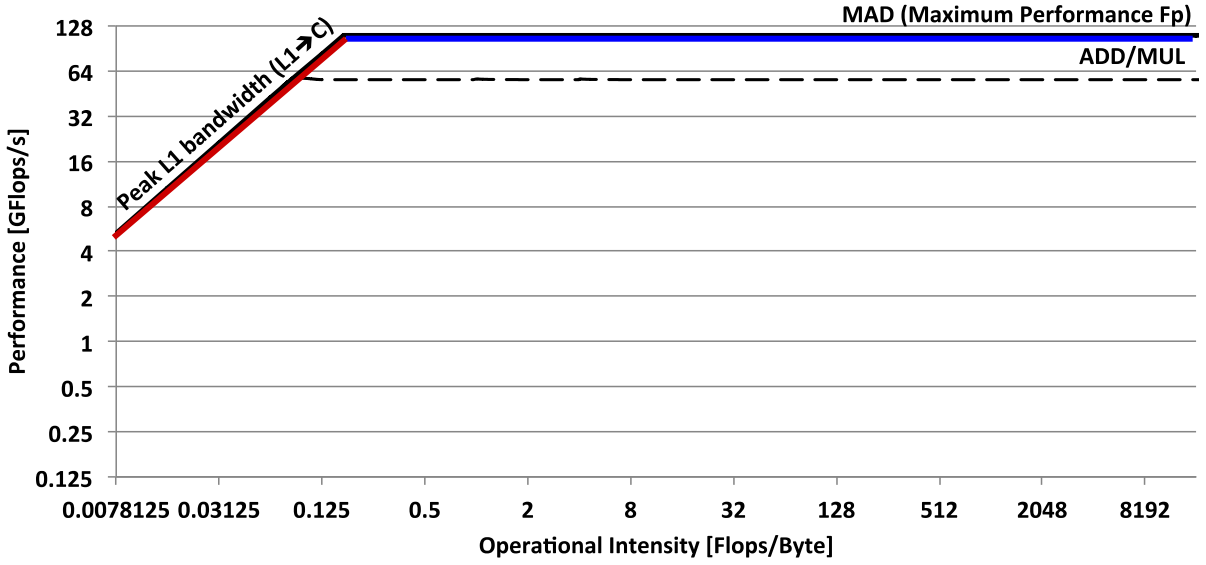
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

Memory bandwidth



Performance variation

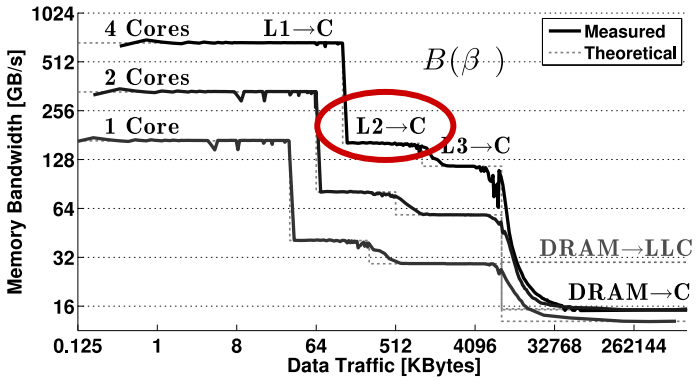




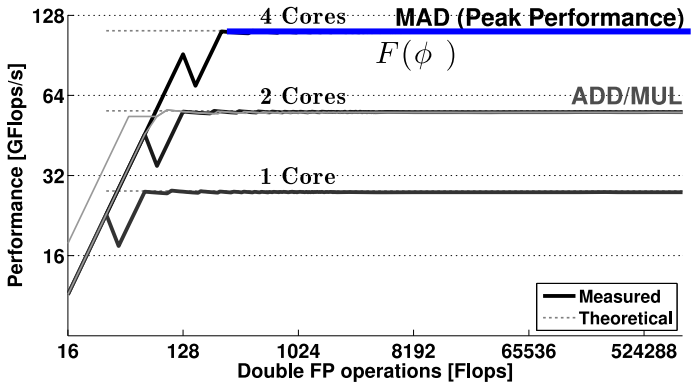
Cache-aware Roofline Model

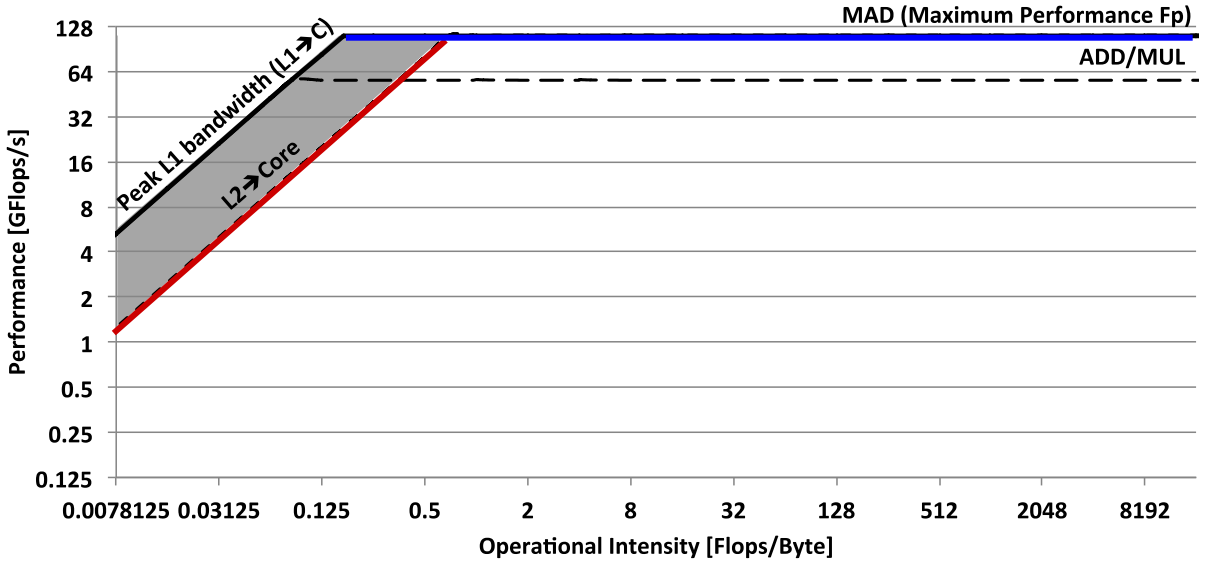
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

Memory bandwidth



Performance variation

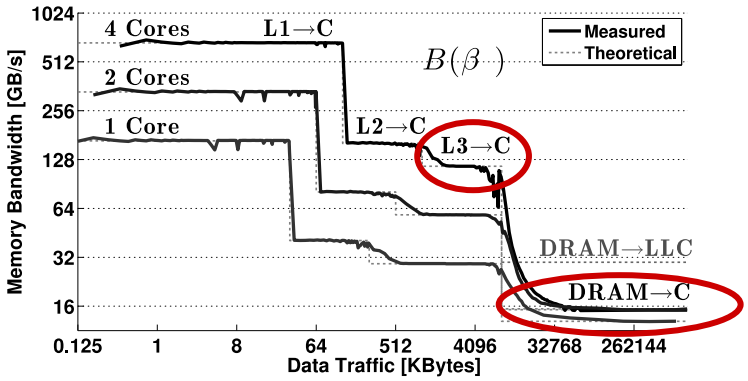




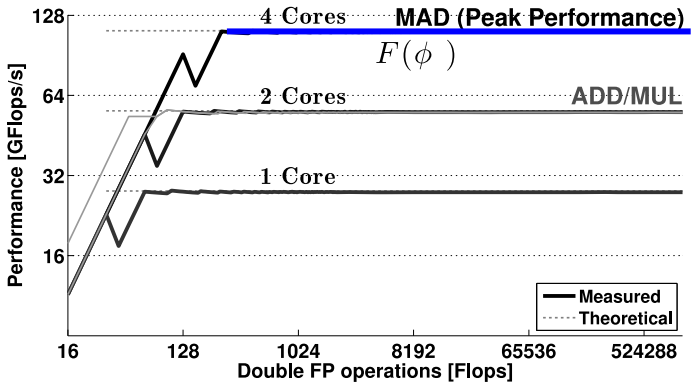
Cache-aware Roofline Model

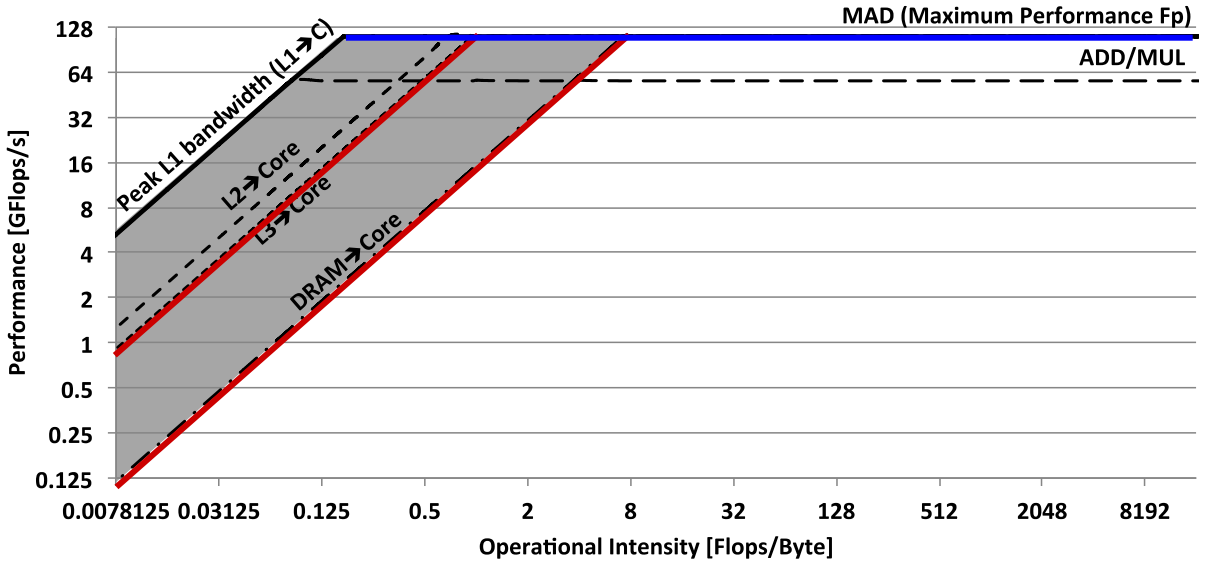
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

Memory bandwidth



Performance variation

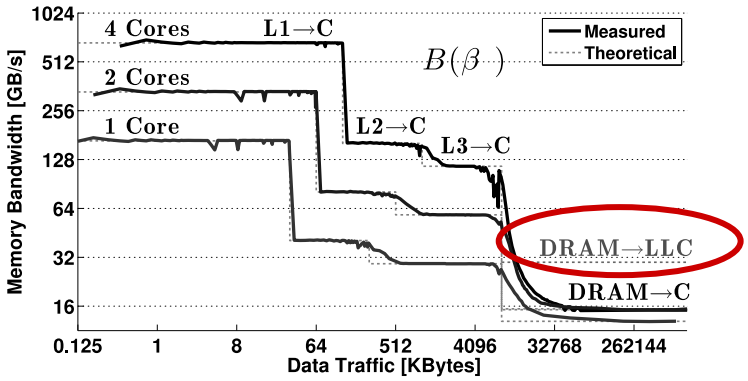




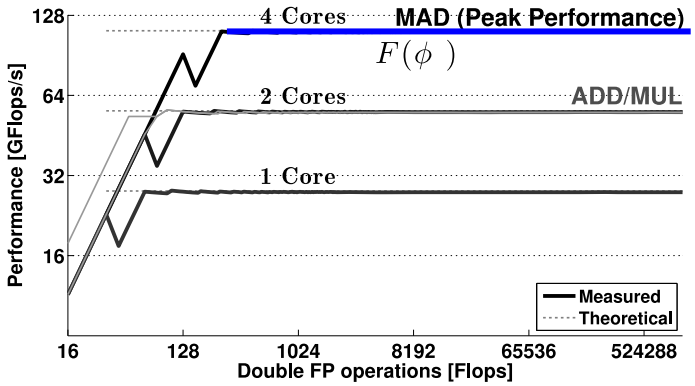
Cache-aware Roofline Model

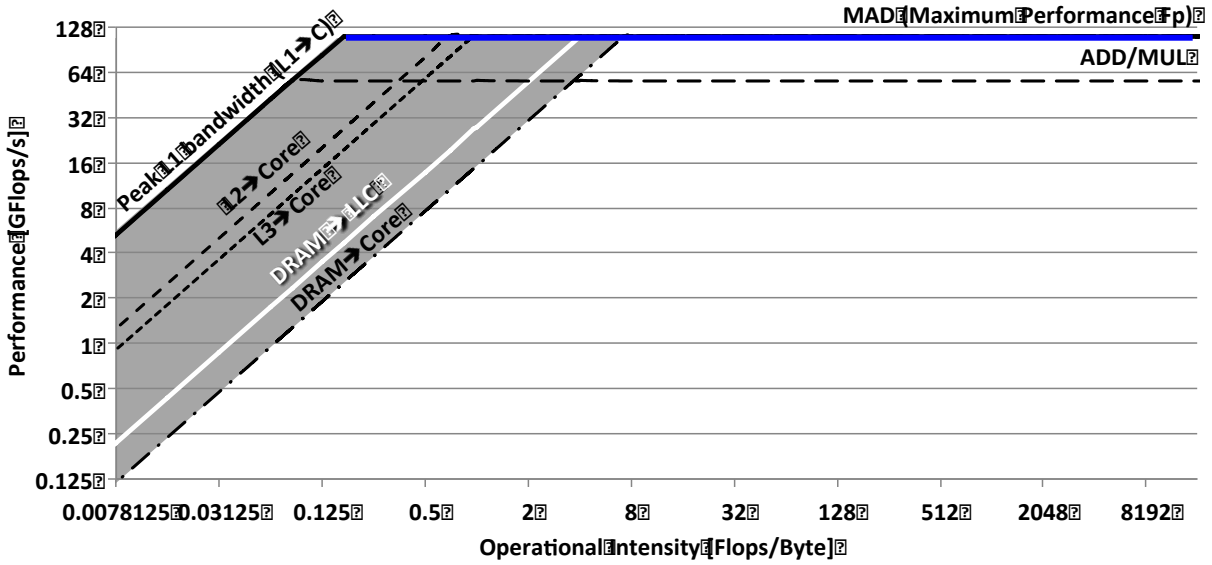
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

Memory bandwidth



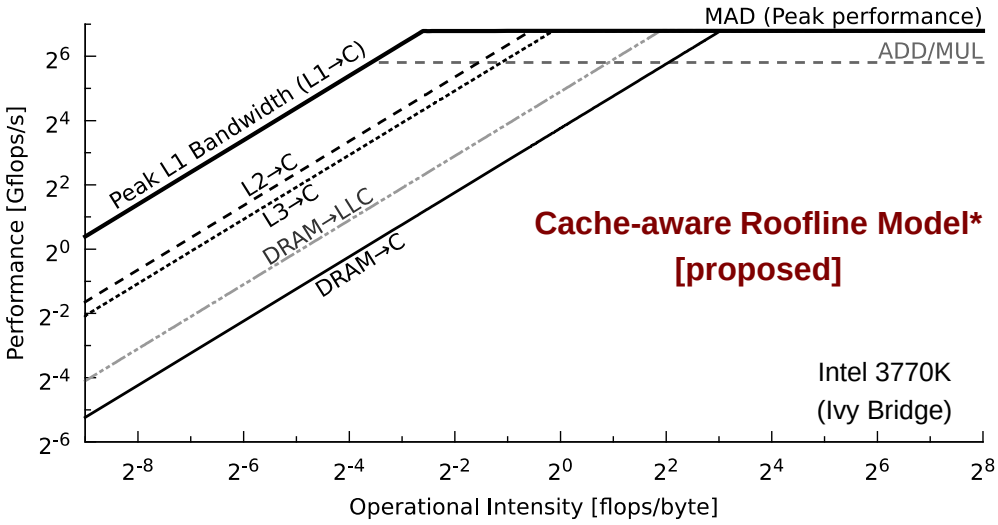
Performance variation





Cache-aware Roofline Model

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

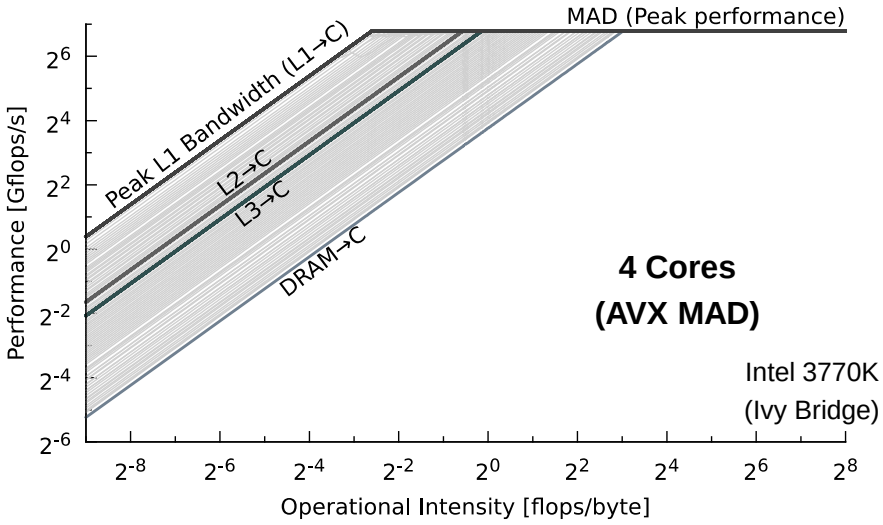


- Insightful **single plot model**
  - Shows performance limits of multicores
  - Redefined OI: flops and bytes as seen by core
  - Constructed once per architecture
- Considers **complete memory hierarchy**
  - Influence of caches and DRAM to performance
- Applicable to **other types of operations**
  - 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

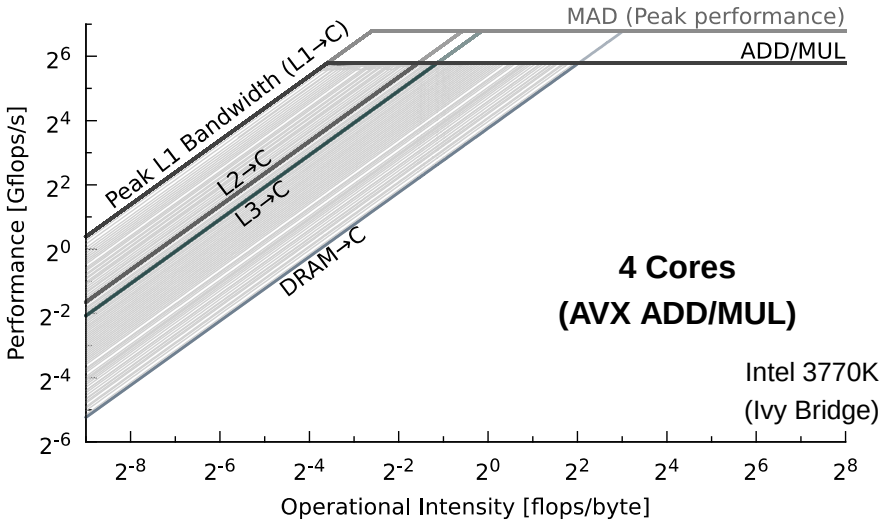
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*



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

Cache-aware Roofline Model

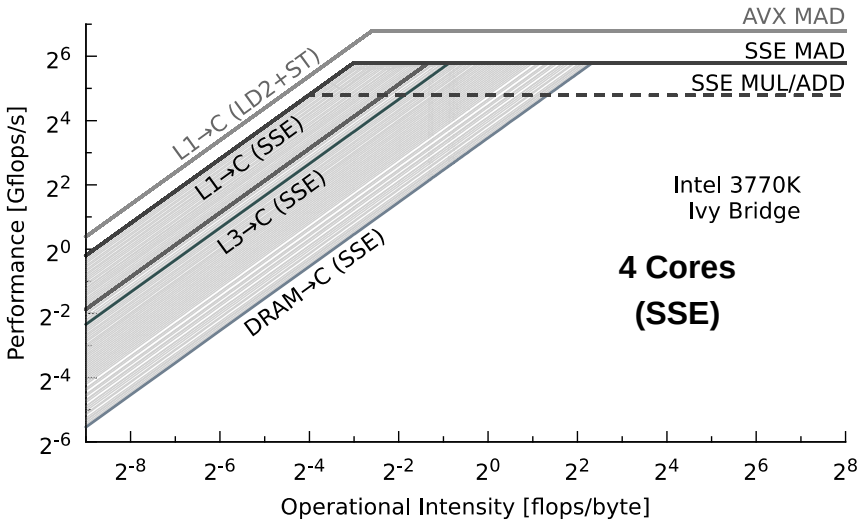
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*



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

Cache-aware Roofline Model

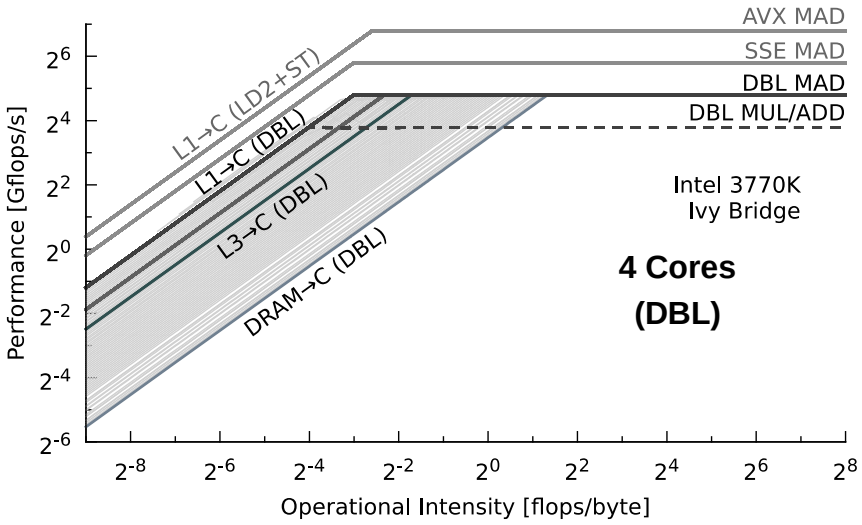
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*



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

Cache-aware Roofline Model

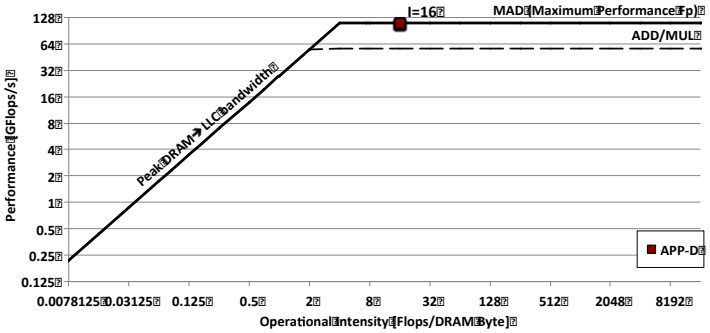
What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

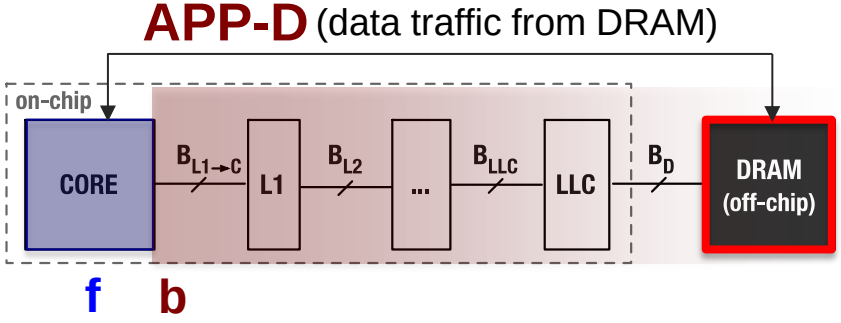


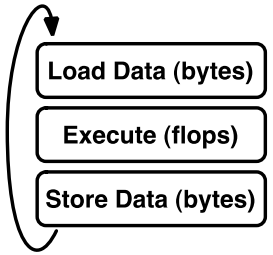
- Insightful **single plot model**
  - Shows performance limits of multicores
  - Redefined OI: flops and bytes as seen by core
  - Constructed once per architecture
- Considers **complete memory hierarchy**
  - Influence of caches and DRAM to performance
- Applicable to **other types of operations**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

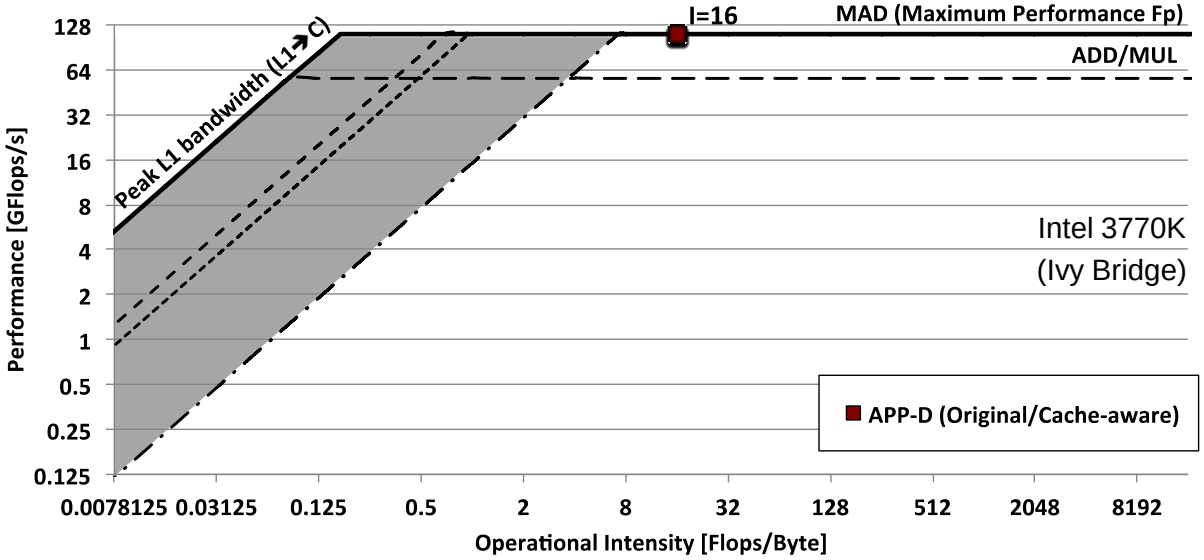
Cache-aware Roofline Model: Hands On

What? *Applications* → **Where?** *Systems and Devices* → **How?** *Modeling and Load Balancing*









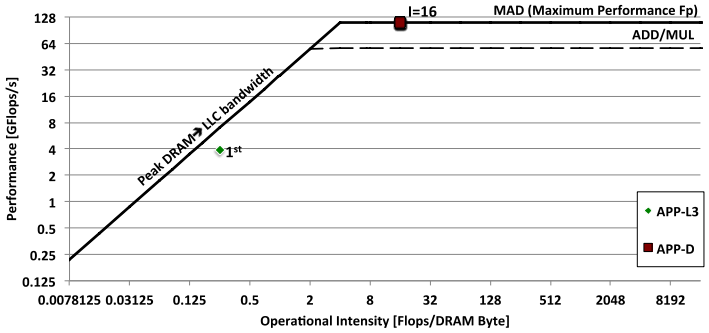
$I=(\Sigma f_i)/(\Sigma b_i)$

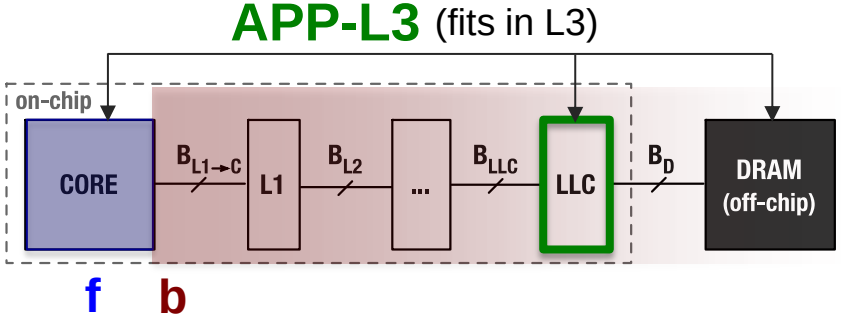
I is constant

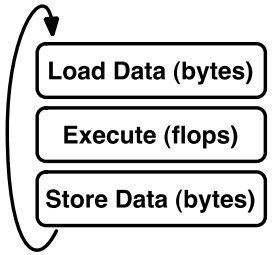
** Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

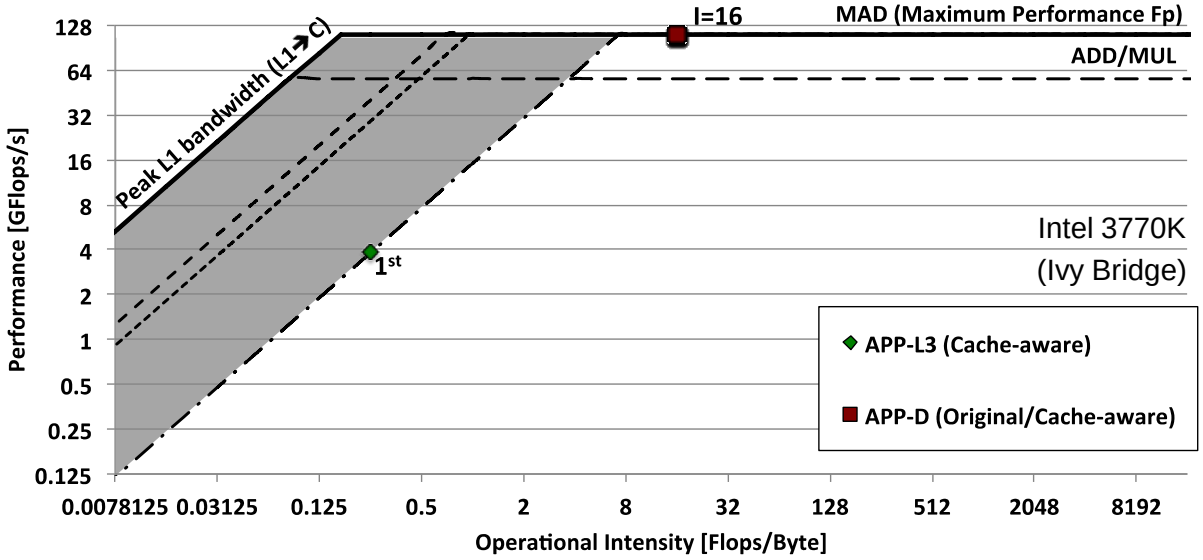
Cache-aware Roofline Model: Hands On

What? *Applications* → **Where?** *Systems and Devices* → **How?** *Modeling and Load Balancing*









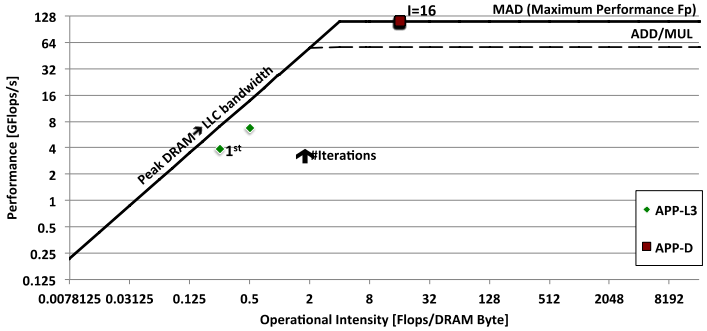
$I=(\Sigma f_i)/(\Sigma b_i)$

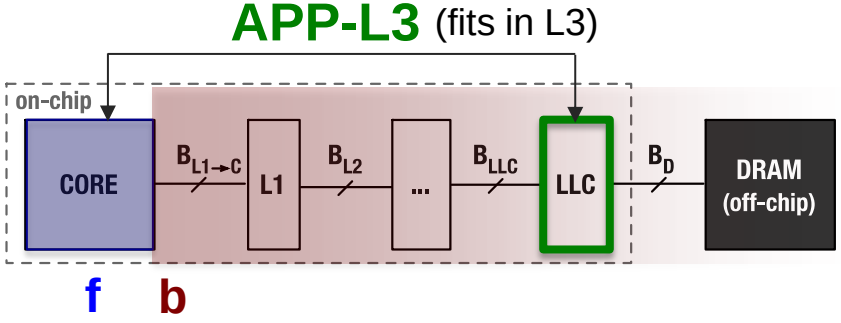
I is constant

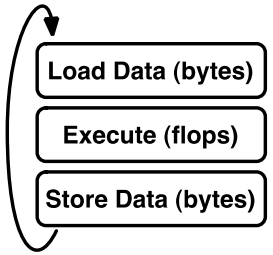
** Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

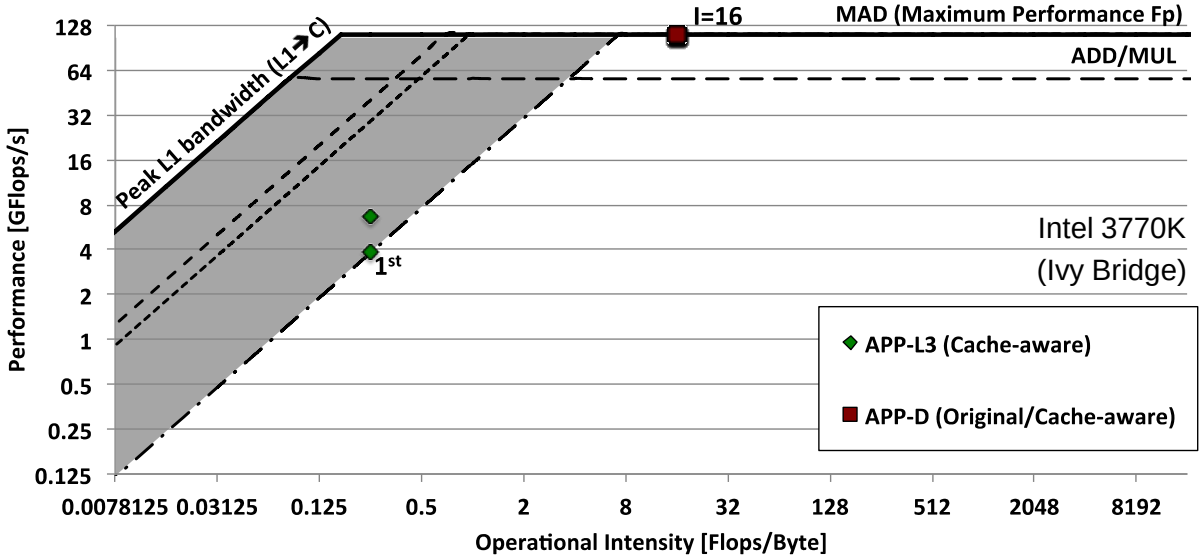
Cache-aware Roofline Model: Hands On

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*









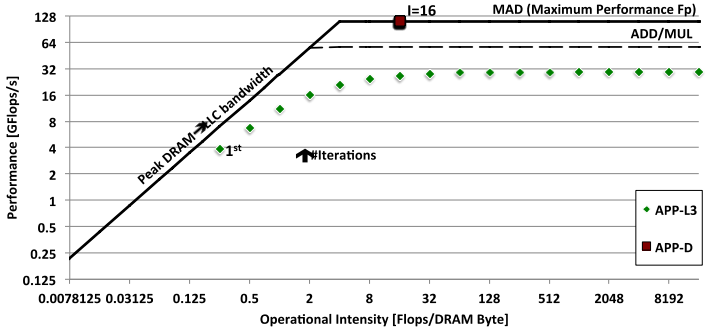
$I=(\Sigma f_i)/(\Sigma b_i)$

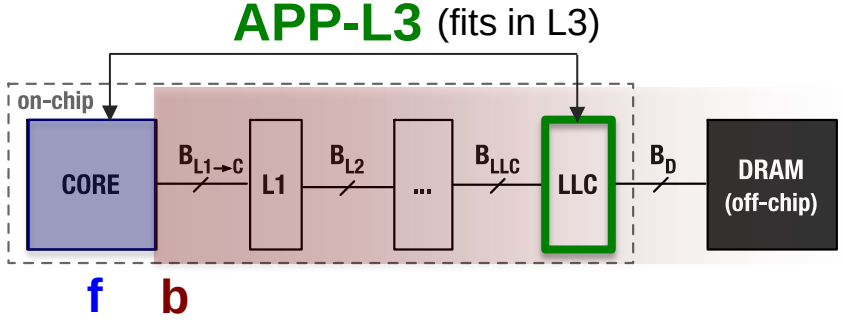
I is constant

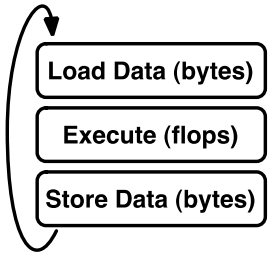
** Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

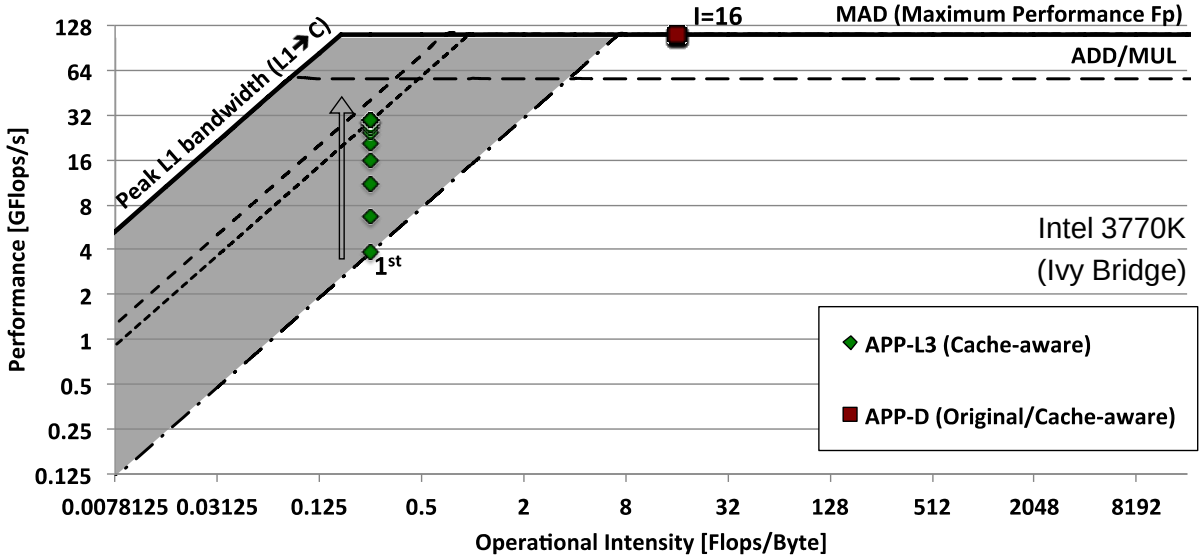
Cache-aware Roofline Model: Hands On

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*









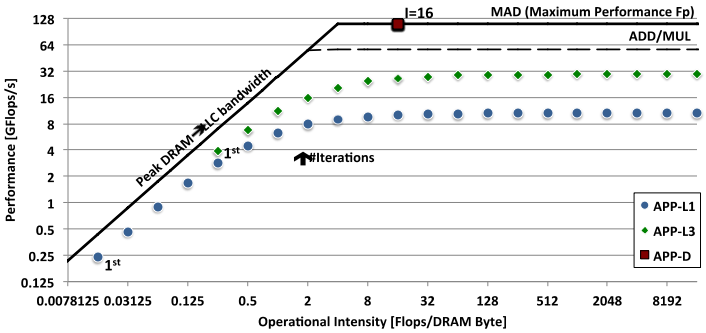
$I=(\Sigma f_i)/(\Sigma b_i)$

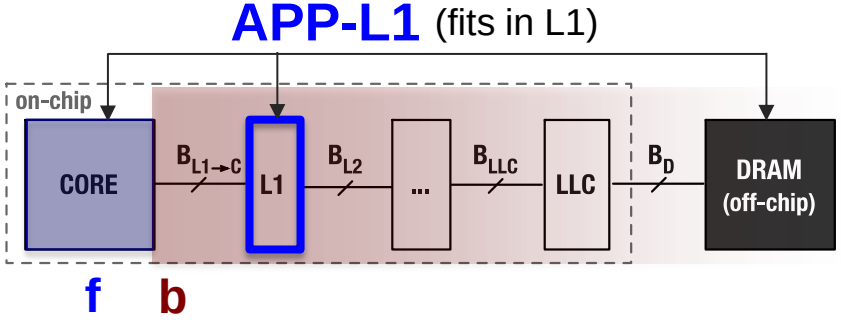
I is constant

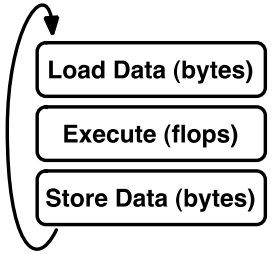
** Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

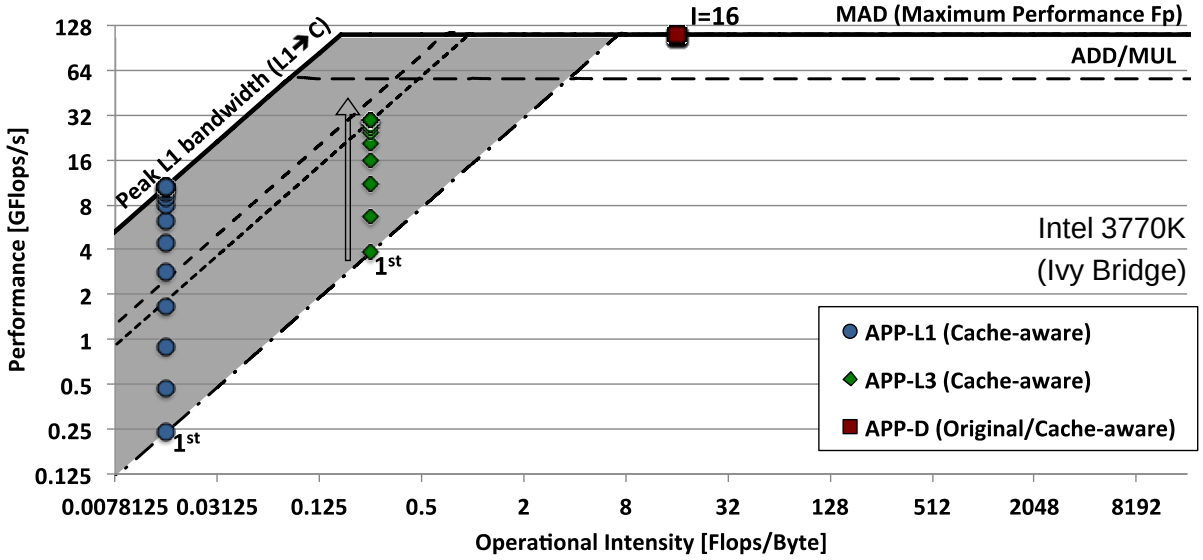
Cache-aware Roofline Model: Hands On

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*









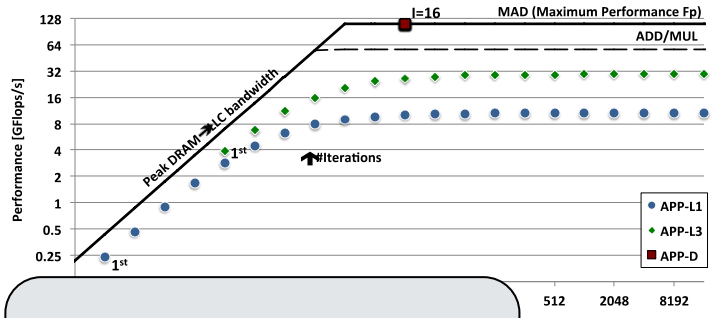
$I=(\Sigma f_i)/(\Sigma b_i)$

I is constant

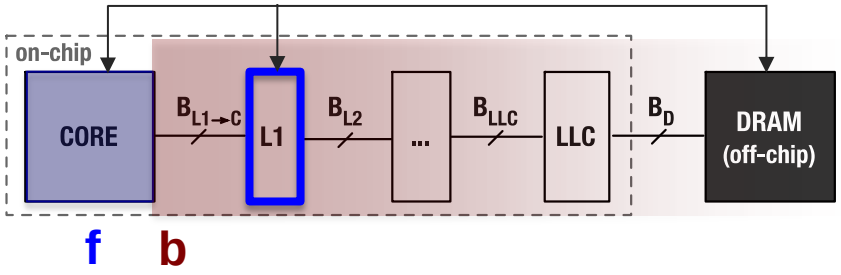
** Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

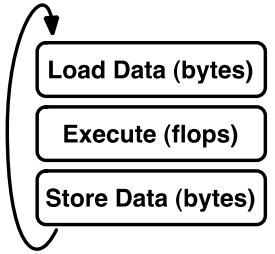
Cache-aware Roofline Model: Hands On

What? *Applications* | **Where?** *Systems and Devices* | **How?** *Modeling and Load Balancing*

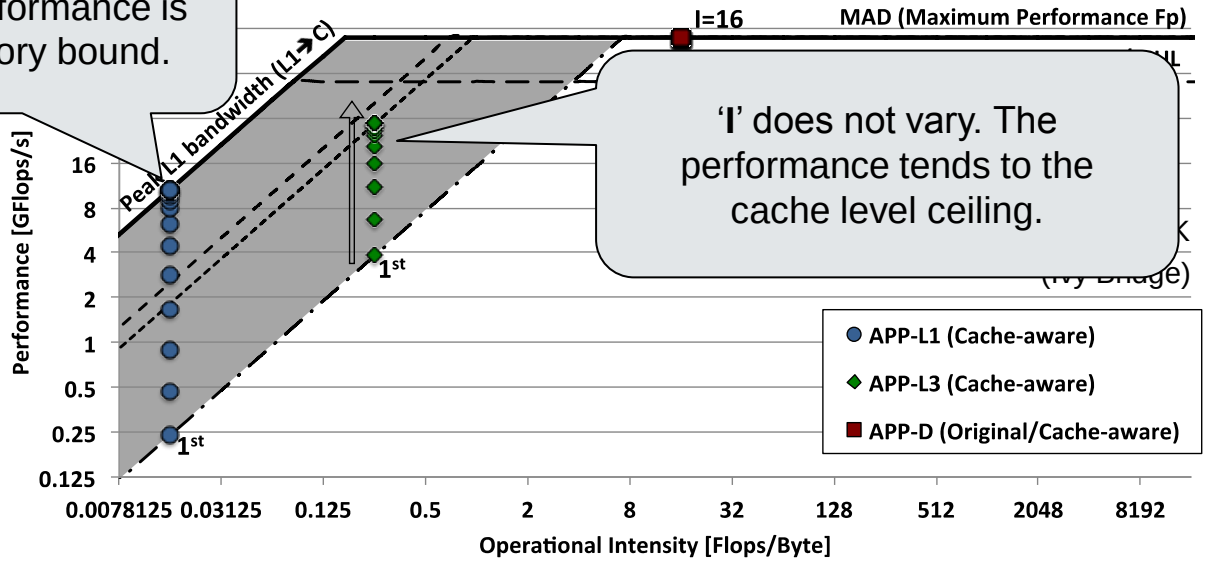


APP-L1 (fits in L1)





Achieves maximum attainable performance is always memory bound.



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

$I=(\Sigma f_i)/(\Sigma b_i)$

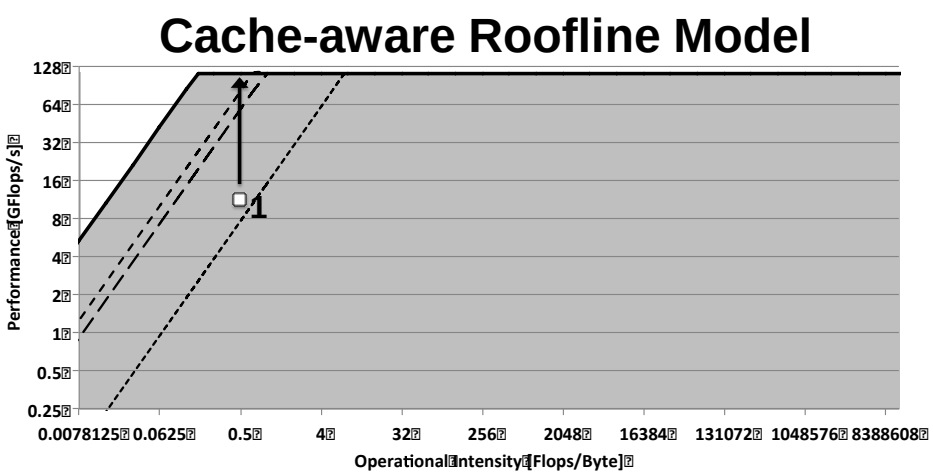
I is constant

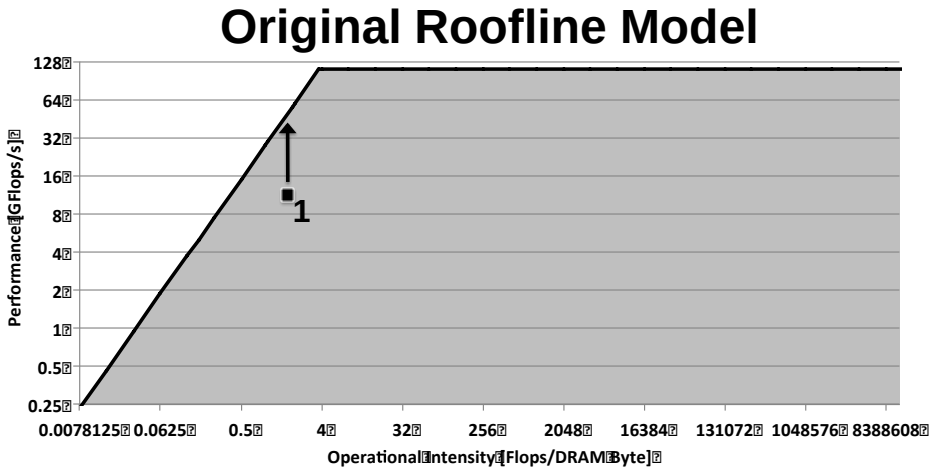
** Ilic, A., Pratas, F. and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft", IEEE Computer Architecture Letters, 2013*

Practical Example: Dense Matrix Multiplication

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*

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





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

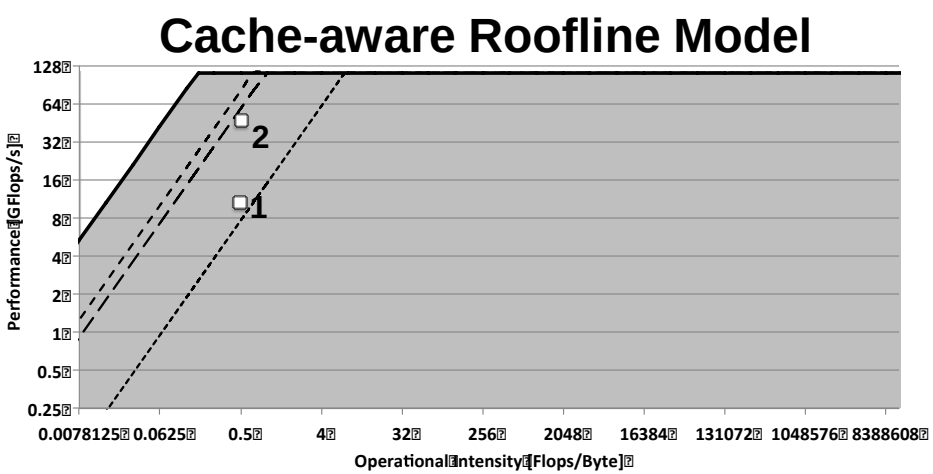
application is in the memory bound region
mainly limited by DRAM
can be optimized up to the slanted part of the model

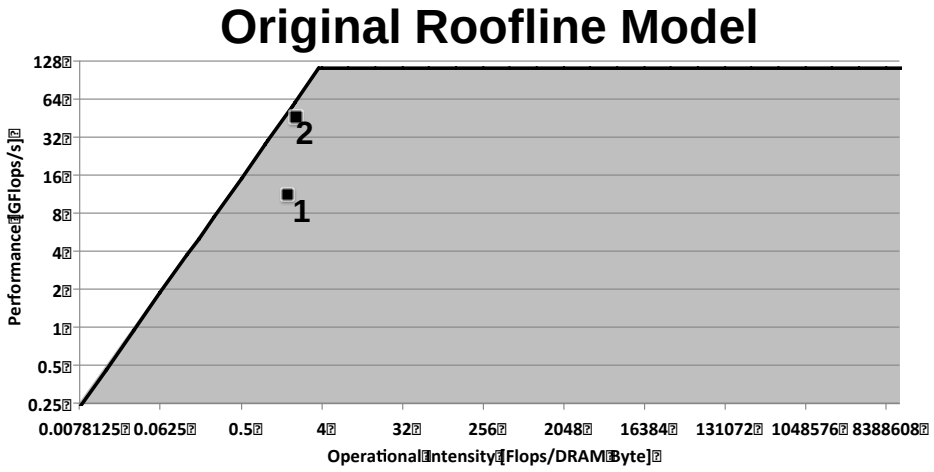
Practical Example: Dense Matrix Multiplication

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*

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

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





application is in the compute bound region
almost hits L3
can be further optimized to hit higher cache levels

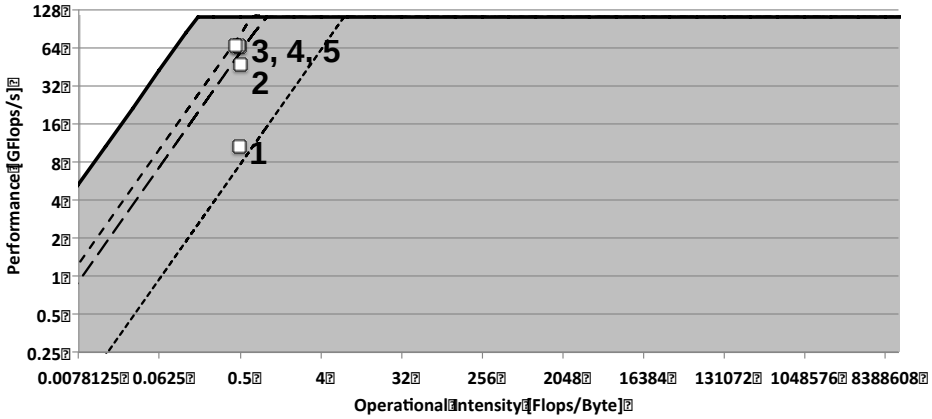
application is in the memory bound region
performance hits the roof of the model
the model suggests that the optimization process is finished

Practical Example: Dense Matrix Multiplication

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*

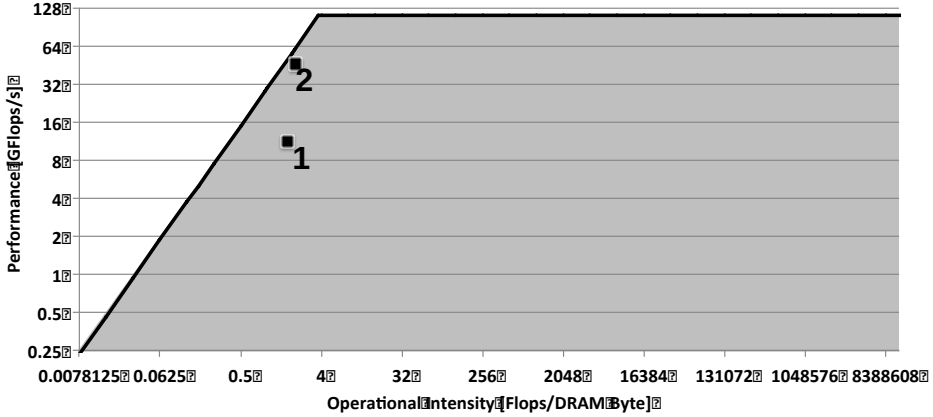
1) **Basic implementation:** All matrices stored in row-major order
2) **Transposition:** One matrix is transposed into column-major
3) **Blocking for L3:** All matrices are blocked to efficiently exploit L3
4) **Blocking for L2:** Second level of blocking to efficiently exploit L2
5) **Blocking for L1:** Data is further blocked to exploit L1

Cache-aware Roofline Model



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

Original Roofline Model



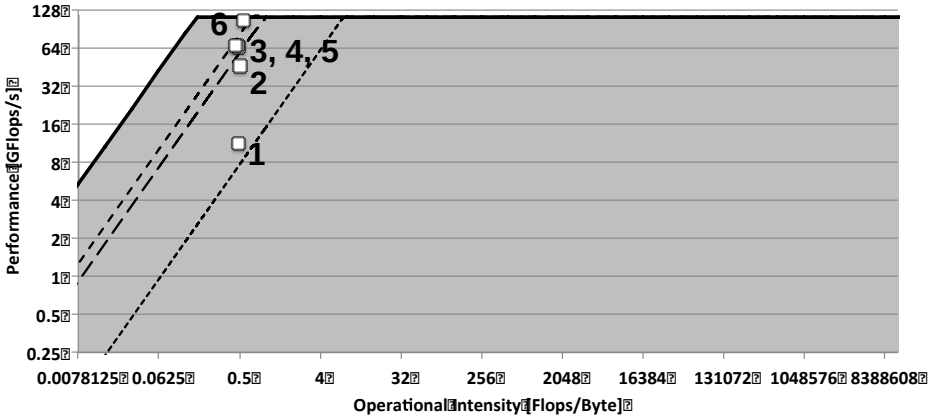
optimization process finished

Practical Example: Dense Matrix Multiplication

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*

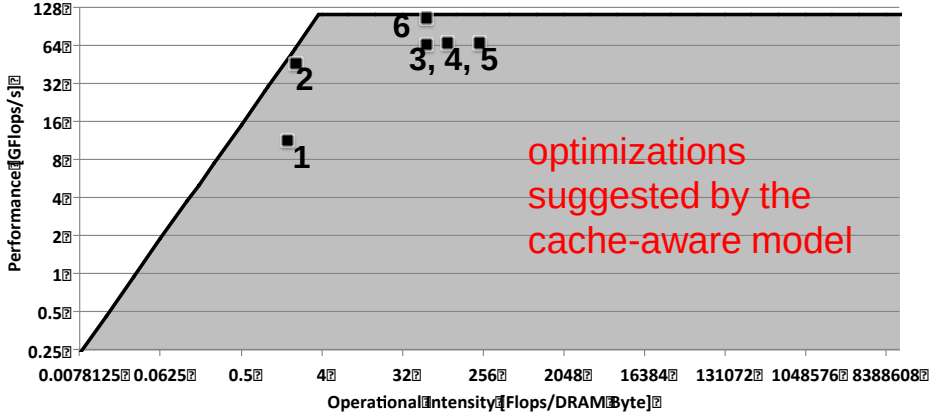
1) **Basic implementation:** All matrices stored in row-major order
2) **Transposition:** One matrix is transposed into column-major
3) **Blocking for L3:** All matrices are blocked to efficiently exploit L3
4) **Blocking for L2:** Second level of blocking to efficiently exploit L2
5) **Blocking for L1:** Data is further blocked to exploit L1
6) **Intel MKL:** Highly optimized implementation

Cache-aware Roofline Model



6 is able to achieve near theoretical performance

Original Roofline Model



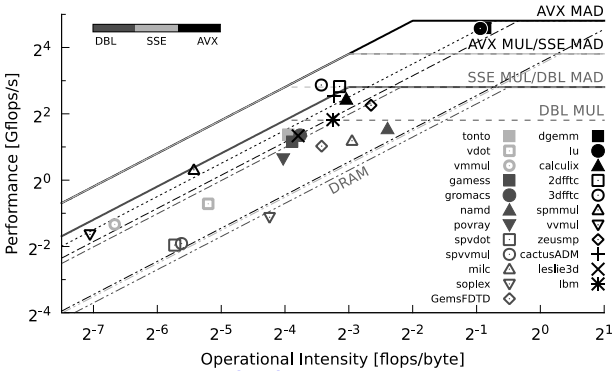
moves to the compute bound region (shift in operational intensity)

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

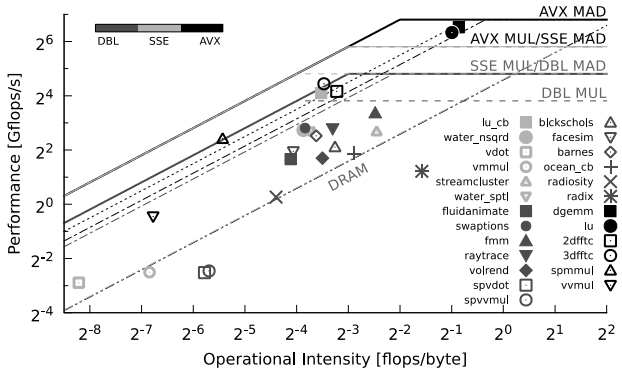
Cache-aware Roofline Models: Use Cases

What? *Applications* → **Where?** *Systems and Devices* → **How?** *Modeling and Load Balancing*

Application Characterization

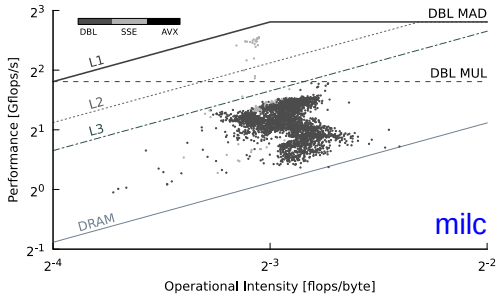


single core

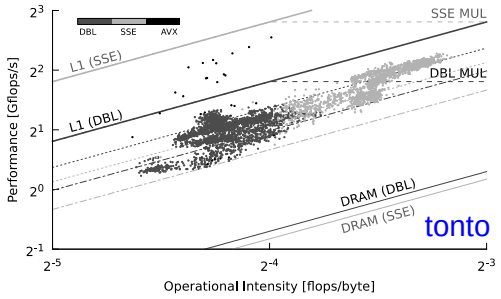


quad-core

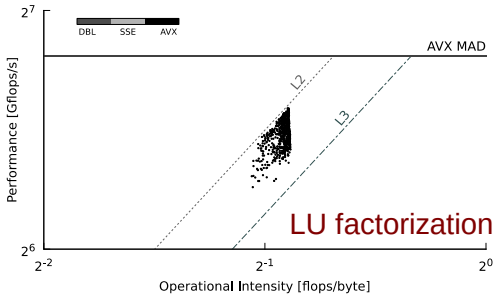
Online Monitoring



milc



tonto



LU factorization

** 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*

Roundup and Conclusions

What?	Where?	How?
Applications	*Systems and Devices*	*Modeling and Load Balancing*
• **Multi-module Applications** - Divisible Load Applications - H.264/AVC Video Encoding (inter-prediction mode)	• **Node: CPU+GPU platform**	• **Load Balancing** **- FEVES -** **F**ramework for **E**fficient parallel **V**ideo **E**ncoding on heterogeneous **S**ystems
• **General (FP) Applications**	• **Device: multicore CPUs**	• **Cache-aware Roofline Model** - Performance and Total Performance

Performance modeling

On-going and Further Work

- **Porting and extending load balancing algorithms**
  - Highly heterogeneous **systems** (CPU+GPU+FPGA), embedded systems …
  - Power- and energy-**efficient computing** (DVFS)

- **Cache-aware Roofline modeling: Future**
  - Power, energy, efficiency …
  - Extending for other device **architectures** (mainly GPUs)
  - **Scheduling** and load balancing for general applications

- Introduce all these techniques and algorithms in the OS
  - Automatic approach: by identifying the characteristics of the applications
  - To have support for the different approaches and user provides additional information
    - Multiple performance modeling and load balance strategies for different architectures, and solutions for all applications

Additional readings

- A. Ilic, F. Pratas and L. Sousa, *"Cache-aware Roofline Model: Upgrading the loft"*, **IEEE Computer Architecture Letters**, **2013**
- A. Ilic, S. Momcilovic, N. Roma and L. Sousa, *"FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems*", **ICPP'14**
- S. Momcilovic, A. Ilic, N. Roma and L. Sousa, "*Dynamic load balancing for real-time video encoding on heterogeneous systems*", **IEEE Transactions on Multimedia, 2014**
- S. Momcilovic, A. Ilic, N. Roma and L. Sousa, *"Collaborative Inter-Prediction on CPU+GPU Systems*", **ICIP'14**
- D. Antão, L. Taniça, A. Ilic, F. Pratas, P. Tomás and L. Sousa, "*Monitoring performance and power for application characterization with Cache-aware Roofline Mode*l", **PPAM'13**

Thank you for your attention!

Questions?