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? Applications

• Multi-module Applications

Where? Systems and Devices

• Node: CPU+GPU platform

How? Modeling and Load Balancing

• Load Balancing

• General (FP) Applications

• Device: multicore CPUs

• Performance modeling
What? Applications

- Multi-module Applications

Where? Systems and Devices

- Node: CPU+GPU platform

How? Modeling and Load Balancing

- Load Balancing

application- and hardware-specific

- General (FP) Applications

- Device: multicore CPUs

- Performance modeling

hardware-specific
Outline

What?
Applications

Where?
Systems and Devices

How?
Modeling and Load Balancing

• Multi-module Applications

• Node: CPU+GPU platform

• Load Balancing

• General (FP) Applications

• Device: multicore CPUs

• Performance modeling
**Node: CPU+GPU platform**

**What?**
- Applications

**Where?**
- Systems and Devices

**How?**
- 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?**
- **Applications**

**Where?**
- **Systems and Devices**

**How?**
- **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, …)
• **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!

![Diagram showing CPU and GPU cores and interconnection buses.](image)
Node: CPU+GPU platform

**What?**
Applications

**Where?**
Systems and Devices

**How?**
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)
• **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*)
• **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? Applications

- Multi-module Applications
  - Divisible Load Applications
  - H.264/AVC Video Encoding (inter-prediction mode)

Where? Systems and Devices

- Node: CPU+GPU platform
- Device: multicore CPUs

How? Modeling and Load Balancing

- Load Balancing
- Performance modeling
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

M1: load balancing, modeling…

M2: load balancing, modeling…

repartitioning
Divisible Load Processing

- **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

![Diagram of H.264/AVC Video Encoding]

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

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

ME+INT+SME
- min. 94% on GPU (92% CPU)
- Load balancing and modeling
Outline

What?
Applications

Where?
Systems and Devices

How?
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 - Framework for Efficient parallel Video Encoding on heterogeneous Systems

- General (FP) Applications
- Device: multicore CPUs
- Performance modeling
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

FEVES: Framework Control

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**UNIFIED FRAMEWORK**

- Load Balancing
- Performance Characterization
- Video Coding Manager
- Parallel Modules (CPU, GPU, ...)
- Data Access Management

**FRAMEWORK CONTROL**

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

**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

---

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)

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

FEVES: Load Balancing

What?
Applications

Where?
Systems and Devices

How?
Modeling and Load Balancing

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

Input: \( N, n_w, n_c, T_{\text{R}}, K^m_i, K^l_i, K^s_i \)
Input: \( K^f_i, K^d_i, K^s_i, K^d_i, K^m_i, K^v_i, K^d_i, K^m_i, K^v_i \)
Output: \( m_i = \{m_i\}, l_i = \{l_i\}, s = \{s_i\}, \sigma = \{\sigma_i\}, \sigma^r = \{\sigma_i^r\} \)

Objective: minimize \( T_{\text{tot}} \)

\[
\begin{align*}
\sum_{i=1}^{n_w+n_c} m_i &= N, \quad \sum_{i=1}^{n_w+n_c} l_i &= N, \quad \sum_{i=1}^{n_w+n_c} s_i &= N \\
\forall i \in \{n_w+1, \ldots, n_w+n_c\} : \\
\tau_1 + s_i K^l_i &\leq \tau_1 \\
\tau_1 + s_i K^l_i &\leq \tau_2 \\
m_1 K^f_i + m_1 K^m_i + m_1 K^v_i &\leq \tau_1 \\
l_1 K^f_i + l_1 K^d_i + \Delta m_1 K^s_i + m_1 K^v_i &\leq \tau_1 \\
m_1 K^f_i + m_1 K^v_i + \Delta m_1 K^s_i + m_1 K^v_i &\leq \tau_1 \\
\tau_1 + \Delta m_1 K^f_i + \Delta m_1 K^s_i + m_1 K^v_i &\leq \tau_2 \\
\tau_1 + \Delta m_1 K^f_i + \Delta m_1 K^s_i + m_1 K^v_i &\leq \tau_2 \\
\tau_2 + (N - s_1) K^v_i &\leq \tau_{\text{tot}} \\
\forall i \in \{2, \ldots, n_w\} : \\
N K^f_i + m_1 K^s_i + m_1 K^v_i &\leq \tau_1 \\
N K^f_i + m_1 K^s_i + m_1 K^v_i + s_i K^d_i &\leq \tau_1 \\
N K^f_i + m_1 K^s_i + m_1 K^v_i + s_i K^d_i + \sigma^r_{i-1} K^f_i + \Delta m_1 K^s_i &\leq \tau_1 \\
\tau_1 + \Delta m_1 K^f_i + \Delta m_1 K^s_i &\leq \tau_2 \\
\sigma_i &= \min(N-l_i - \Delta m_1, (\tau_{\text{tot}} - \tau_2)/K^f_i) \\
\sigma^r_i &= N - l_i - \Delta m_1 - \sigma_i \\
\Delta m_1 &= \text{MS_BOUNDS}(m, s) \\
\Delta l_i &= \text{LS_BOUNDS}(l, s)
\end{align*}
\]

FEVES: Experimental results

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

### 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
- **CPU_N**: Intel Nehalem i7 950
- **CPU_H**: Intel Haswell i7 4770K
- **GPU_F**: NVIDIA Fermi GTX580
- **GPU_K**: NVIDIA Kepler GTX780Ti

#### Heterogeneous Systems
- **SysNF**: CPU_N + GPU_F
- **SysNFF**: CPU_N + 2xGPU_F
- **SysHK**: CPU_H + GPU_K

---

FEVES: Experimental results

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

- 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

Outsage

What?
Applications

• Multi-module Applications
  - Divisible Load Applications
  - H.264/AVC Video Encoding (inter-prediction mode)

Where?
Systems and Devices

• Node: CPU+GPU platform

How?
Modeling and Load Balancing

• Load Balancing
  - FEVES - Framework for Efficient parallel Video Encoding on heterogeneous Systems

• General (FP) Applications

• Device: multicore CPUs

• Performance modeling
Outline

**What?**
Applications

**Where?**
Systems and Devices

**How?**
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 - Framework for Efficient parallel Video Encoding on heterogeneous Systems

**Device: multicore CPUs**

**Cache-aware Roofline Model**
- Performance and Total Performance

**General (FP) Applications**
• **Multi-cores**: Powerful cores and memory hierarchy (caches and DRAM)

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

---

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)

---

![Diagram of multi-cores](image)

**APP-D** (data traffic from DRAM)

\[ I = \frac{\sum f_i}{\sum b_i} \]

\( I \) is constant

---

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

APP-L3 (data fits in L3)

\[ I_1 = \frac{f_1}{b_1} \]

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

**APP-L3** (data fits in L3)

\[ I_1 = \frac{f_1}{b_1} \]
\[ I_2 = \frac{(f_1 + f_2)}{b_1} \]

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

![Diagram of the original roofline model](https://example.com/diagram)

**APP-L3** (data fits in L3)

\[ I_1 = \frac{f_1}{b_1} \]

\[ I_2 = \frac{(f_1 + f_2)}{b_1} \]

\[ I_i = \frac{(\Sigma f_i)}{b_1} \]

\( I \) is variable

---

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

![Diagram of multi-cores and memory hierarchy](image)

**APP-L1** (data fits in L1)

\[ I_1 = \frac{f_1}{b_1} \]
\[ I_2 = \frac{f_1 + f_2}{b_1} \]
\[ I_i = \frac{\sum f_i}{b_1} \]

*I is variable*

---

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

- **Fixed I**: unexpected performance for different $I$ levels
- **Does not achieve maximum attainable performance**
- **I varies with the problem size. Memory bound becomes compute bound.**

\[ I_1 = \frac{f_1}{b_1} \]
\[ I_2 = \frac{f_1 + f_2}{b_1} \]
\[ I_i = \frac{\Sigma f_i}{b_1} \]

• **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

- Measured
- Theoretical

- MAD (Peak Performance)
- ADD/MUL

- Operational Intensity [Flops/Byte]
- Performance [GFlops/s]
Cache-aware Roofline Model

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

Memory bandwidth

Performance variation

7/1/2014
Cache-aware Roofline Model

What?  
Applications

Where?  
Systems and Devices

How?  
Modeling and Load Balancing

Memory bandwidth

Performance variation

- Measured
- Theoretical

Data Traffic [KBytes]

Double FP operations [Flops]

Operational Intensity [Flops/Byte]
Cache-aware Roofline Model

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

Memory bandwidth

Performance variation

- Measured
- Theoretical

4 Cores  L1 → C
2 Cores  L2 → C
1 Core  L3 → C
DRAM → LLC
DRAM → C

MAD (Peak Performance)

F(φ)

ADD/MUL

MAD (Maximum Performance Fp)

ADD/MUL
Cache-aware Roofline Model

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

Memory bandwidth

Performance variation

MAD (Maximum Performance Fp)

Operational Intensity [Flops/Byte]

Performance [GFlops/s]

Memory Bandwidth [GB/s]

Data Traffic [KBytes]
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

---

Cache-aware Roofline Model

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

- **Complete memory hierarchy**
  - Influence of caches and DRAM to performance

- **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 multicore systems
  - 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

**Intel 3770K (Ivy Bridge)**

4 Cores (AVX ADD/MUL)

- 4 Cores (AVX ADD/MUL)
- 3 Cores (L1-C, L2-C, L3-C)
- MAD (Peak performance)

Performance [Gflops/s]

Operational Intensity [flops/byte]
• 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

---

**APP-D (data traffic from DRAM)**

$I = \frac{\sum f_i}{\sum b_i}$

$I$ is constant

---

Cache-aware Roofline Model: Hands On

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
- Applications

**Where?**
- Systems and Devices

**How?**
- Modeling and Load Balancing

---

### Intel 3770K (Ivy Bridge)

**I** is constant

\[
I = \frac{\sum f_i}{\sum b_i}
\]

Cache-aware Roofline Model: Hands On

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**Intel 3770K**
(Ivy Bridge)

\[ I = \frac{\sum f_i}{\sum b_i} \]

\( I \) is constant

---

Cache-aware Roofline Model: Hands On

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---

**What?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

---
Cache-aware Roofline Model: Hands On

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

What?
Applications

Where?
Systems and Devices

How?
Modeling and Load Balancing

**Intel 3770K (Ivy Bridge)**

\[ I = \frac{\sum f_i}{\sum b_i} \]

**I** is constant

Cache-aware Roofline Model: Hands On

What?
Applications

Where?
Systems and Devices

How?
Modeling and Load Balancing

Achieves maximum attainable performance is always memory bound.

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

\[ I = \frac{\sum f_i}{\sum b_i} \]

\( I \) is constant

Practical Example: Dense Matrix Multiplication

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

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

---

Cache-aware Roofline Model

![Graph showing the cache-aware Roofline Model]

Original Roofline Model

![Graph showing the original Roofline Model]

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?**
Applications

**Where?**
Systems and Devices

**How?**
Modeling and Load Balancing

1) **Basic implementation**: All matrices stored in row-major order.
2) **Transposition**: One matrix is transposed into column-major

---

**Cache-aware Roofline Model**

---

**Original Roofline Model**

---

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

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

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

Optimization process finished
Practical Example: Dense Matrix Multiplication

**What?**
- Applications

**Where?**
- Systems and Devices

**How?**
- 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)

---

Cache-aware Roofline Models: Use Cases

**What?**

Applications

**Where?**

Systems and Devices

**How?**

Modeling and Load Balancing

---

**Application Characterization**

---

**Online Monitoring**

---

Roundup and Conclusions

What?
- Multi-module Applications
  - Divisible Load Applications
  - H.264/AVC Video Encoding (inter-prediction mode)

Where?
- Node: CPU+GPU platform

How?
- Load Balancing
  - FEVES - Framework for Efficient parallel Video Encoding on heterogeneous Systems

What?
- General (FP) Applications

Where?
- Systems and Devices

How?
- Device: multicore CPUs

- Cache-aware Roofline Model
  - Performance and Total Performance
• **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


– S. Momcilovic, A. Ilic, N. Roma and L. Sousa, “Collaborative Inter-Prediction on CPU+GPU Systems”, ICIP’14

Thank you for your attention!

Questions?