Module 2 Part 3 Performance Analysis: Roofline Model

Carnegie Mellon University
18-645

Jike Chong
Ian Lane

What we discussed last time:

Fast Platforms + **Good Techniques**

- **Multicore platforms**
- **Manycore platforms**
- **Cloud platforms**

- **Data structures**
- **Algorithms**
- **Software Architecture**

- Highlighted the difference between multicore and manycore platforms
- **Exposing** concurrency in k-means, **Exploiting** parallelism by **exploring** mappings from application to platform
- OpenMP: **An abstraction for multicore parallel programming**
- Advanced optimizations for CPUs
- How do we know **how much optimization is possible**?

Answers you should know after this…

- What is the roofline model? What are the metrics and axis used?
- What's the difference between:
  - "flop's per memory instruction" from "flop's per DRAM byte"?
- Consider an image Image[height][width]. If one were to stride through the columns of values, what would be the effects? How would they be mapped to the roofline ?
- How does one model incomplete SIMDization (e.g. half the flop's can be SIMDized), insufficient ILP (some dependent flop's), or an imbalance between FPMUL's and FPADD's on the roofline ?
- How would one model {branch mispredicts, TLB misses, or too many streams for the prefetchers} on the roofline?

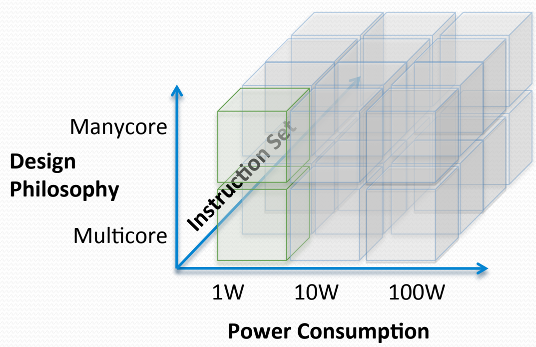
Performance Analysis

- Motivation: Diversity of Computation Platforms
- What is "Fast"?
  - Latency and throughput
  - Task and data
- **The Roofline Model**
  - The Ceilings and the Walls
  - Categories of Optimizations
- Measuring Arithmetic Intensity
- How is this relevant to writing fast code?

Landscape of Computing Platforms

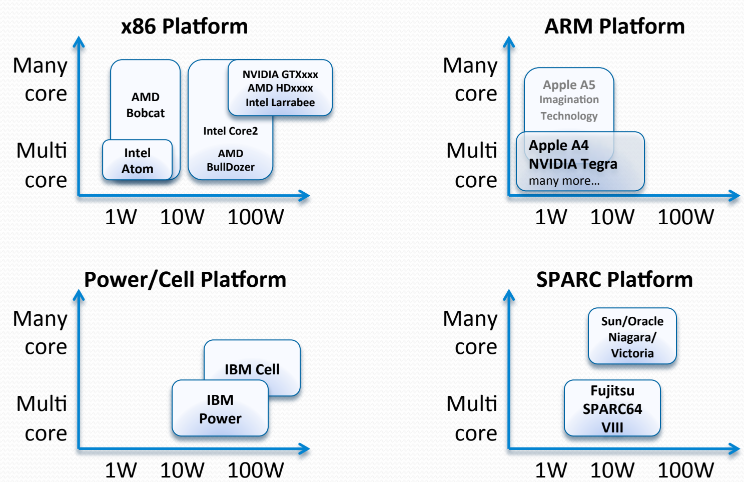
- We are living in a flurry of technology innovations!
- We have been looking at the innovations along three axis:
  - Instruction Set
  - Design Philosophy
  - Power Consumption

Instruction Set
- **x86**
- **ARM**
- **Power/Cell**
- **SPARC**



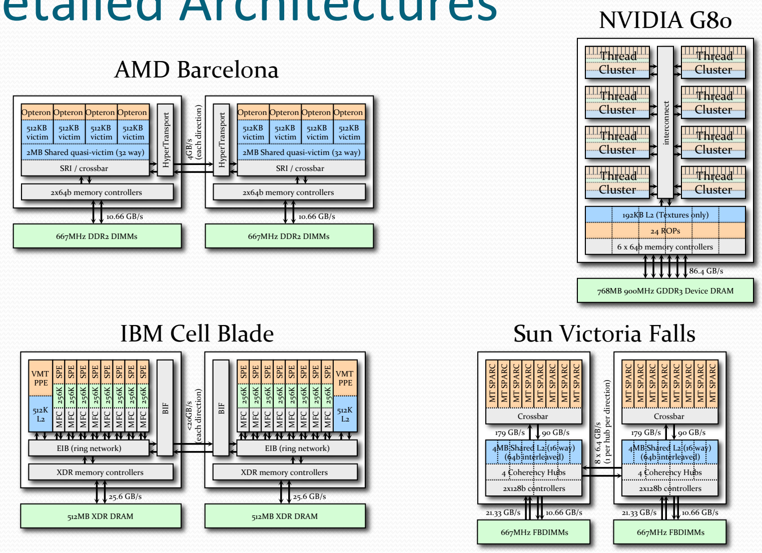
18-645 – How to Write Fast Code? *Carnegie Mellon University (c) 2012-2014* 5

Computing Technology Innovations



18-645 – How to Write Fast Code? *Carnegie Mellon University (c) 2012-2014* 6

Detailed Architectures



Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Challenges / Goals

- Situation:
  - Architectures of parallel processors have extreme variations
  - Characteristics of numerical methods also vary dramatically
- Challenge:
  - Optimization varies from one architecture-kernel combination to another
  - How do we understand whether we have attained good performance? (high fraction of theoretical peak performance)
  - How do we identify performance bottlenecks?
  - How do we enumerate potential remediation strategies?

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

What is “Fast”?

	Task	Data	Synchronization
LOWER Latency	Time from task start to task finish (seconds)	Time from request to receiving data (seconds)	Time from start of synchronization to completion (seconds)
HIGHER Throughput	# of tasks executed per unit time (tasks/second)	# of Bytes transferred per unit time (Bytes/second)	# of sync operations per unit time (sync ops/second)
Concurrency = (Latency * Throughput)	# of Tasks concurrently managed (tasks)	# of memory operations concurrently managed (Memory instructions)	# of sync operation in flight at the same time (sync operations)

Performance Analysis

- Motivation: Diversity of Computation Platforms
- What is “Fast”?
  - Latency and throughput
  - Task and data
- **The Roofline Model**
  - The Ceilings and the Walls
  - Categories of Optimizations
- Measuring Arithmetic Intensity
- How is this relevant to writing fast code?

What is "Fast"?

- Latency ➔ "Runtime"
- Throughput ➔ "Performance"

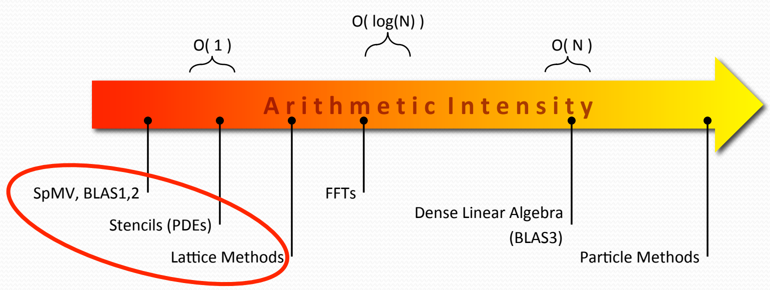
- **Latency**:
  - What ultimately matters

- **Throughput**:
  - A means to reduce total "Runtime" of an application
  - Usually measured in floating point operations per second (**FLOPS**)
    (Floating point operations = addition + multiplication)
  - Assume negligible amount of division, exponent, sin/cos…

Careful: **Higher Performance != Shorter Runtime**

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Task: Arithmetic Intensity



- **True Arithmetic Intensity (AI) ~ Total Flops / Total DRAM Bytes**

- Some HPC kernels have an arithmetic intensity that scales with problem size (increased temporal locality)
- Others have constant intensity

- Arithmetic intensity is ultimately limited by compulsory traffic
- Arithmetic intensity is diminished by conflict or capacity misses.

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Data: Three Classes of Locality

- Spatial Locality
  - data is transferred from cache to registers in words.
  - However, data is transferred to the cache in 64-128Byte lines
  - using every word in a line maximizes spatial locality.
  - **transform data structures into *structure of arrays* (SoA) layout**

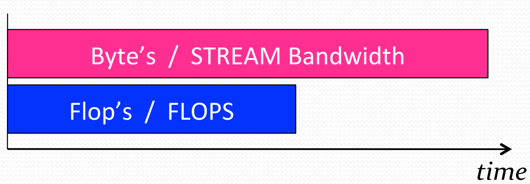
- Temporal Locality
  - reusing data (either registers or cache lines) multiple times
  - amortizes the impact of limited bandwidth.
  - **transform loops or algorithms to maximize reuse.**

- Sequential Locality
  - Many memory address patterns access cache lines sequentially.
  - CPU's hardware stream prefetchers exploit this observation to hide speculatively load data to memory latency.
  - **Transform loops to generate (a few) long, unit-stride accesses.**

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Data: Overlap of Communication

- Consider a simple example in which a FP kernel maintains a working set in DRAM.
- ***We assume*** we can perfectly overlap computation with communication or v.v. either through prefetching/DMA and/or pipelining (decoupling of communication and computation)
- Time, then, is the maximum of the time required to transfer the data and the time required to perform the floating point operations.



Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Performance Analysis

- Motivation: Diversity of Computation Platforms
- What is "Fast"?
  - Latency and throughput
  - Task and data
- **The Roofline Model**
  - The Ceilings and the Walls
  - Categories of Optimizations
- Measuring Arithmetic Intensity
- How is this relevant to writing fast code?

Roofline Model

- Synthesize communication, computation, and locality into a single visually-intuitive performance figure using bound and bottleneck analysis.

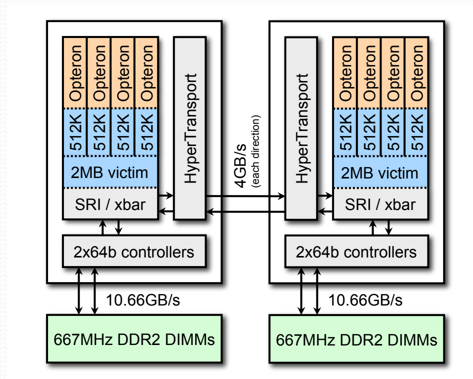
$$\textbf{Attainable Performance}_{ij} = \min \begin{cases} \textbf{FLOP/s with Optimizations}_{1\text{-}i} \\ \textbf{AI * Bandwidth with Optimizations}_{1\text{-}j} \end{cases}$$

- where *optimization i* can be SIMDize, or unroll, or SW prefetch, …
- Given a kernel's arithmetic intensity (based on DRAM traffic after being filtered by the cache), programmers can inspect the figure, and bound performance.
- Moreover, provides insights as to which optimizations will potentially be beneficial.

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Example

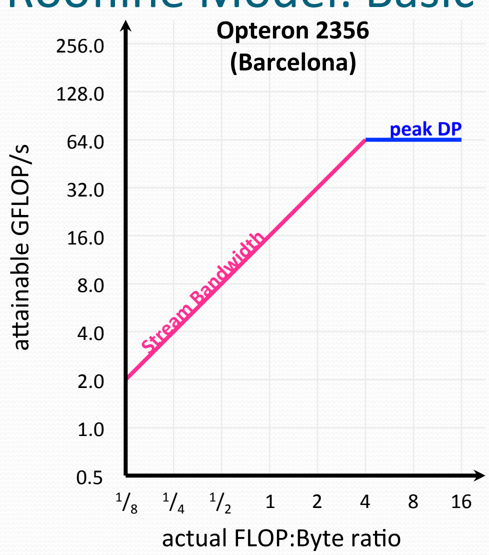
- Consider the Opteron 2356:
  - Dual Socket (NUMA)
  - limited HW stream prefetchers
  - quad-core (8 total)
  - 2.3GHz
  - 2-way SIMD (DP)
  - separate FPMUL and FPADD datapaths
  - 4-cycle FP latency



- Assuming **expression of parallelism** is the challenge on this architecture, what would a Roofline Model look like ?

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Roofline Model: Basic Concept

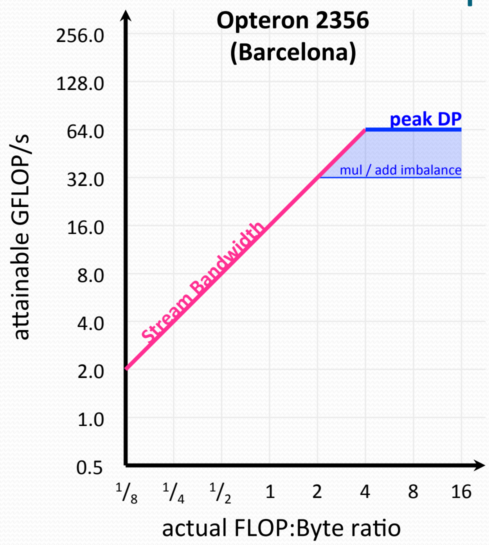


- Plot on log-log scale
- Given Arithmetic Intensity, we can easily bound performance
- But architectures are much more complicated

- We will bound performance as we eliminate specific forms of in-core parallelism

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Roofline Model: Computational Ceilings

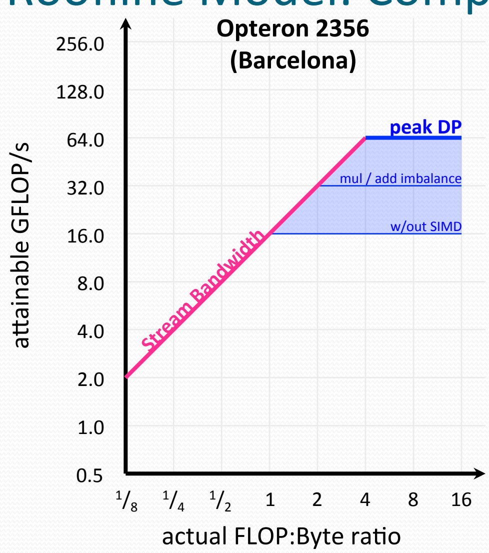


- Opterons have dedicated multipliers and adders.
- If the code is dominated by adds, then attainable performance is half of peak.
- We call these **Ceilings**
- They act like constraints on performance

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

18-645 – How to Write Fast Code? *Carnegie Mellon University (c) 2012-2014* 19

Roofline Model: Computational Ceilings

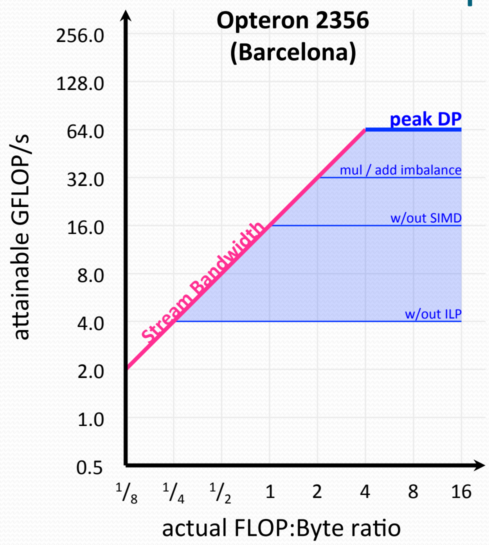


- Opterons have 128-bit datapaths.
- If instructions aren't SIMDized, attainable performance will be halved

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

18-645 – How to Write Fast Code? *Carnegie Mellon University (c) 2012-2014* 20

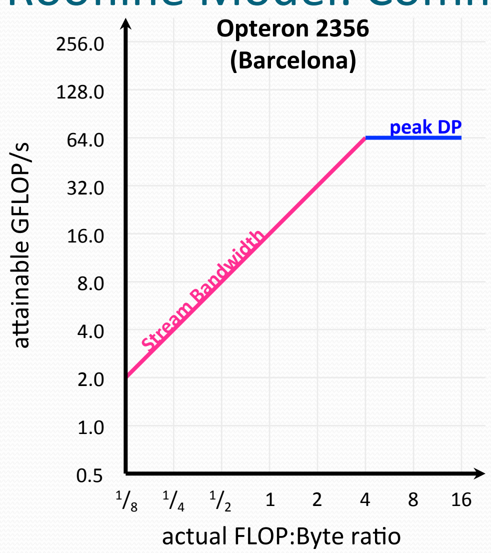
Roofline Model: Computational Ceilings



- On Opterons, floating-point instructions have a 4 cycle latency.
- If we don't express 4-way ILP, performance will drop by as much as 4x

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

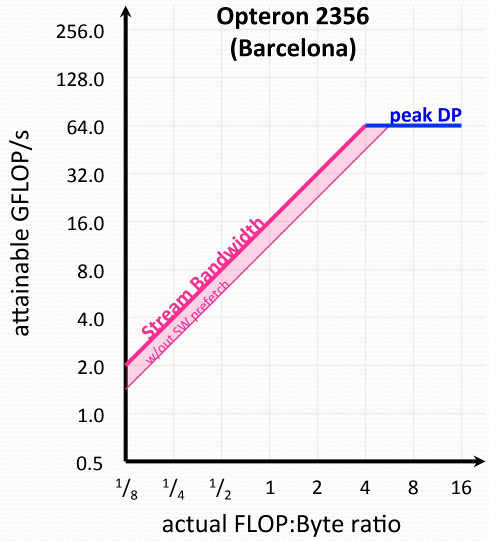
Roofline Model: Communication Ceilings



- We can perform a similar exercise taking away parallelism from the memory subsystem

Sam Williams, The Roofline Model, Lawrence Berkeley national Laboratory, 2011

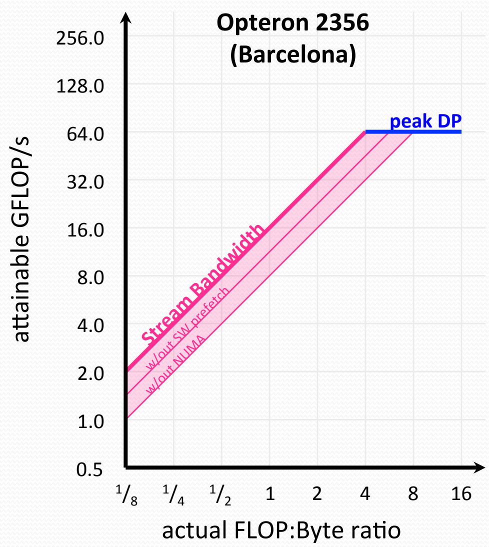
Roofline Model: Communication Ceilings



- Explicit software prefetch instructions are required to achieve peak bandwidth

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

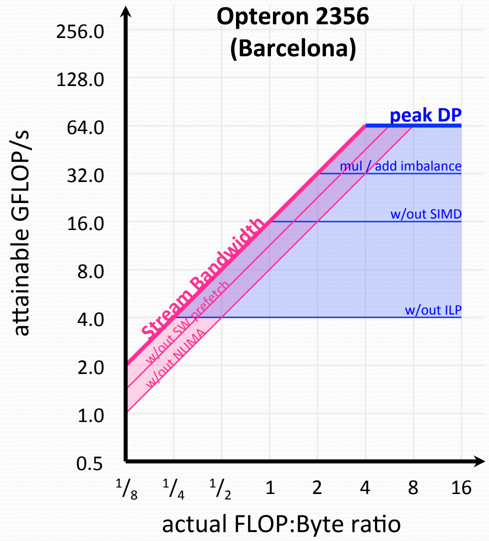
Roofline Model: Communication Ceilings



- Opterons are NUMA
- As such memory traffic must be correctly balanced among the two sockets to achieve good Stream bandwidth.
- We could continue this by examining strided or random memory access patterns

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

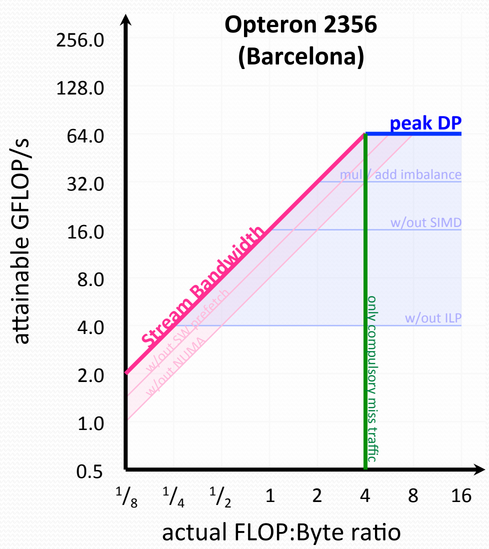
Roofline Model: Computation + Communication Ceilings



- We may bound performance based on the combination of expressed in-core parallelism and attained bandwidth.

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Roofline Model: Computation + Communication Ceilings



- Remember, memory traffic includes more than just compulsory misses.
- Actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Compulsory Misses}}$$

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

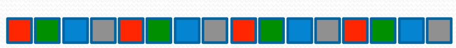
Cache Behavior

- Knowledge of the underlying cache operation can be critical.
- For example, caches are organized into lines. Lines are organized into sets & ways (associativity)
  - Thus, we must mimic the effect of Mark Hill's 3C's of caches
  - Impacts of conflict, compulsory, and capacity misses are both architecture- and application-dependent.
  - **Ultimately they reduce the actual flop:byte ratio.**

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Cache Behavior

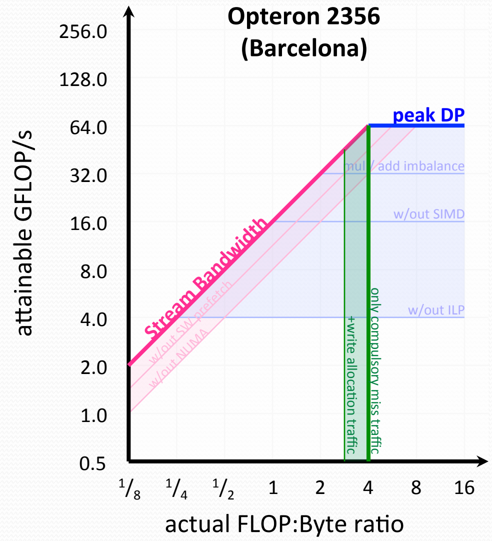
- Moreover, many caches are write allocate.
  - a write allocate cache read in an entire cache line upon a write miss.
  - If the application ultimately overwrites that line, the read was superfluous (**further reduces flop:byte ratio**)
- Because programs access data in words, but hardware transfers it in 64 or 128B cache lines, spatial locality is key
  - **Array-of-structure data layouts can lead to dramatically lower flop:byte ratios.**
  - e.g. if a program only operates on the "red" field of a pixel, bandwidth is wasted.



...

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Roofline Model: Locality Walls

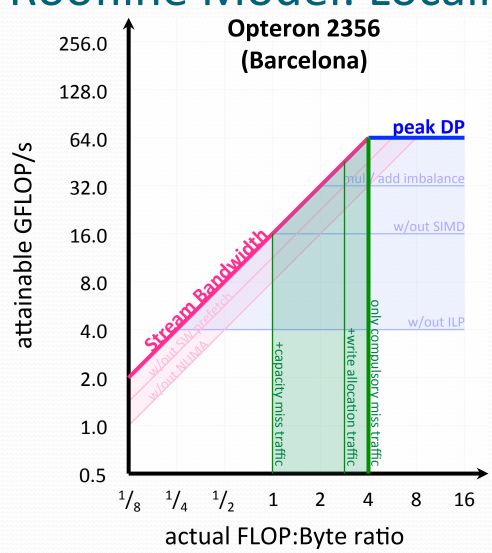


- Remember, memory traffic includes more than just compulsory misses.
- As such, actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Allocations + Compulsory Misses}}$$

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Roofline Model: Locality Walls

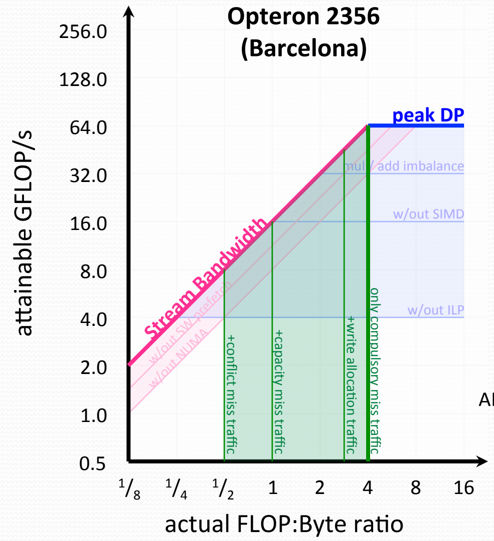


- Remember, memory traffic includes more than just compulsory misses.
- As such, actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Allocations + Compulsory Misses}}$$

Sam Williams, The Roofline Model, Lawrence Berkeley national Laboratory, 2011

Roofline Model: Locality Walls

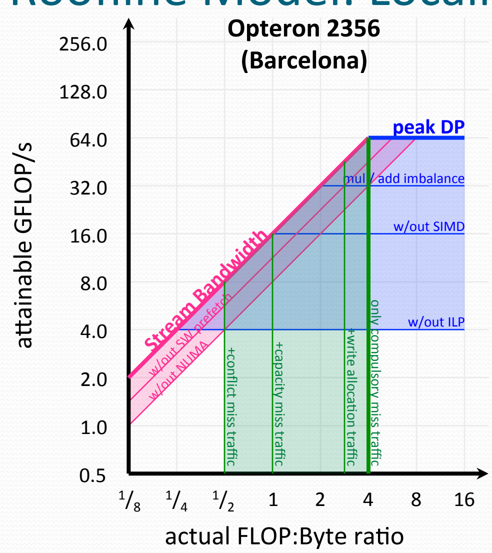


- Remember, memory traffic includes more than just compulsory misses.
- As such, actual arithmetic intensity may be substantially lower.
- Walls are unique to the architecture-kernel combination

$$AI = \frac{\text{FLOPs}}{\text{Conflict + Capacity + Allocations + Compulsory}}$$

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Roofline Model: Locality Walls



- Optimizations remove these walls and ceilings which act to constrain performance.

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

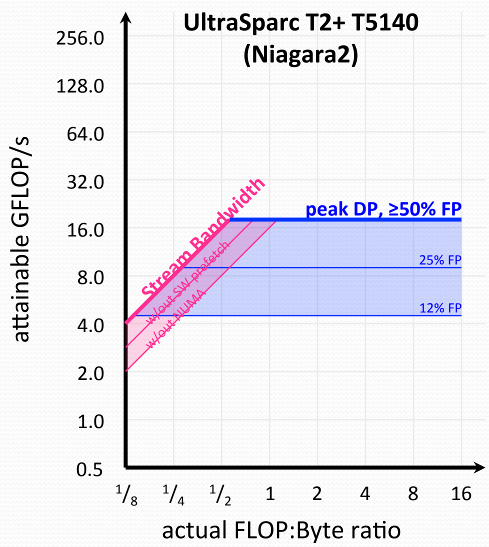
Instruction Issue Bandwidth

- On a superscalar processor, there is likely ample instruction issue bandwidth.
- This allows loads, integer, and FP instructions to be issued simultaneously.
- As such, we assumed that expression of parallelism was the underlying challenge for in-core.

- However, on some architectures, finite instruction-issue bandwidth can become a major impediment to performance.

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Roofline Model: Instruction Mix



- As the instruction mix shifts away from floating-point, finite issue bandwidth begins to affect limits on in-core performance.
- On Niagara2, with dual issues units but only 1 FPU, FP instructions must constitute ≥50% of the mix to attain peak performance.

- A similar approach should be used on GPUs where proper use of CUDA solves the parallelism challenges.

Performance Analysis

- Motivation: Diversity of Computation Platforms
- What is "Fast"?
  - Latency and throughput
  - Task and data
- **The Roofline Model**
  - The Ceilings and the Walls
  - Categories of Optimizations
- Measuring Arithmetic Intensity
- How is this relevant to writing fast code?

Optimization Categorization

Maximizing *(attained)* **In-core Performance**	**Maximizing** *(attained)* **Memory Bandwidth**	**Minimizing** *(total)* **Memory Traffic**

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Optimization Categorization

Maximizing In-core Performance	Maximizing Memory Bandwidth	Minimizing Memory Traffic

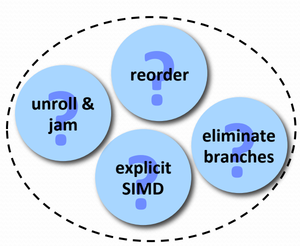
- Exploit in-core parallelism (ILP, DLP, etc...)
- Good (enough) floating-point balance

Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

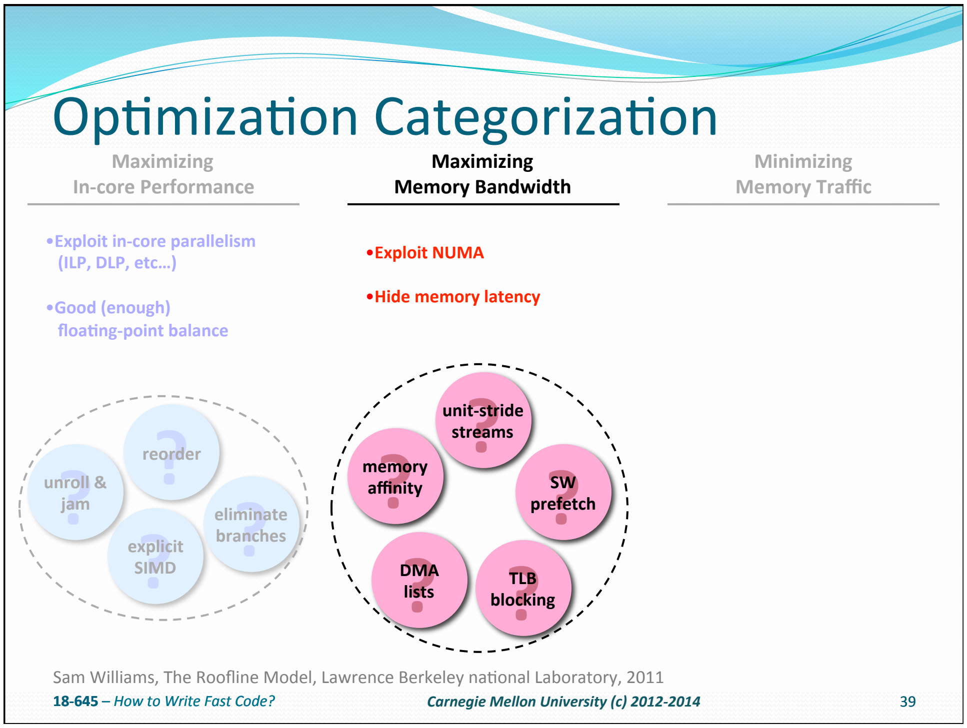
Optimization Categorization

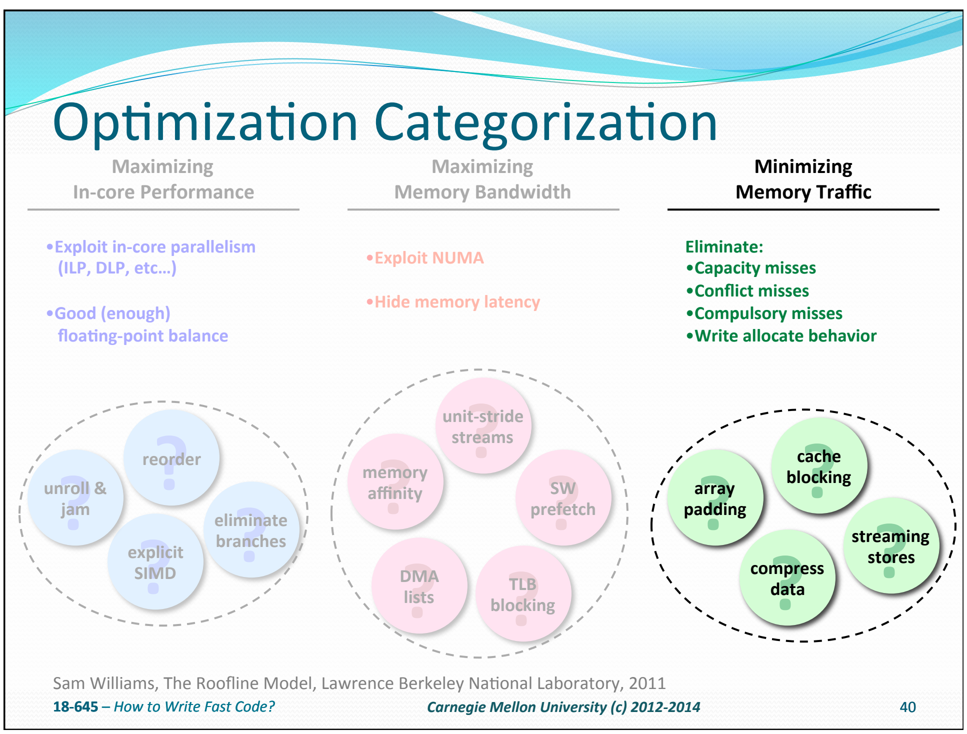
Maximizing In-core Performance	Maximizing Memory Bandwidth	Minimizing Memory Traffic

- Exploit in-core parallelism (ILP, DLP, etc...)
- Good (enough) floating-point balance



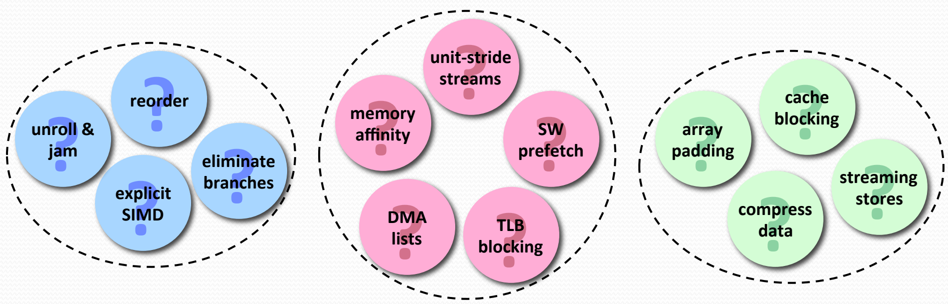
Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011


Optimization Categorization
Maximizing In-core Performance
Maximizing Memory Bandwidth
Minimizing Memory Traffic
•Exploit in-core parallelism (ILP, DLP, etc…)
•Good (enough) floating-point balance
•Exploit NUMA
•Hide memory latency
reorder
unroll & jam
eliminate branches
explicit SIMD
unit-stride streams
memory affinity
SW prefetch
DMA lists
TLB blocking
Sam Williams, The Roofline Model, Lawrence Berkeley national Laboratory, 2011
18-645 – How to Write Fast Code?
Carnegie Mellon University (c) 2012-2014
39


Optimization Categorization
Maximizing In-core Performance
Maximizing Memory Bandwidth
Minimizing Memory Traffic
•Exploit in-core parallelism (ILP, DLP, etc…)
•Good (enough) floating-point balance
•Exploit NUMA
•Hide memory latency
Eliminate:
•Capacity misses
•Conflict misses
•Compulsory misses
•Write allocate behavior
reorder
unroll & jam
eliminate branches
explicit SIMD
unit-stride streams
memory affinity
SW prefetch
DMA lists
TLB blocking
cache blocking
array padding
streaming stores
compress data
Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011
18-645 – How to Write Fast Code?
Carnegie Mellon University (c) 2012-2014
40

Optimization Categorization

Maximizing In-core Performance	Maximizing Memory Bandwidth	Minimizing Memory Traffic
•Exploit in-core parallelism (ILP, DLP, etc...) •Good (enough) floating-point balance	•Exploit NUMA •Hide memory latency	Eliminate: •Capacity misses •Conflict misses •Compulsory misses •Write allocate behavior



Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

Performance Analysis

- Motivation: Diversity of Computation Platforms
- What is "Fast"?
  - Latency and throughput
  - Task and data
- **The Roofline Model**
  - The Ceilings and the Walls
  - Categories of Optimizations
- Measuring Arithmetic Intensity
- How is this relevant to writing fast code?

Measuring Arithmetic Intensity

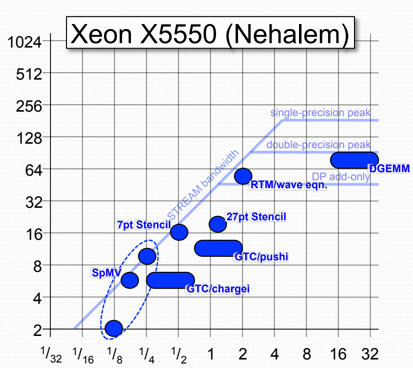
$$\text{Arithmetic Intensity} = \frac{\text{\# of FP Operations to run the program}}{\text{\# of Bytes Accessed in the Main Memory}}$$

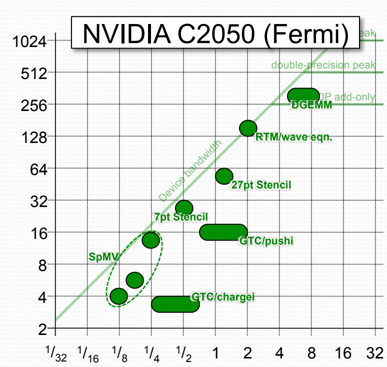
- How to measure # of FP Operations to run the program?
  - This is an assembly level operation

```
# create assembler code:
c++ -S -fverbose-asm -g -O2 test.cc -o test.s
# create asm interlaced with source lines:
as -alhnd test.s > test.lst
```


Example Kernels

- We have examined and heavily optimized a number of kernels and applications for both CPUs and GPUs.
- We observe that for most, performance is highly correlated with DRAM bandwidth – particularly on the GPU





Sam Williams, The Roofline Model, Lawrence Berkeley National Laboratory, 2011

How is this relevant to writing fast code?

Fast Platforms + **Good Techniques**

- **Multicore platforms**
- **Manycore platforms**
- **Cloud platforms**

- **Data structures**
- **Algorithms**
- **Software Architecture**

- How do we know **how much optimization is possible**?
  - Use Roofline model to investigate how much more optimization is possible
  - **Goal: Reduce runtime**

Can You Answer These Questions Now?

- What is the roofline model? What are the metrics and axis used?
- What's the difference between:
  - "flop's per memory instruction" from "flop's per DRAM byte"?
  - Can you imagine an example where the former is much greater than the latter?
  - Can you imagine an example where the latter is much greater than the former
- Consider an image Image[height][width]. If one were to stride through the columns of values, what would be the effects? How would they be mapped to the roofline ?
- How does one model incomplete SIMDization (e.g. half the flop's can be SIMDized), insufficient ILP (some dependent flop's), or an imbalance between FPMUL's and FPADD's on the roofline ?
- How would one model {branch mispredicts, TLB misses, or too many streams for the prefetchers} on the roofline?