Topic 7

Introduction to Multicore II

Xinfei Guo
xinfei.guo@sjtu.edu.cn

July 16th, 2024

T7 learning goals

- Multicore/Multiprocessing
  - Section I
    - Motivation
    - SMT vs. Multicore
    - Communication
    - Cache coherence protocol
  - Section II
    - Memory consistency
    - Parallel programming
    - Dark silicon
    - Roofline model

MEMORY CONSISTENCY

Memory Consistency vs. Coherence

- Coherence is about ordering of operations from different processors to the same memory location
  - Local ordering of accesses to each cache block (so that they do not get stale values)

- Consistency is about ordering of all memory operations from different processors (i.e., to different memory locations)
  - Global ordering of accesses to all memory locations

Memory Consistency

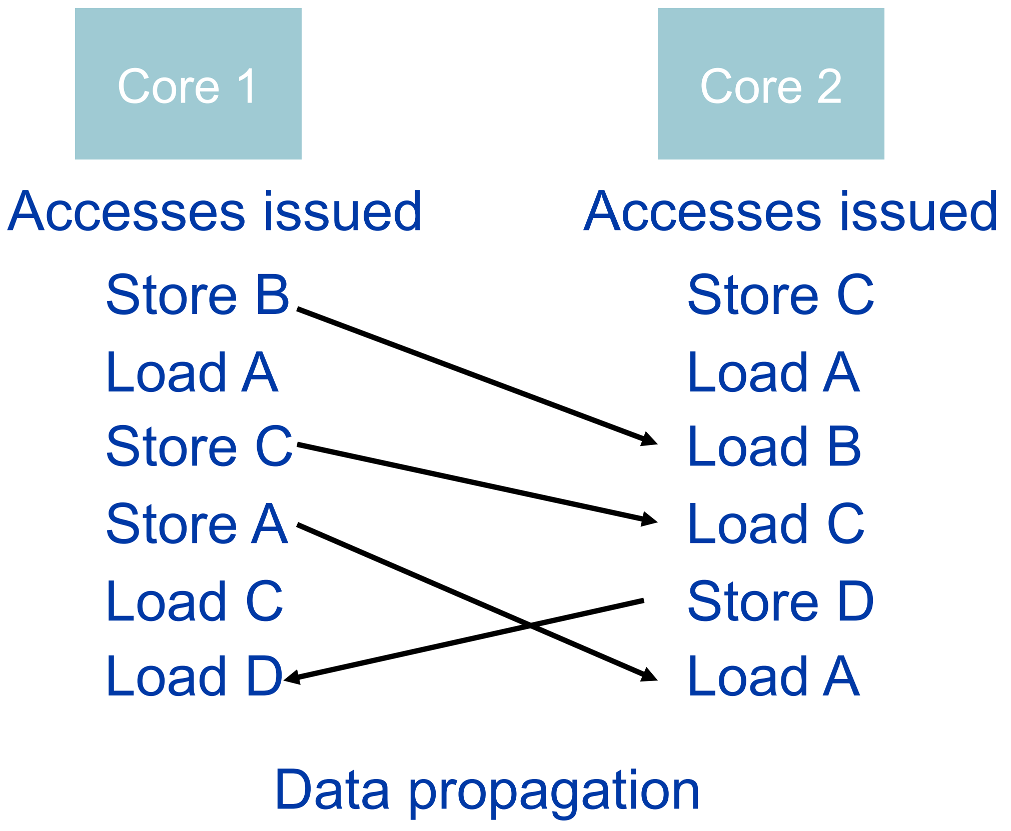
- The purpose of cache coherence is to ensure data propagation and coherence.
  - So when data are written by one core, all other cores can later read the correct value.
  - When a core attempts to write, others know that their copies are stale.
  - When a core attempts to read, others know they must provide their data, if modified.
- The cache-coherence protocol is run **independently** on each block of data.
  - There is **no direct interaction between different blocks**, as far as the protocol is concerned.
- So what about the **order** in which data accesses by one core are seen by others?
  - If a core performs certain reads and writes to different data, in what order do other cores see them?
- It is the purpose of the memory-consistency model to define this.
  - And the job of the memory hierarchy (and core) to implement it

Memory Consistency

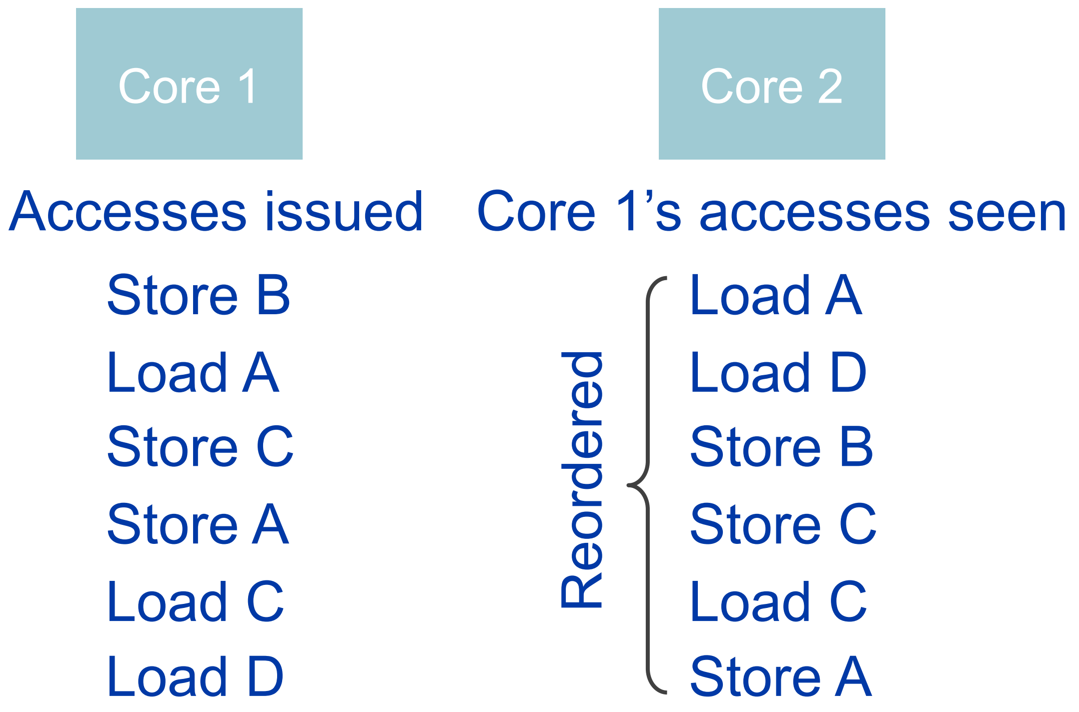
- Modern processors may reorder memory operations.
- Out-of-order processing can allow loads to access the cache ahead of older stores.
  - Either because the addresses they access don't match
  - Or because the load has been speculatively executed and will be replayed later if a dependence is found
- This avoids stalling loads unnecessarily, even though their effects can be seen externally.
  - By other cores in the system

Memory Consistency vs Cache Coherence

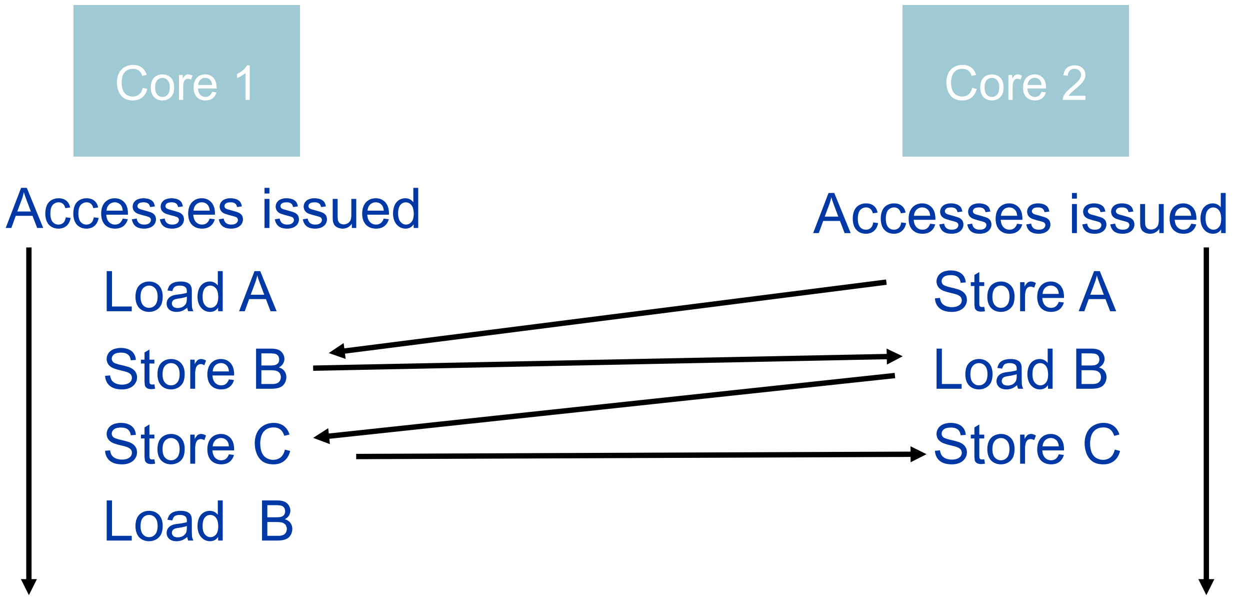
Cache coherence



Memory consistency



Sequential Consistency



Program order + Arbitrary Interleaving

Sequential Consistency

Sequential consistency model: the execution of the threads is equivalent to some sequential interleaving of their operations.

```
// Initially, x = 0

// Thread 1 (P1)
x = 1;
y = x;

// Thread 2 (P2)
x = 2;
z = x;
```

Possible Outcomes:

1. P1's operations occur before P2's operations:
    1. P1: x = 1, y = 1
    2. P2: x = 2, z = 2
2. P2's operations occur before P1's operations:
    1. P2: x = 2, z = 2
    2. P1: x = 1, y = 1
3. Some interleaving of the operations:
    1. P1: x = 1, y = 0 (reads the initial value)
    2. P2: x = 2, z = 2
    3. P1: y = 1

Memory Consistency

- The memory consistency model defines valid outcomes of sequences of accesses of the different cores.
- **Sequential consistency (SC) is the strongest and most intuitive model.**
  - The operations of each core occur in program order, and these are interleaved (at some granularity) across all cores.
  - This means that no loads or stores can bypass other loads or stores.
  - SC is overly strong because it prevents many useful optimizations without being needed by most programs.
- **Total store order (TSO) is widely implemented (e.g., x86 architectures).**
  - The same as sequential consistency but allows a younger load to observe a state of memory in which the effects of an older store have not yet become observable
- **Forms of relaxed consistency have been adopted (e.g., Arm and PowerPC architectures).**
  - In more relaxed consistency models, other constraints in SC are removed, such as a younger load observing a state of memory before an older load does.

Total store order (TSO) Consistency

```
// Initially, x = y =
0

// Thread 1
x = 1;
y = 2;

// Thread 2
if (y == 2) {
    result = x;
}
```

The TSO model allows for some reordering of memory operations but maintains a total order of store operations.

Possible Outcomes:

1. Thread 1's stores occur before Thread 2's load:
   Thread 1: x = 1, y = 2
   Thread 2: result = x (result = 1)
2. Thread 2's load occurs before Thread 1's stores:
   Thread 2: result = x (result = 0, as x is still 0 when Thread 2 loads)
   Thread 1: x = 1, y = 2

JOINT INSTITUTE
交大密西根学院

PARALLEL PROGRAMMING

Parallel Programming

Q: So lets just all use multicore from now on!

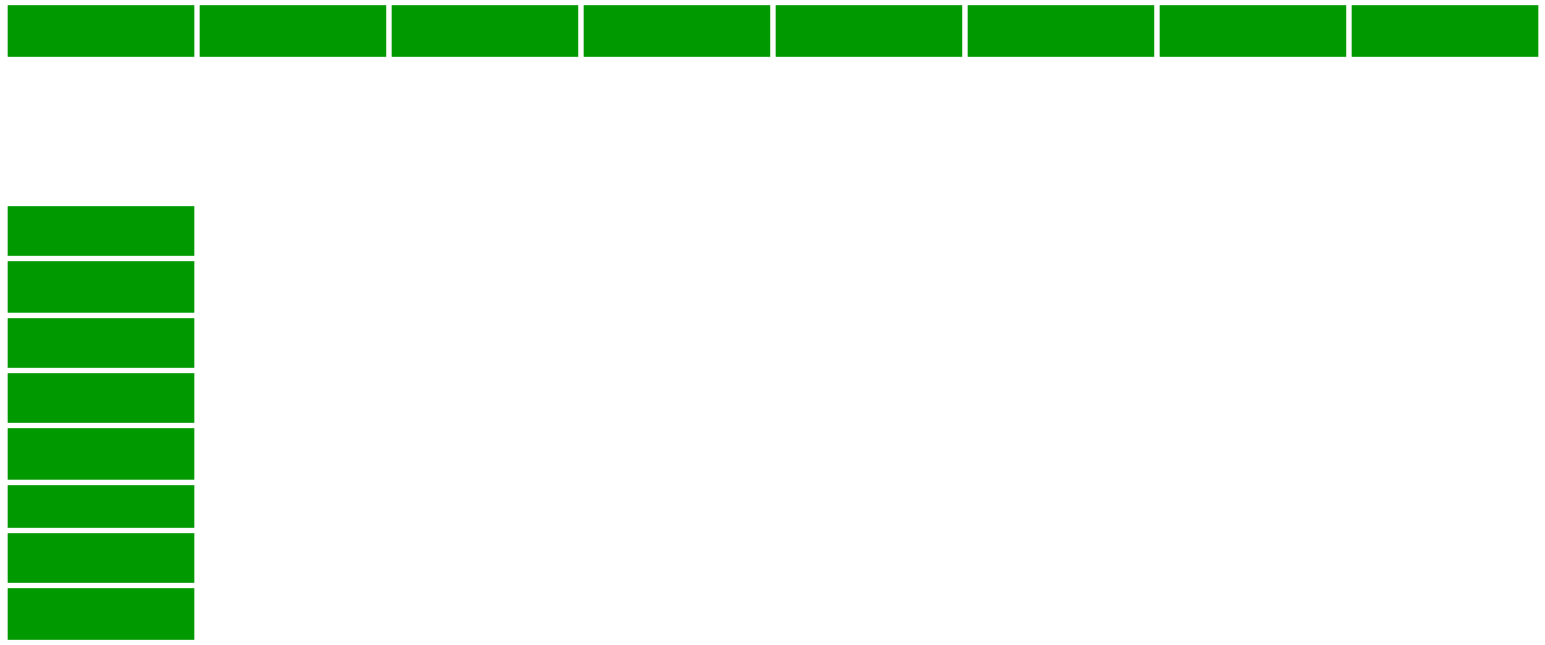
A: Software must be written as parallel program

Multicore difficulties

- **Partitioning work**
- **Concurrency poses new challenges**

Work Partitioning

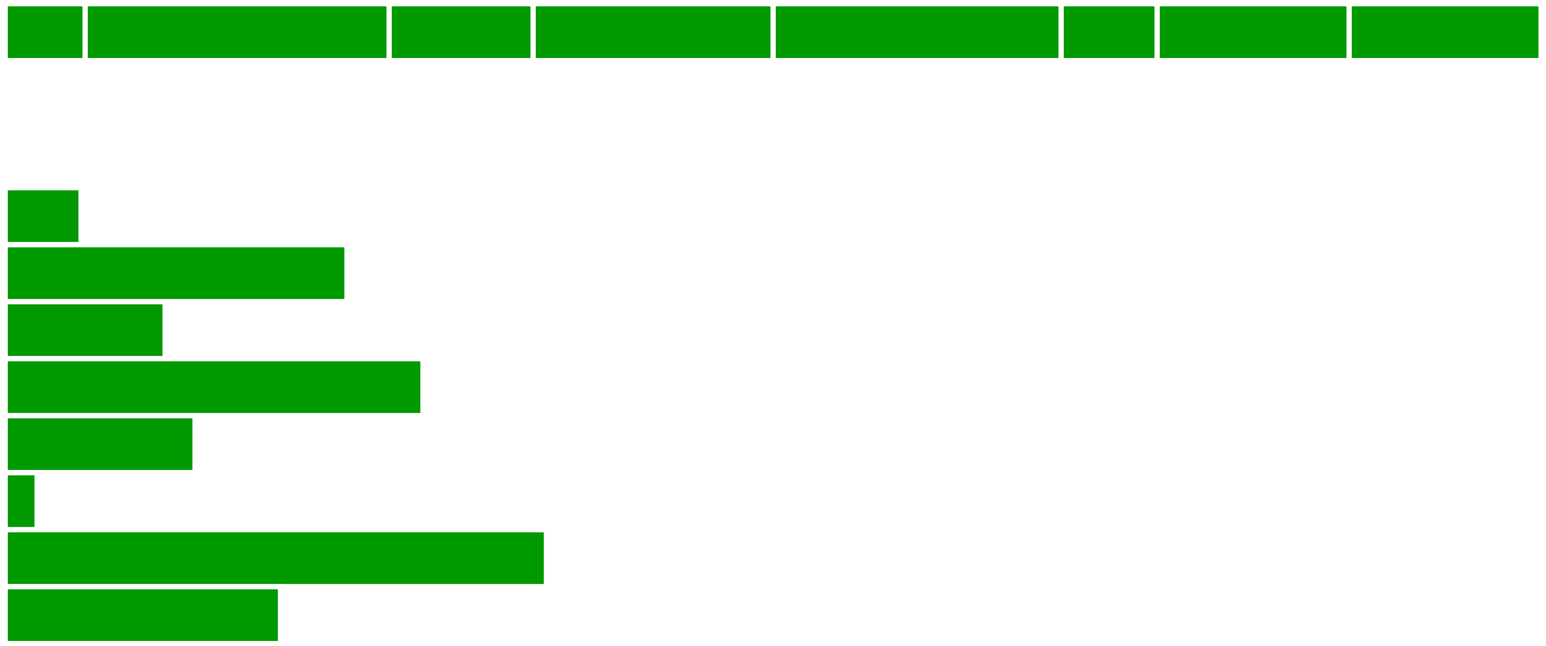
Partition work so all cores have something to do



Load Balancing

Load Balancing

Need to partition so all cores are actually working



Amdahl's Law

If tasks have a serial part and a parallel part…

Example:

step 1: divide input data into *n* pieces

step 2: do work on each piece

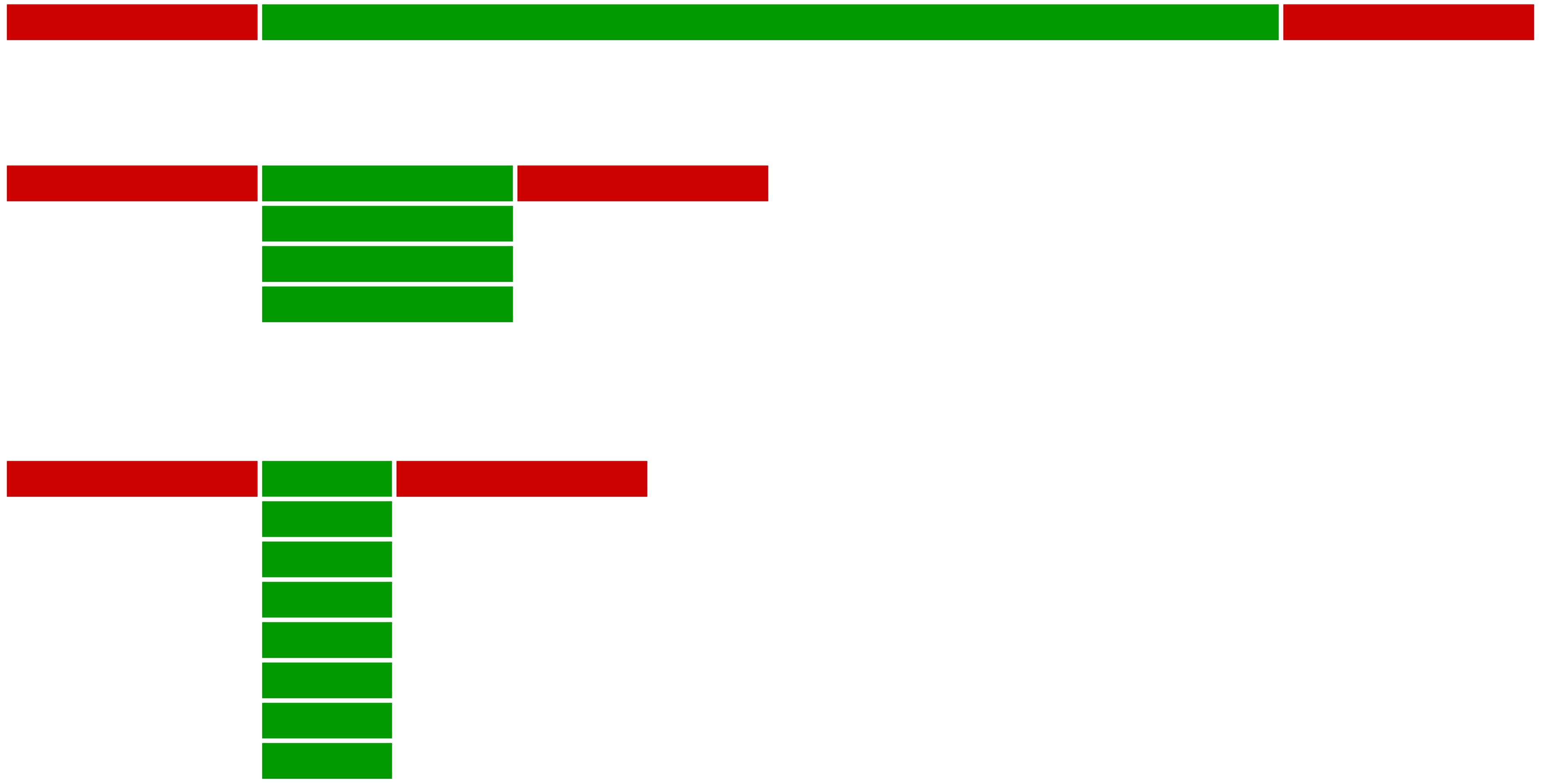
step 3: combine all results

Recall: Amdahl's Law

As number of cores increases …

- time to execute parallel part? goes to zero
- time to execute serial part? Remains the same
- *Serial part eventually dominates*

Amdahl’s Law



Programming with Threads

Concurrency poses challenges for:

Correctness

- Threads accessing shared memory should not interfere with each other

Liveness

- Threads should not get stuck, should make forward progress

Efficiency

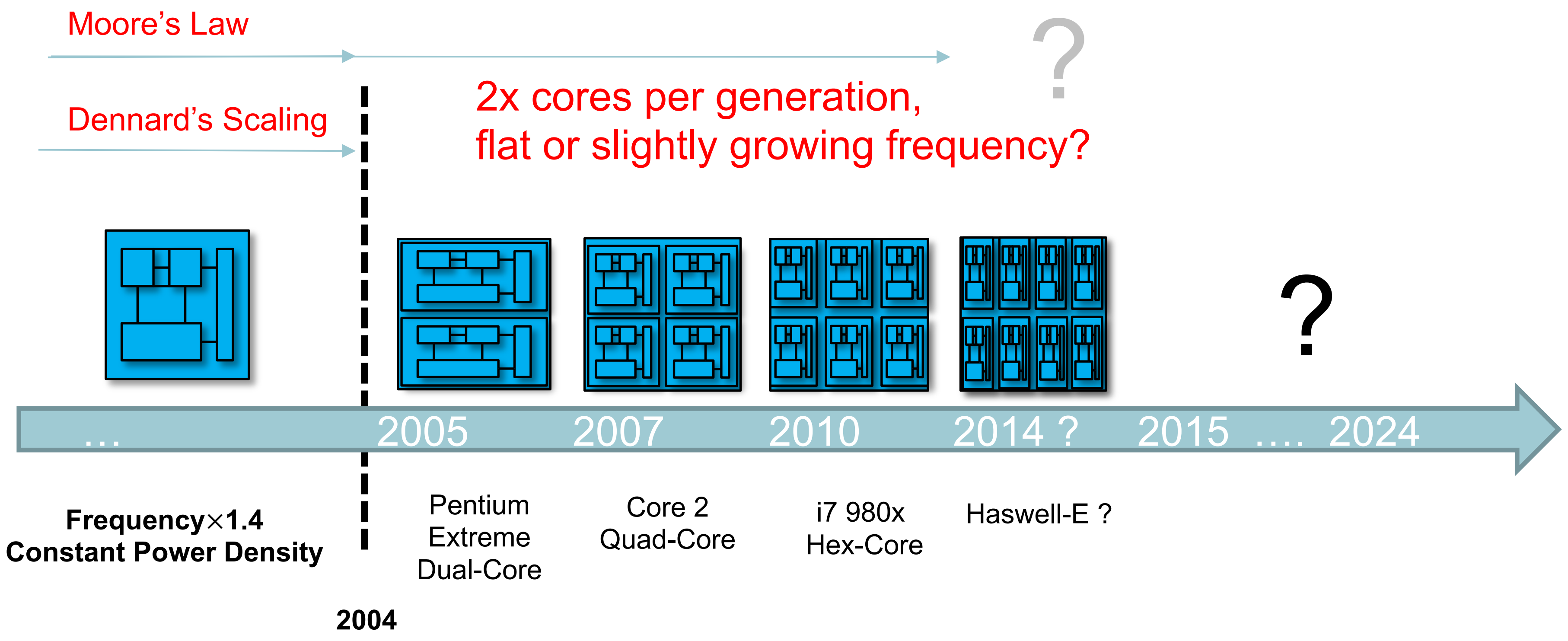
- Program should make good use of available computing resources (e.g., processors).

Fairness

- Resources apportioned fairly between threads

DARK SILICON

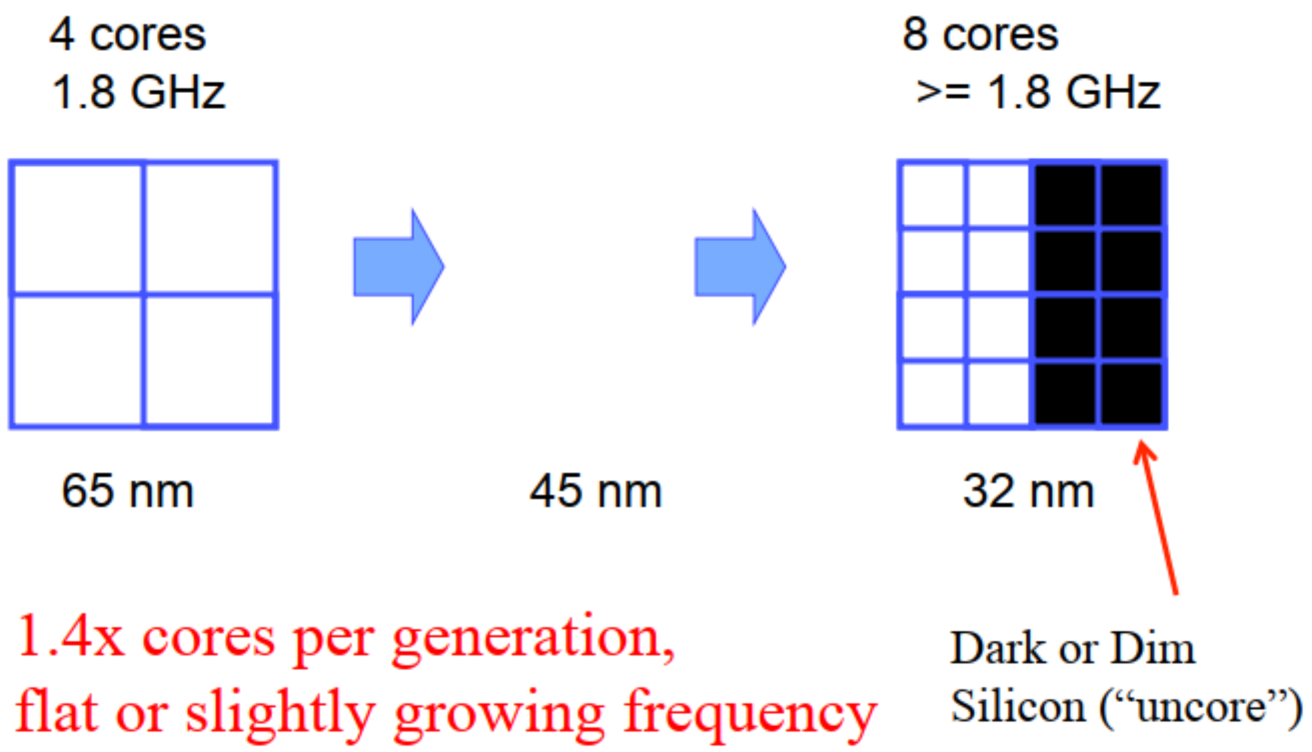
Multicore Era?



Reference: Karu Sankaralingam, "The Dark Silicon Implications for Microprocessors" Presentation

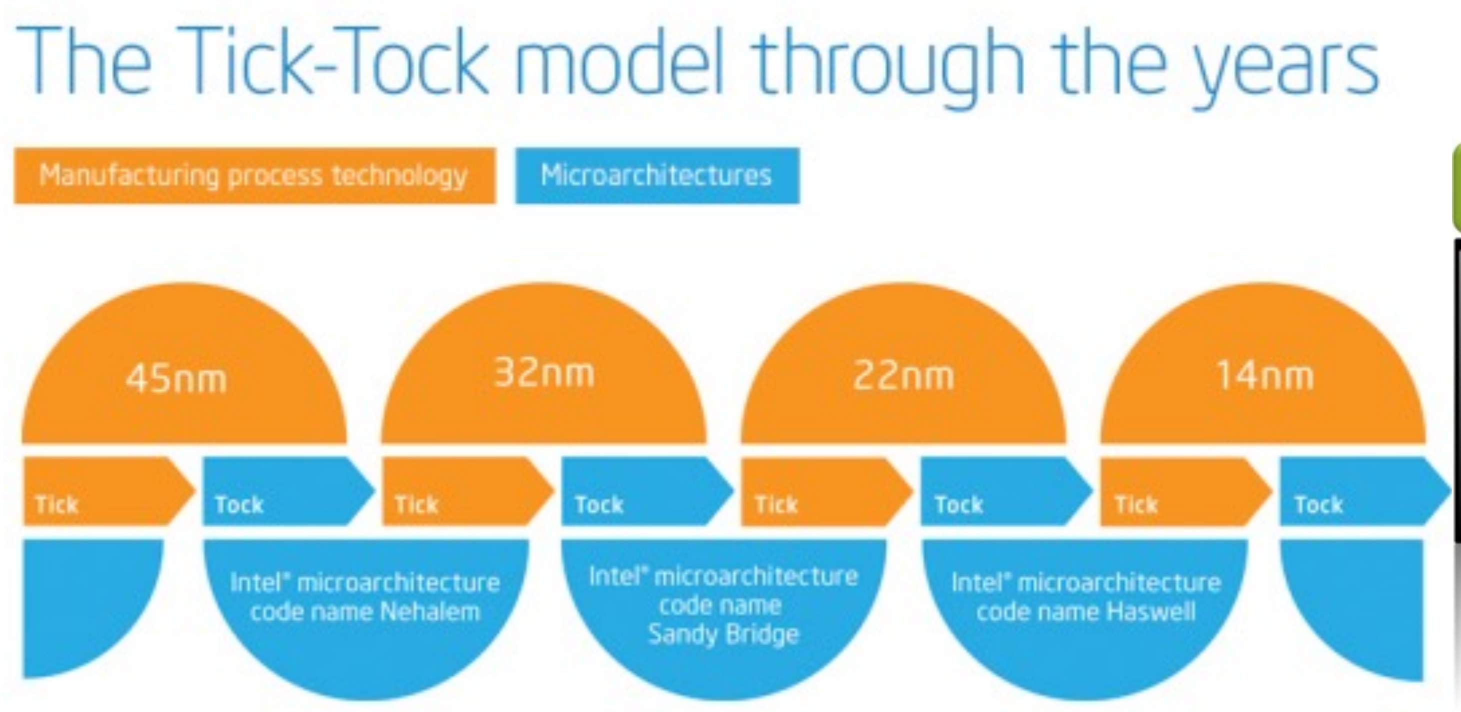
The truth

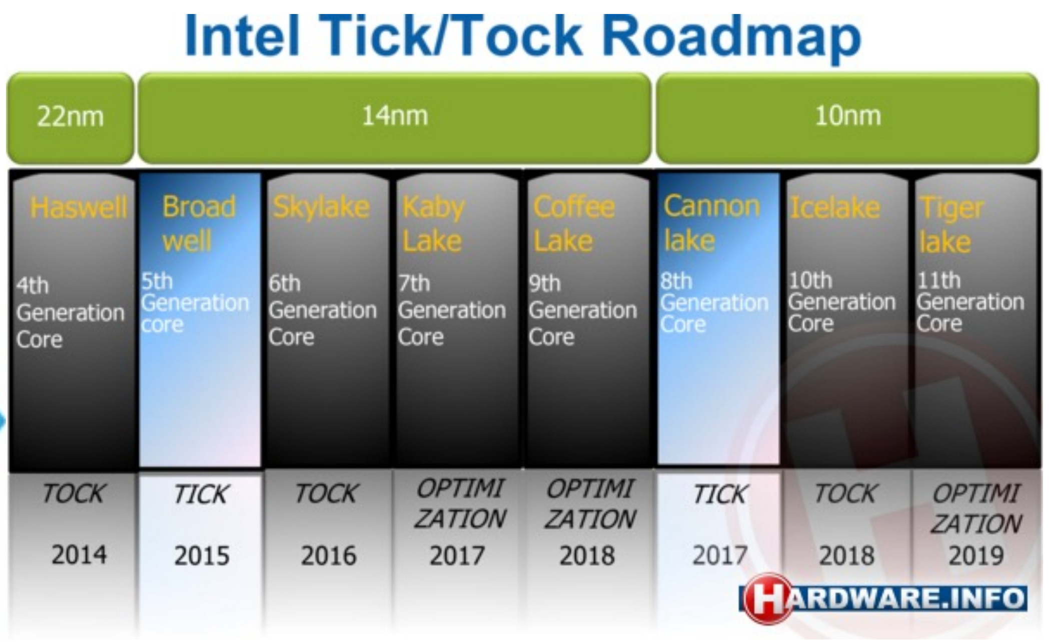
- But actually,
  that's not what's happening



With each successive process generation, the percentage of a chip that can switch at full frequency drops exponentially due to power constraints

What Used to Be: Intel Tick-Tock Model


The Tick-Tock model through the years
Manufacturing process technology
Microarchitectures
45nm
32nm
22nm
14nm
Tick
Tock
Tick
Tock
Tick
Tock
Tick
Tock
Intel® microarchitecture code name Nehalem
Intel® microarchitecture code name Sandy Bridge
Intel® microarchitecture code name Haswell



What happened (2019):
In 2016, Intel deprecated tick-tock in favor of "process–architecture–optimization"
Skylake (14nm) now has five planned optimizations (Skylake, Kaby Lake, Coffee Lake, and Whiskey Lake) with no 10nm chip in production yet

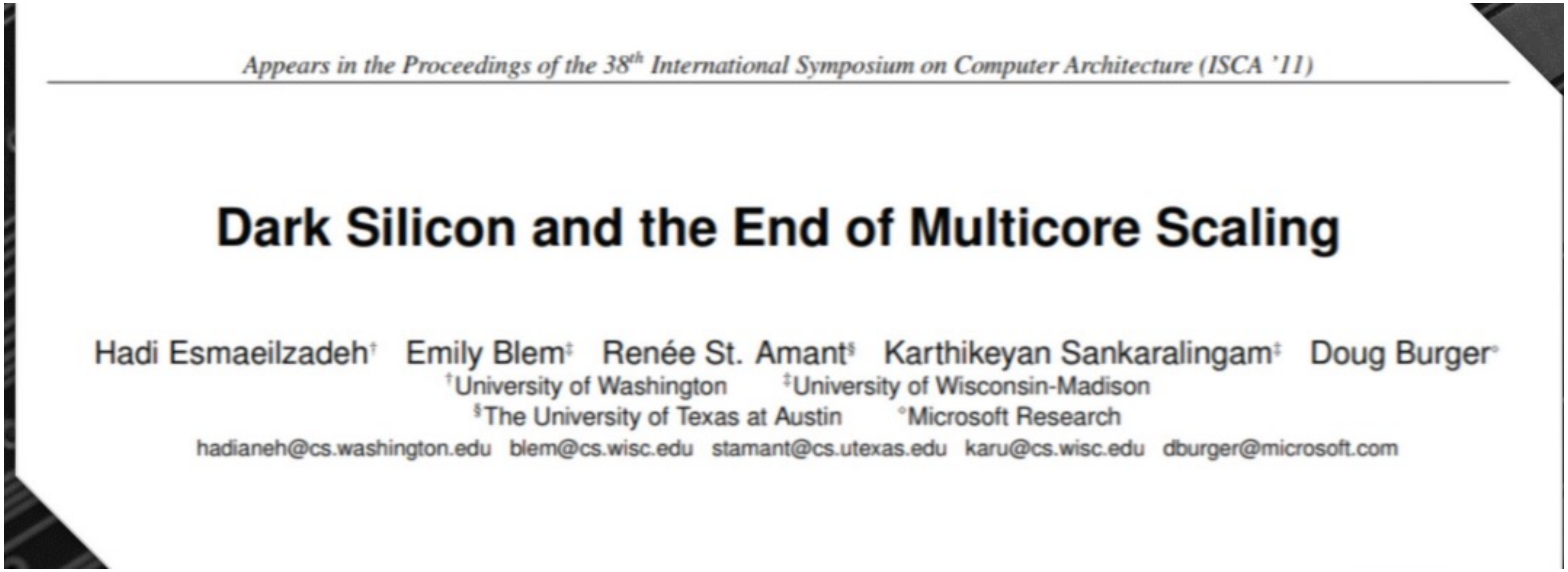
Slide Credit: Prof. Sang-Woo Jun @ UCI

Dark Silicon

- Dark Silicon – […]the amount of circuitry of an integrated circuit that cannot be powered-on at the nominal operating voltage for a given thermal design power (TDP) constraint.
  - Wikipedia

- Dark Silicon – Transistors which suffer from underutilization

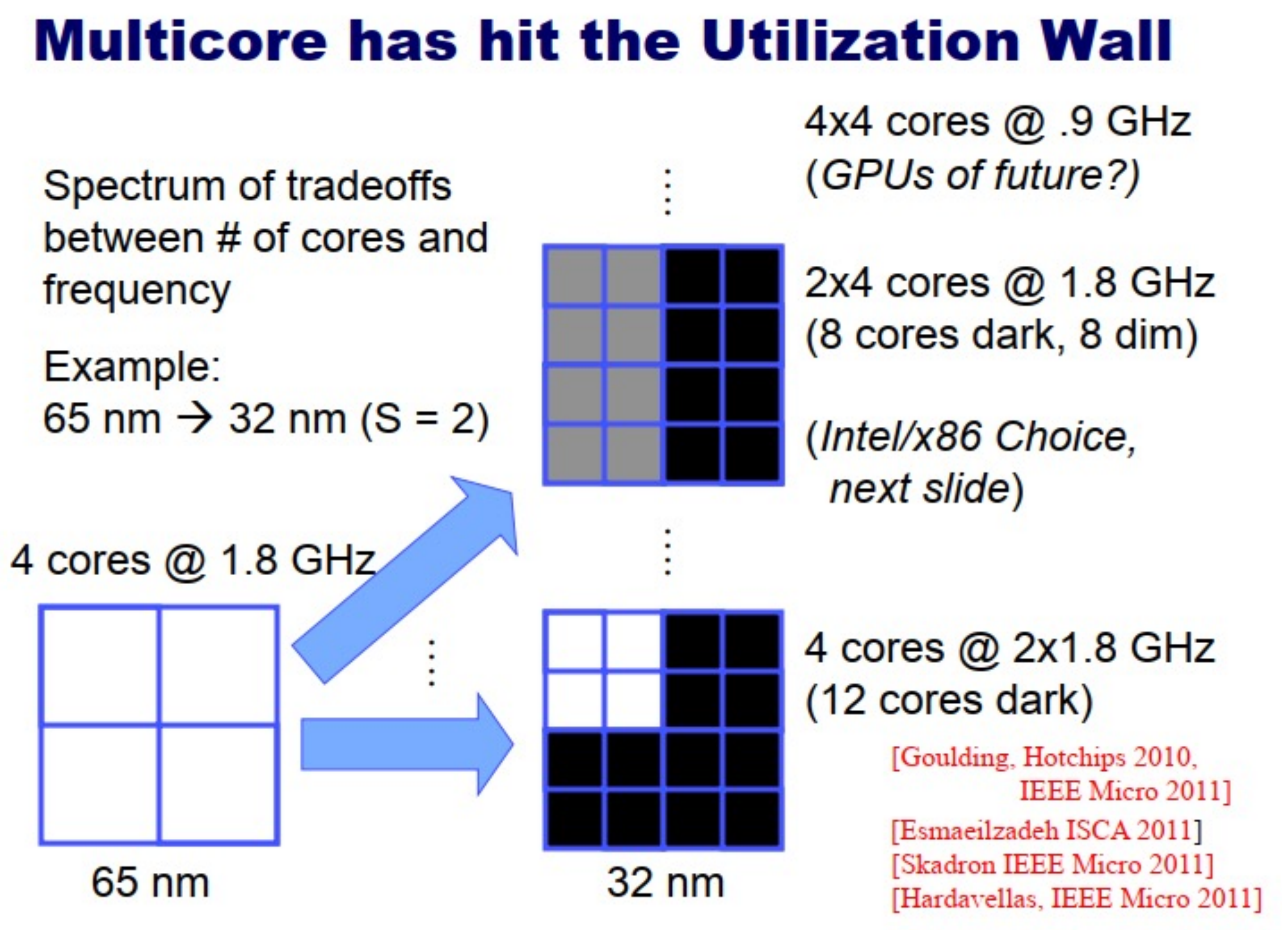
Dark Silicon


Appears in the Proceedings of the 38th International Symposium on Computer Architecture (ISCA '11)

Dark Silicon and the End of Multicore Scaling

Hadi Esmaeilzadeh[†] Emily Blem[‡] Renée St. Amant[§] Karthikeyan Sankaralingam[‡] Doug Burger[°]
[†]University of Washington [‡]University of Wisconsin-Madison
[§]The University of Texas at Austin [°]Microsoft Research
hadianeh@cs.washington.edu blem@cs.wisc.edu stamant@cs.utexas.edu karu@cs.wisc.edu dburger@microsoft.com

Dark Silicon



Multicore has hit the Utilization Wall

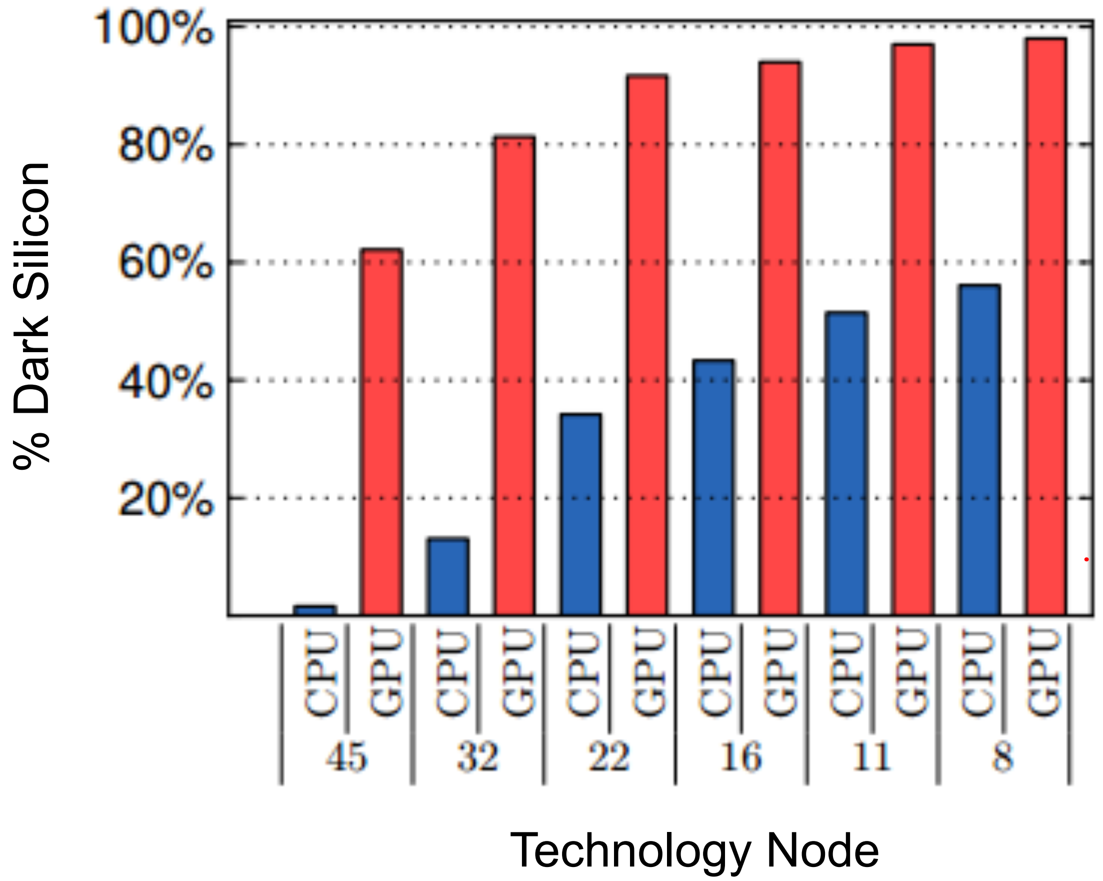
Spectrum of tradeoffs between # of cores and frequency

Example:
65 nm → 32 nm (S = 2)

[Goulding, Hotchips 2010, IEEE Micro 2011]
[Esmaeilzadeh ISCA 2011]
[Skadron IEEE Micro 2011]
[Hardavellas, IEEE Micro 2011]

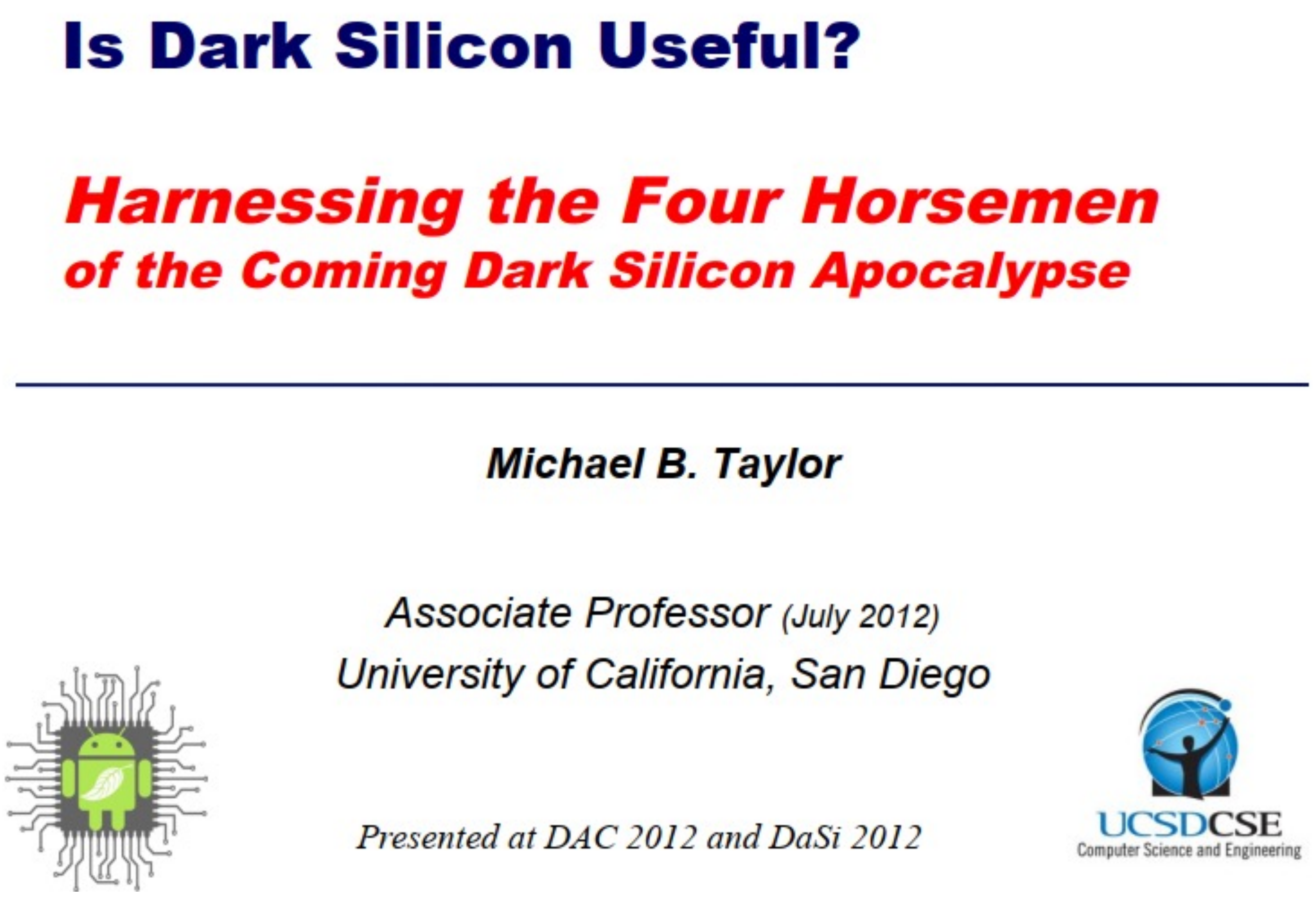
source: Prof. Michael Taylor @ UW

Percentage of Dark Silicon



source: Esmaeilzadeh, Hadi, et al. "Dark silicon and the end of multicore scaling." 2011 38th Annual international symposium on computer architecture (ISCA). IEEE, 2011.

Is Dark Silicon Useful?



Is Dark Silicon Useful?

Harnessing the Four Horsemen
of the Coming Dark Silicon Apocalypse

Michael B. Taylor

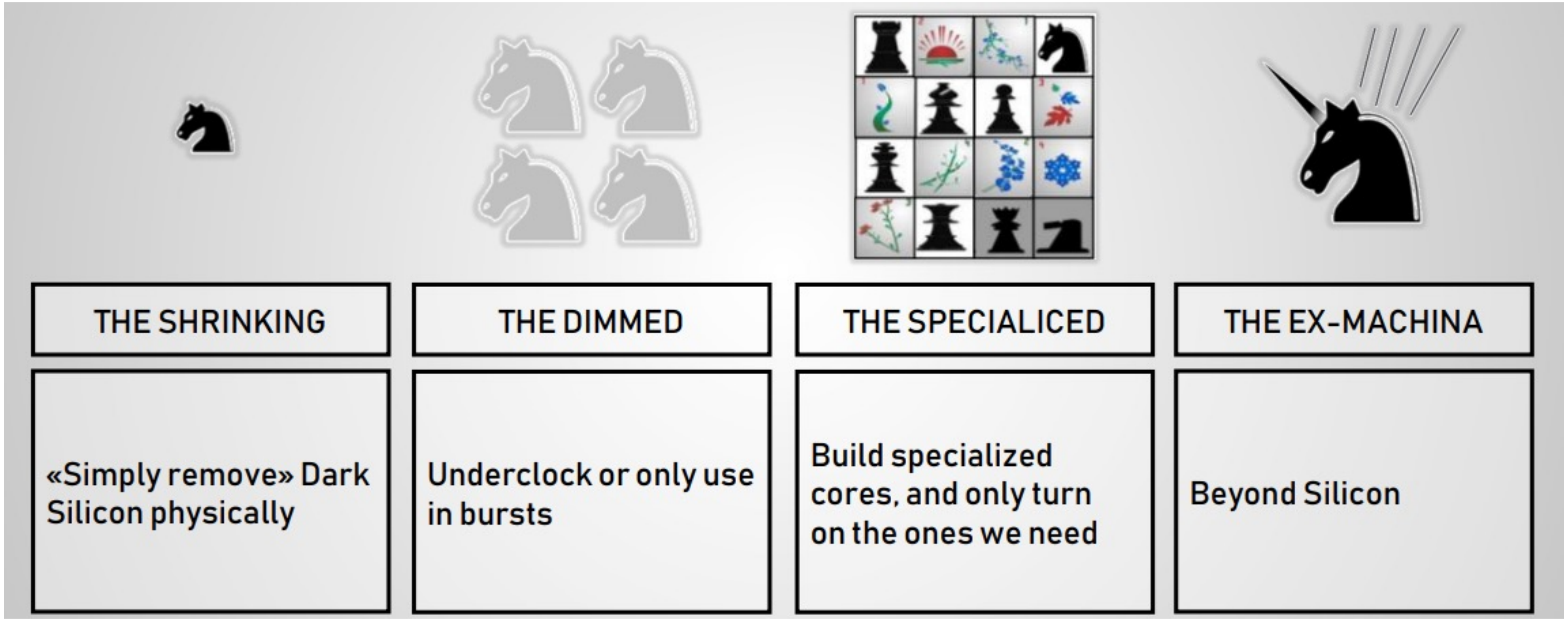
Associate Professor (July 2012)
University of California, San Diego

Presented at DAC 2012 and DaSi 2012

UCSDCSE
Computer Science and Engineering

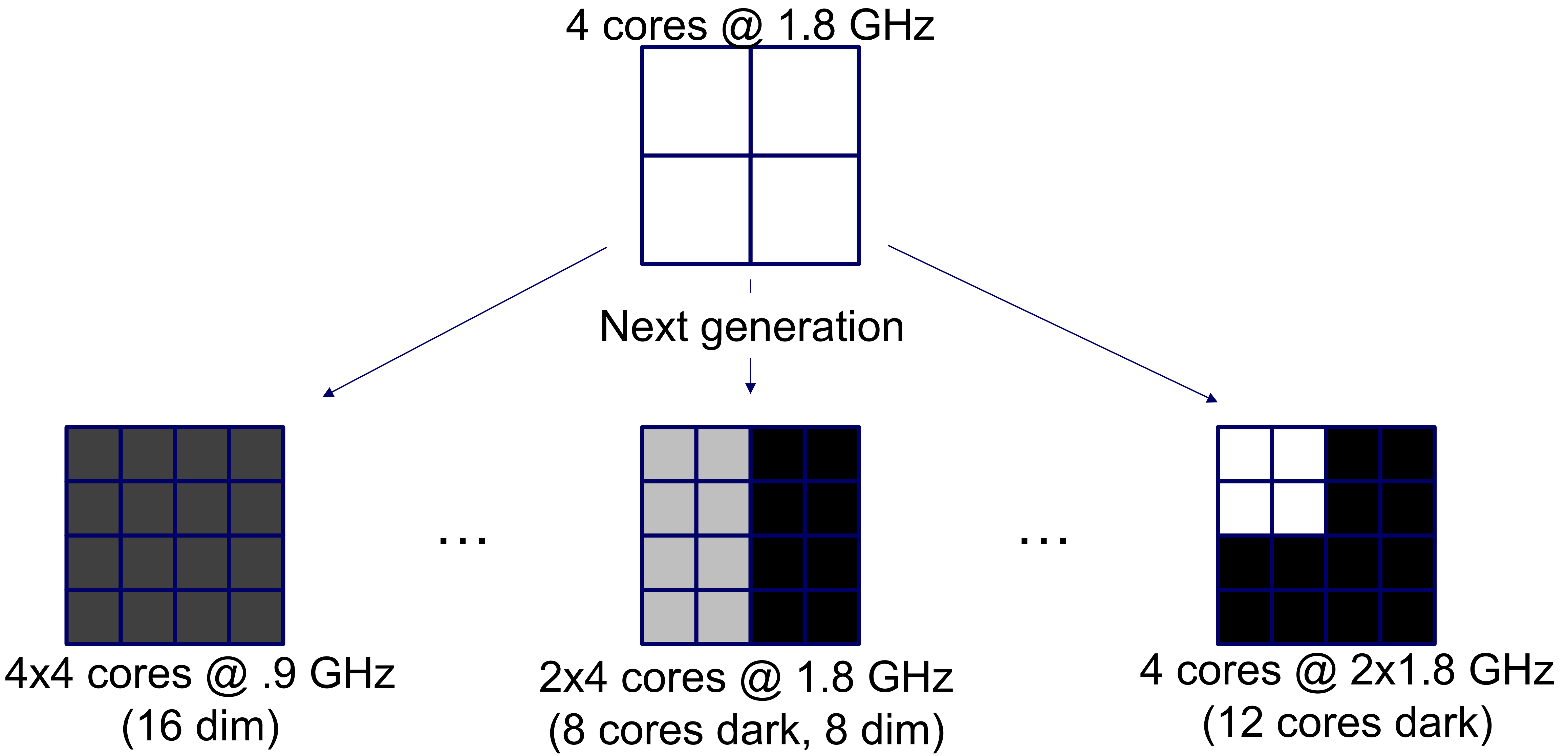
source: https://www.bsg.ai/papers/taylor_dark_silicon_horsemen_dac_2012.pdf

4 horsemen



THE SHRINKING	THE DIMMED	THE SPECIALICED	THE EX-MACHINA
«Simply remove» Dark Silicon physically	Underclock or only use in bursts	Build specialized cores, and only turn on the ones we need	Beyond Silicon

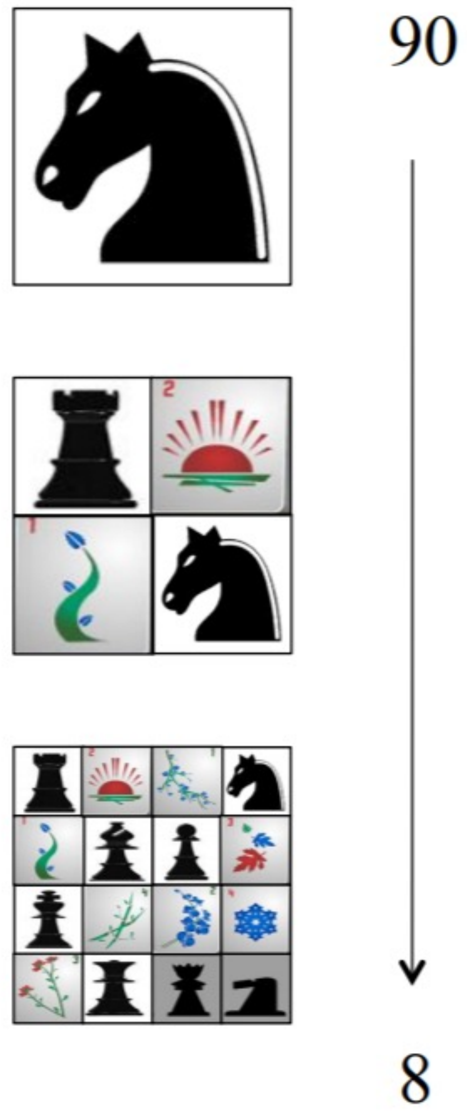
Dim Silicon



Slide Credit: Prof. Sang-Woo Jun @ UCI

The Specialized Horseman

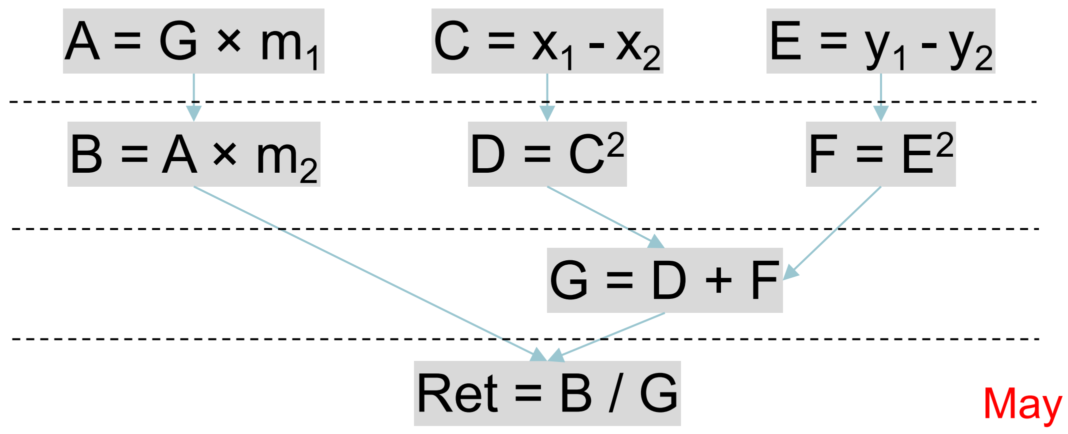
- "We will use all of that dark silicon area to build specialized cores, each of them tuned for the task at hand (10-100x more energy efficient), and only turn on the ones we need…"
- Insights:
  - Power is now more expensive than area
  - Specialized logic can improve energy efficiency by 10-1000x
- This led to today's heterogeneous systems



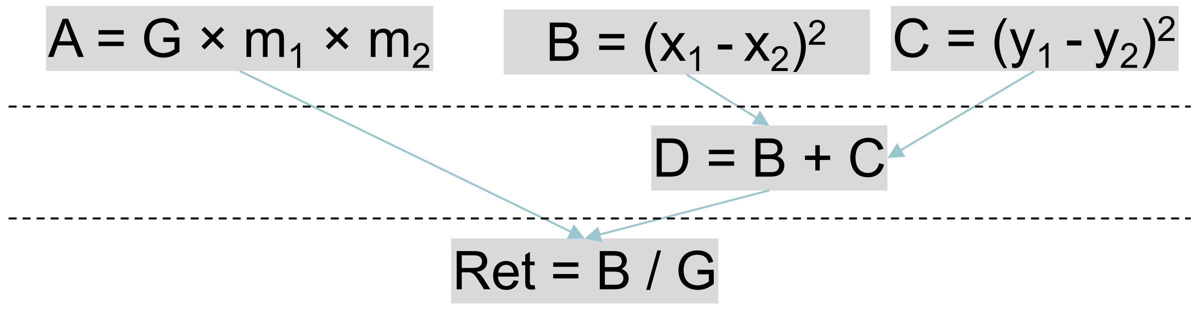
source: Prof. Michael Taylor @ UW

Fine-Grained Parallelism of Special-Purpose Circuits

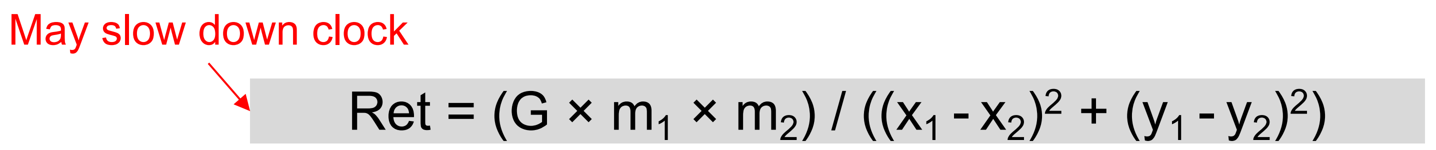
- Example -- Calculating gravitational force: $\frac{G \times m_1 \times m_2}{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- 8 instructions on a CPU → 8 cycles**
- Much fewer cycles on a special purpose circuit



4 cycles with basic operations



3 cycles with compound operations

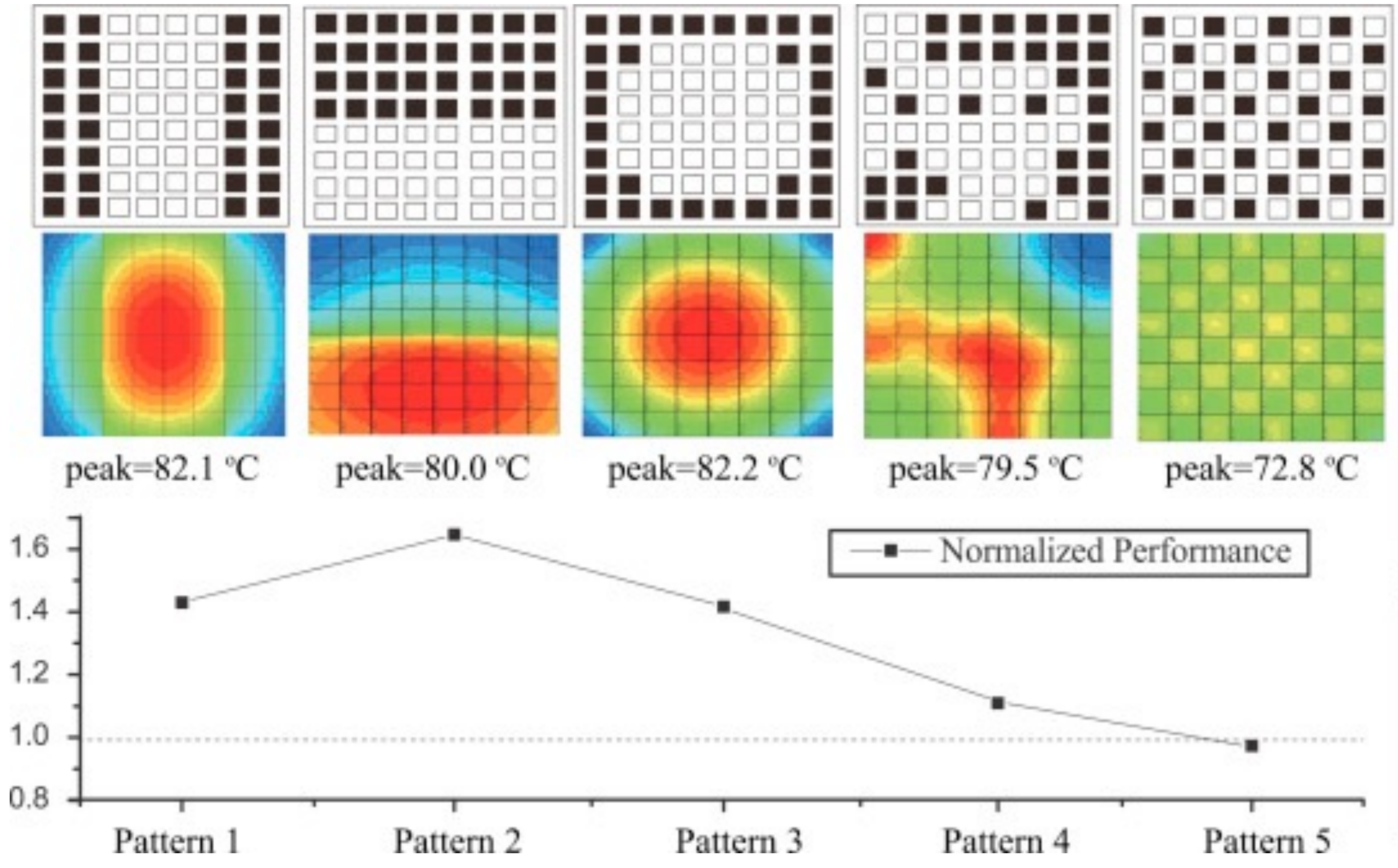


May slow down clock
$Ret = (G \times m_1 \times m_2) / ((x_1 - x_2)^2 + (y_1 - y_2)^2)$

1 cycle with even further compound operations

Slide Credit: Prof. Sang-Woo Jun @ UCI

Dark Silicon Pattern



8*8 Alpha

21264 core array

Tsafe = 81.8 °C;

Number of active

cores = 32;

22 nm technique;

@2.6 Ghz,

64GB Memory.

Map: HotSpot [8]

Power:McPAT[9]

Temp: MatEx [38]

source: Shafique, Muhammad, et al. "Dark silicon as a challenge for hardware/software co-design: Invited special session paper." Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis. 2014.

Where Do GPUs Fit into This?

- GPUs have hundreds of threads running at multiple gigabytes!
- GPU runs much slower clock than CPU
- Much simpler processor architecture
  - Dozens of threads scheduled together in a SIMD fashion
  - Much simpler microarchitecture (doesn't need to boot Linux!)
- Much higher power budget
  - CPUs try to maintain 100 W power budget (Pentium 4)
  - GPUs regularly exceed 400 W

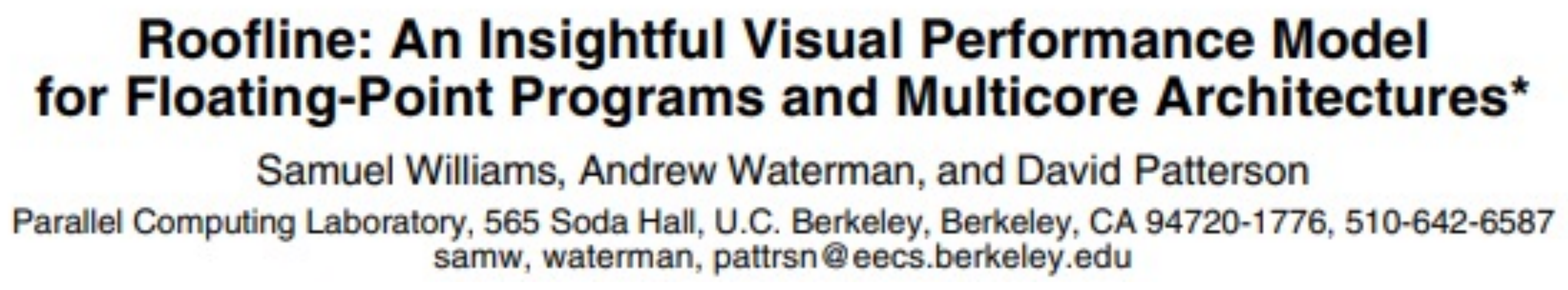
Slide Credit: Prof. Sang-Woo Jun @ UCI

ROOFLINE MODEL

Roofline Model

- In any program, we need to do
  - Bring data from memory
  - Perform computations on the data
- The time consumed by the program is the higher of the two
- This intuition is captured by the roofline model
- Roofline model is unique to the **hardware** (CPU, GPU, etc.)

Roofline Model


Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures*

Samuel Williams, Andrew Waterman, and David Patterson

Parallel Computing Laboratory, 565 Soda Hall, U.C. Berkeley, Berkeley, CA 94720-1776, 510-642-6587
samw, waterman, pattrsn@eecs.berkeley.edu

- Goal: integrate in-core performance, memory bandwidth, and locality into a single readily understandable performance figure
- Bound and bottleneck analysis
- Roofline model will be unique to each architecture

source:
https://people.eecs.berkeley.edu/~kubitron/cs252/handouts/papers/RooflineVyNoYellow.pdf
https://crd.lbl.gov/assets/pubs_presos/parlab08-roofline-talk.pdf

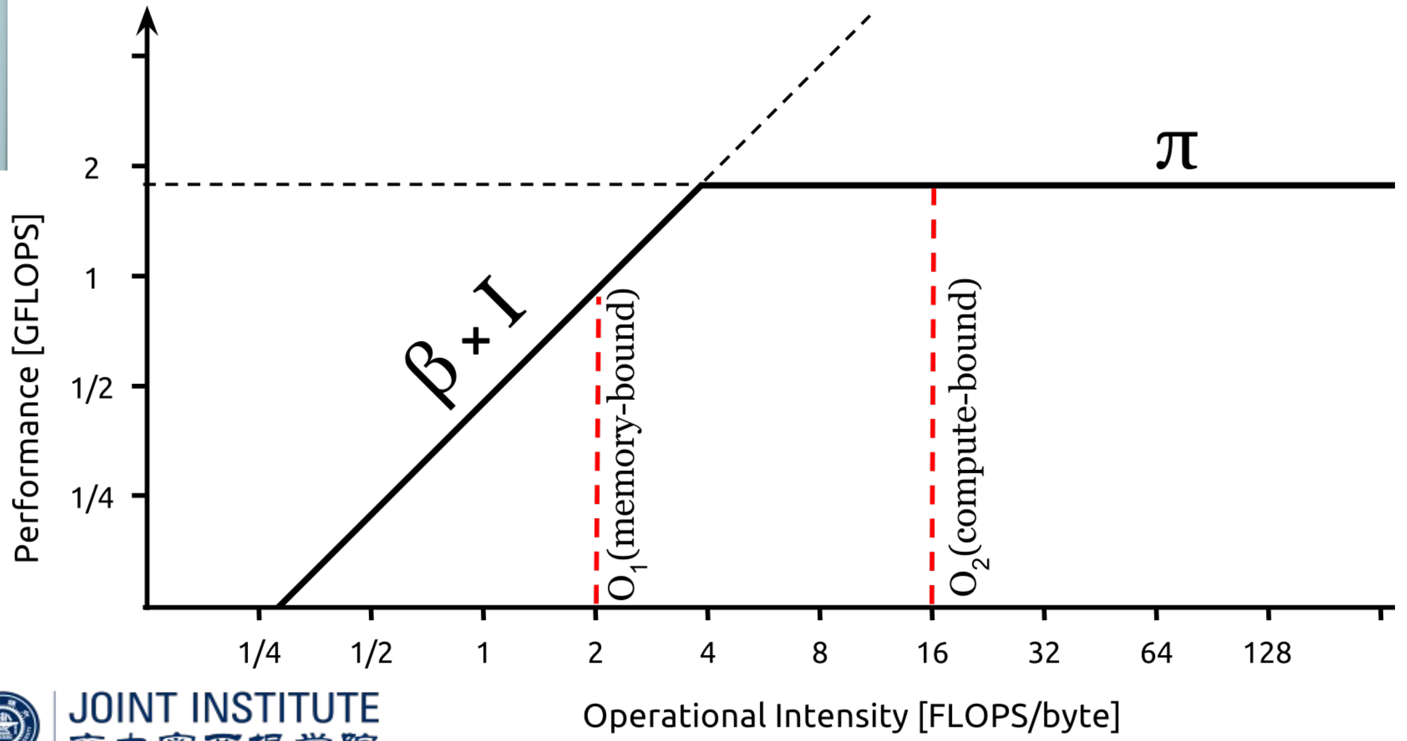
Roofline Model

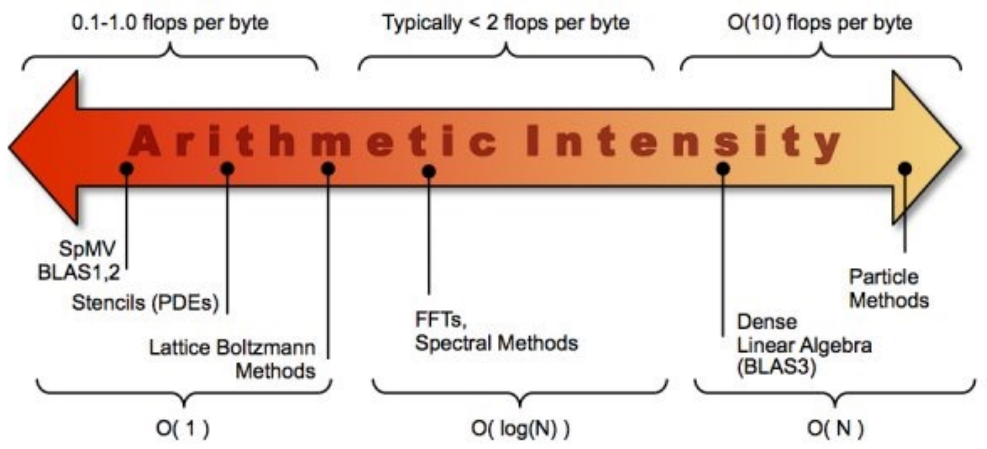
- Performance is upper bounded by both the peak flop rate, and the product of streaming bandwidth and the flop:byte ratio

$$\text{Gflop/s} = \min \begin{cases} \text{Peak Gflop/s} \\ \text{Stream BW} * \underline{\text{actual flop:byte ratio}} \end{cases}$$

Max computation performance

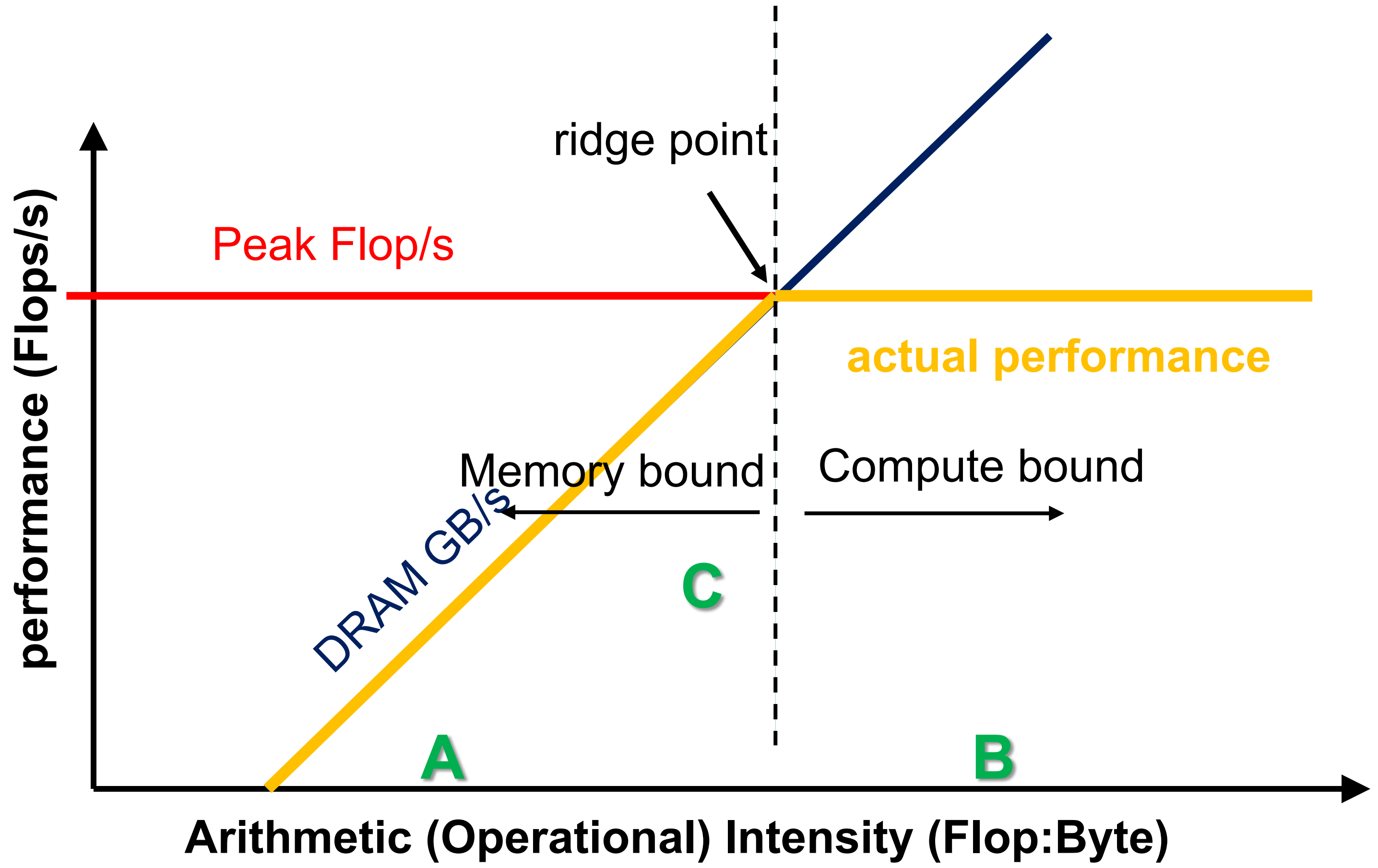
Arithmetic (Operational) Intensity: A metric for the application





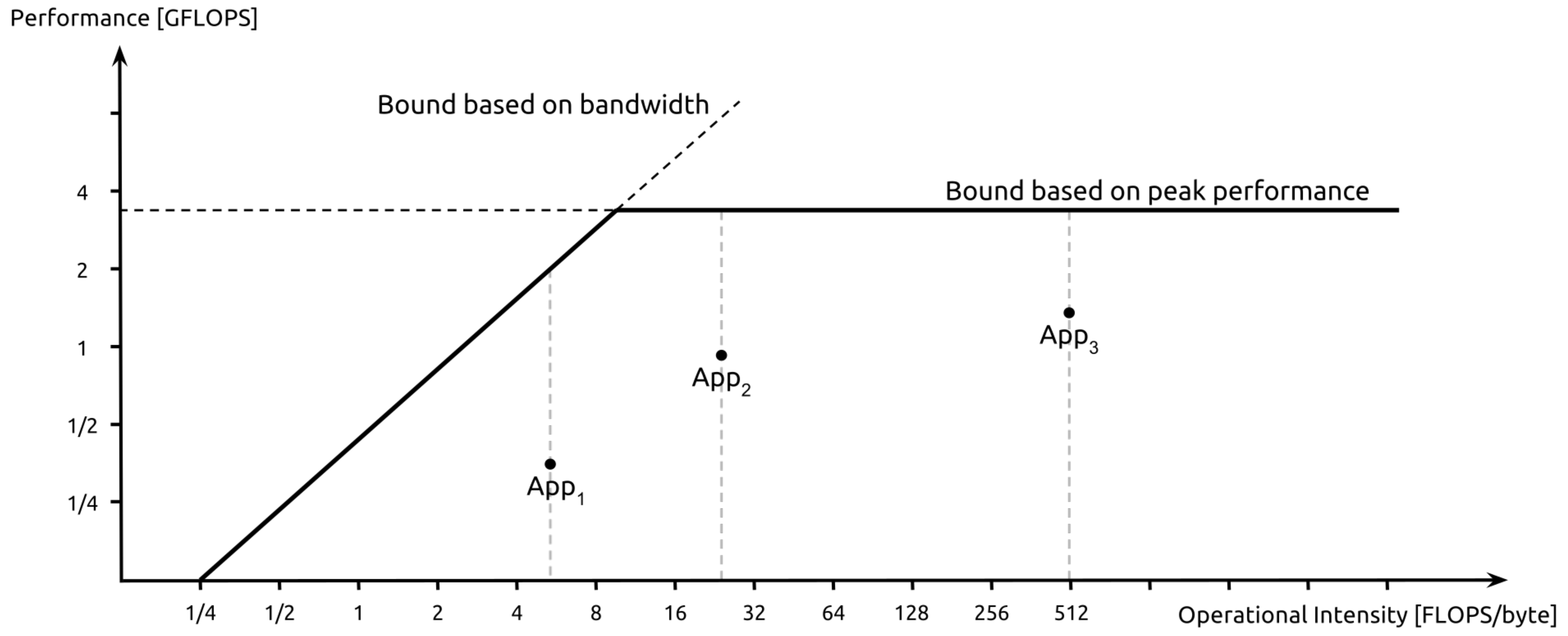
source: Wiki

Insights (Example)



Roofline Model

the processor's peak performance and a ceiling derived from the memory bandwidth.



operational intensity: ratio of the work W to the memory traffic Q, denotes the number of operations per byte of memory traffic

source: Wiki

Parallelism is a necessity

Necessity, but there is always

- Power wall

Not easy to get performance out of Many solutions

- Pipelining
- Multi-issue
- Hyperthreading
- Multicore

Need to tailor the program to support

Where are we Heading?

- T8: Specialized Architectures

Acknowledgement

Slides in this topic are inspired in part by material developed and copyright by:

- ARM Courseware
- Nvidia Courseware
- Prof. Onur Mutlu @ ETH
- Prof. Joe Devietti @ Upenn, CIS 571
- Prof. Hakim Weatherspoon @ Cornell, CS 3410
- Prof. Krste Asanovic @ UCB, CS252
- Xinfei Guo @ JI, VE370 2021 SU

Action Items

- Final project
- Reading Materials
  - Ch. 4.3, 5.5, 5.6
  - Dark Silicon Paper
  - Roofline Paper