Accelerator-level Parallelism

Mark D. Hill, Wisconsin & Vijay Janapa Reddi, Harvard

@ Technion (Virtually), June 2020

Aspects of this work on Mobile SoCs and Gables were developed while the authors were “interns” with Google’s Mobile Silicon Group. Thanks!

Accelerator-level Parallelism Call to Action

Future apps demand much more computing

Standard tech scaling & architecture NOT sufficient

Mobile SoCs show a promising approach:

ALP = Parallelism among workload components concurrently executing on multiple accelerators (IPs)

Call to action to develop "science" for ubiquitous ALP

Outline

I. Computer History & X-level Parallelism

II. Mobile SoCs as ALP Harbinger

III. Gables ALP SoC Model

IV. Call to Action for Accelerator-level Parallelism

20th Century Information & Communication Technology

Has Changed Our World

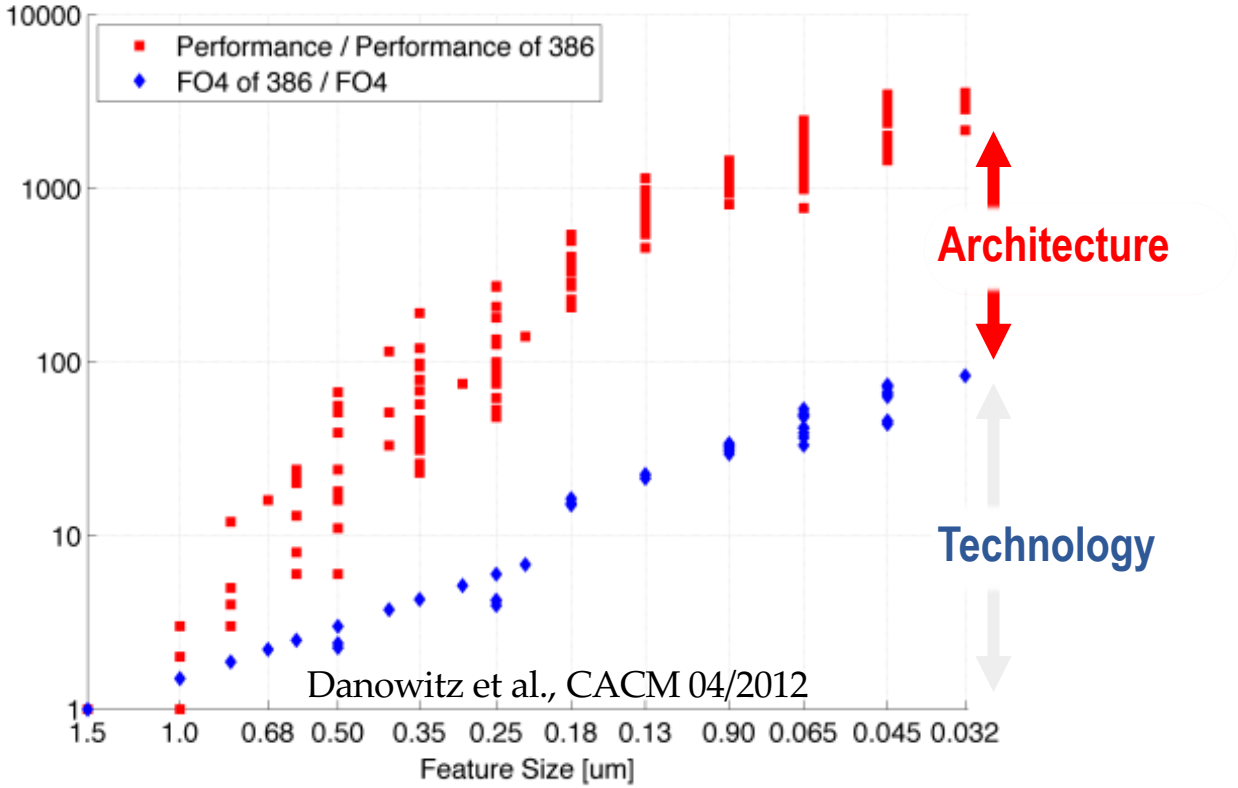
- <long list omitted>

Required innovations in algorithms, applications, programming languages, … , & system software

Key (invisible) enablers (cost-)performance gains

- Semiconductor technology (“Moore’s Law”)
- Computer architecture (~80x per Danowitz et al.)

Enablers: Technology + Architecture



How did Architecture Exploit Moore's Law?

MORE (& faster) transistors ➔ even faster computers

Memory – transistors in parallel

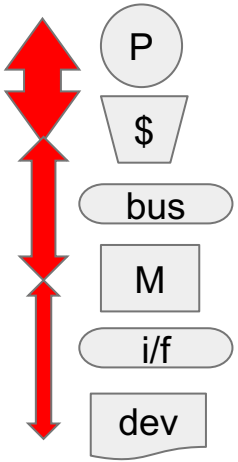
- Vast semiconductor memory (DRAM)
- Cache hierarchy for fast memory illusion

Processing – transistors in parallel

Bit-, Instruction-, Thread-, & Data-level Parallelism

Now Accelerator-level Parallelism

X-level Parallelism in Computer Architecture



1 CPU

BLP+ILP

Bit/Instrn-Level Parallelism

Bit-level Parallelism (BLP)

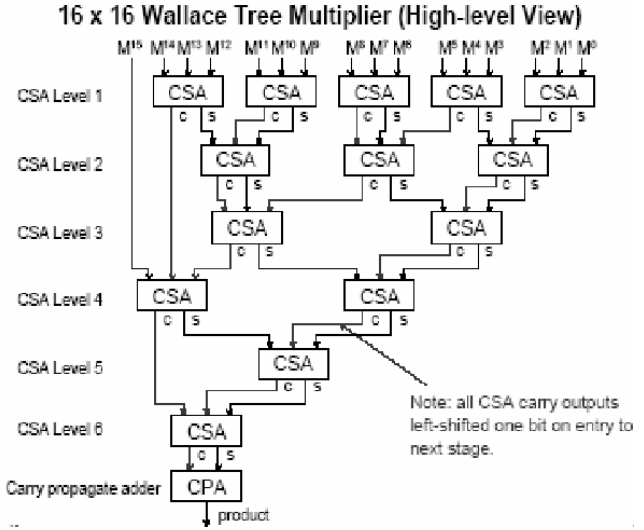
Early computers: few switches (transistors)

- ➔ compute a result in many steps
- E.g., 1 multiplication partial product per cycle

Bit-level parallelism

- More transistors ➔ compute more in parallel
- E.g., Wallace Tree multiplier (right)

Larger words help: 8b→16b→32b→64b

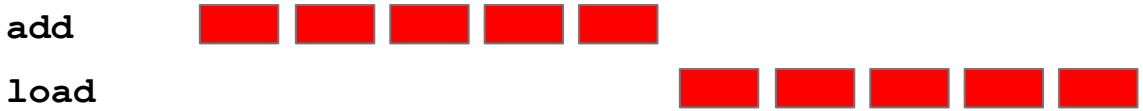


Important: Easy for software

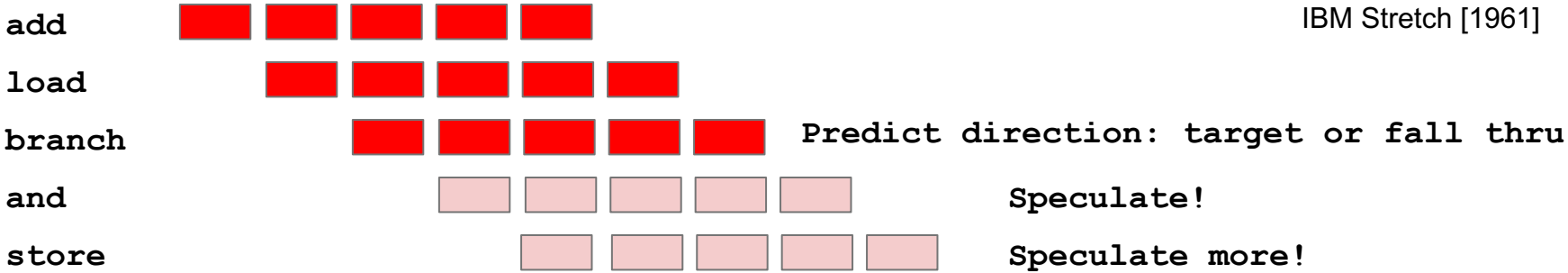
NEW: Smaller word size, e.g. machine learning inference accelerators

Instruction-level Parallelism (ILP)

Processors logically do instructions sequentially (time→)



Actually do instructions in parallel ➔ ILP

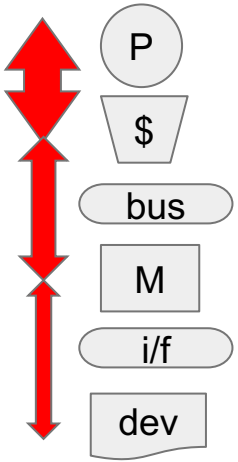


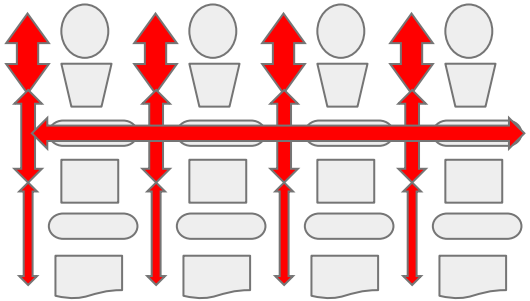
IBM Stretch [1961]

E.g., Intel Skylake has 224-entry reorder buffer w/ 14-19-stage pipeline

Important: Easy for software

X-level Parallelism in Computer Architecture





1 CPU

BLP+ILP

Bit/Instrn-Level Parallelism

Multiprocessor

+ TLP

Thread-Level Parallelism

Thread-level Parallelism (TLP)

Thread-level Parallelism

- HW: Multiple sequential processor cores
- **SW: Each runs asynchronous thread**

SW must **partition work, synchronize, & manage communication**

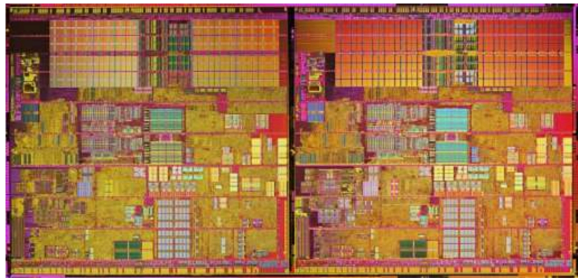
- E.g. pThreads, OpenMP, MPI

On-chip TLP called “multicore” – forced choice

Less easy for software but

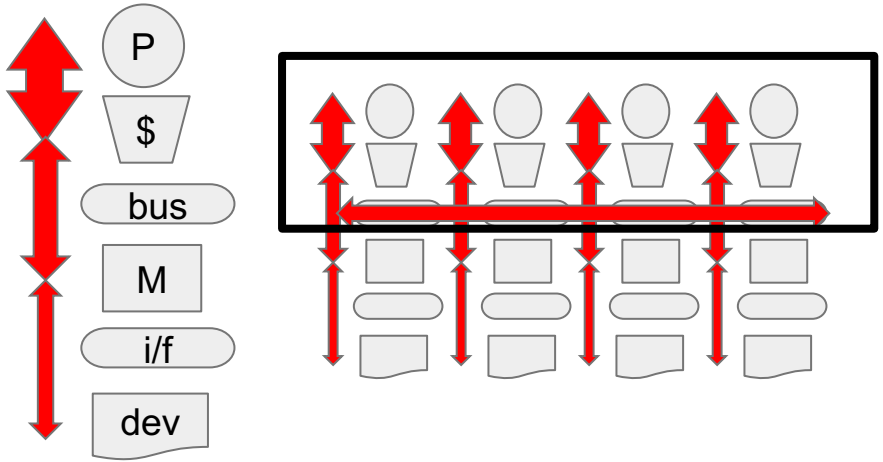
- More TLP in cloud than desktop → cloud!!
- **Bifurcation: experts program TLP; others use it**

CDC 6600, 1964,
(TLP via multithreaded processor)



Intel Pentium Pro Extreme Edition,
early 2000s

X-level Parallelism in Computer Architecture



1 CPU

BLP+ILP

Bit/Instrn-Level Parallelism

Multicore

+ TLP

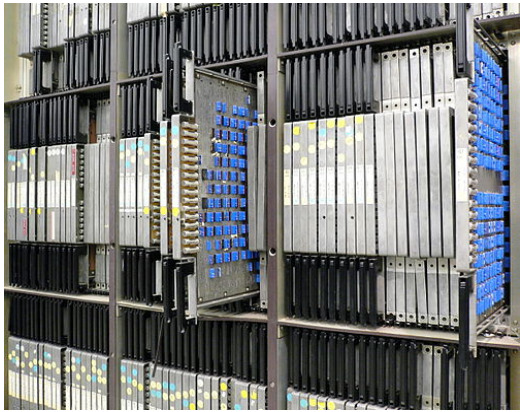
Thread-Level Parallelism

Data-level Parallelism (DLP)

Need same operation on many data items

Do with parallelism ➔ DLP

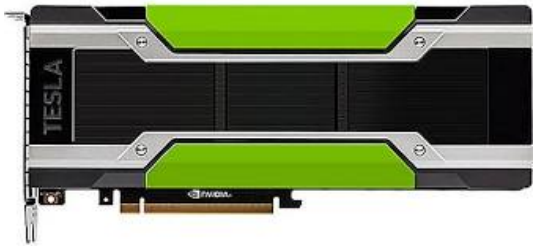
- Array of single instruction multiple data (SIMD)
- Deep pipelines like Cray vector machines
- Intel-like Streaming SIMD Extensions (SSE)



Illinois ILLIAC IV, 1966

Broad DLP success awaited General-Purpose GPUs

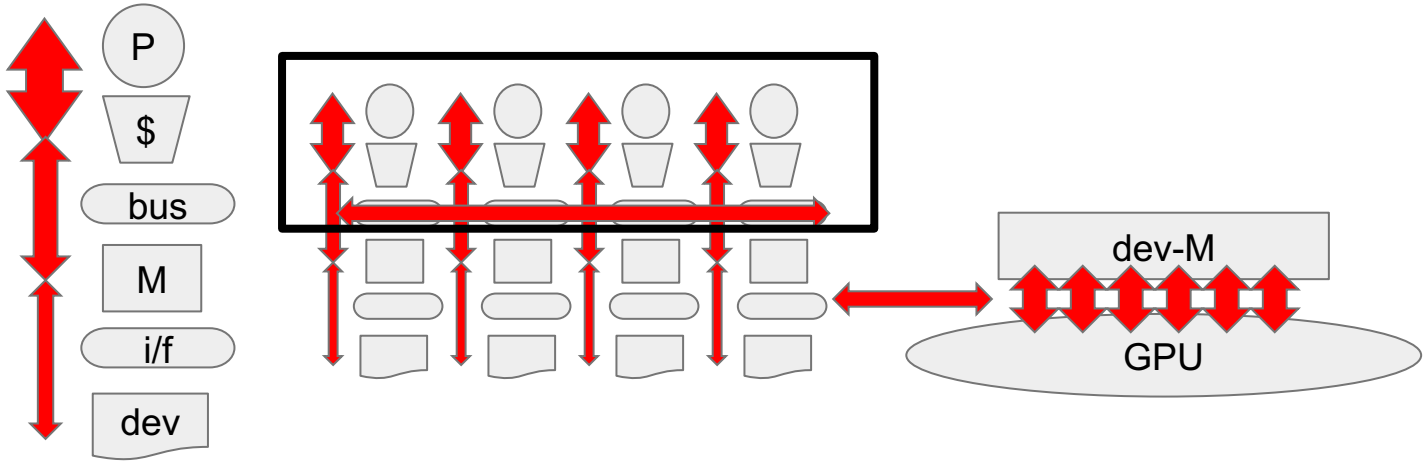
1. **Single Instruction Multiple Thread (SIMT)**
2. **SW (CUDA) & libraries (math & ML)**
3. **Experimentation as $1-10K not $1-10M**



NVIDIA Tesla

Bifurcation again: experts program SIMT (TLP+DLP); others use it

X-level Parallelism in Computer Architecture



1 CPU

BLP+ILP

Bit/Instrn-Level Parallelism

Multicore

+ TLP

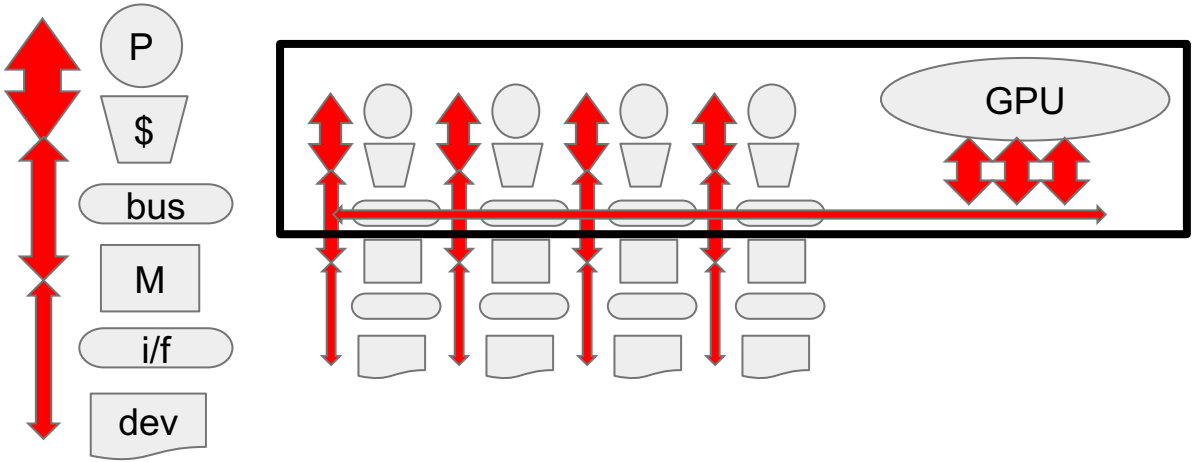
Thread-Level Parallelism

\+ **Discrete GPU**

+ DLP

Data-Level Parallelism

X-level Parallelism in Computer Architecture

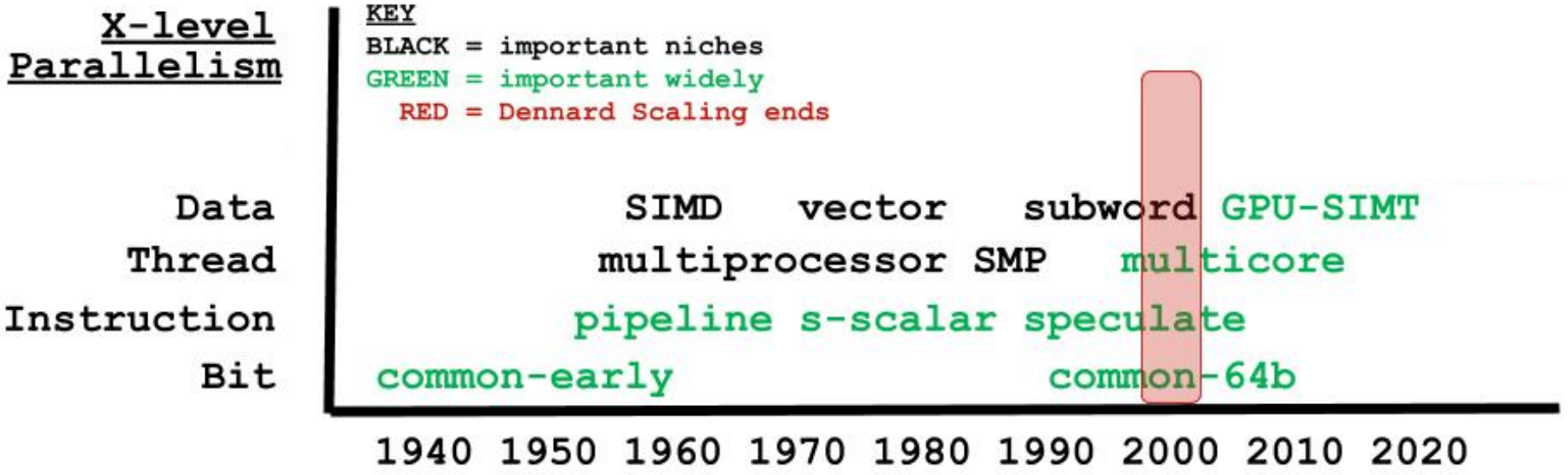


1 CPU | **Multicore** + **Integrated GPU**

BLP+ILP | + TLP | + DLP

Bit/Instrn-Level Parallelism | Thread-Level Parallelism | Data-Level Parallelism

X-level Parallelism in Computer Architecture



Outline

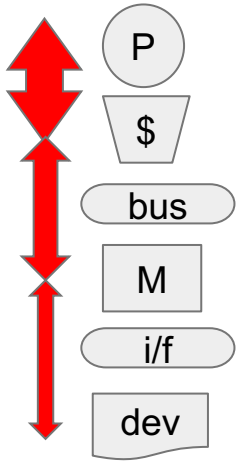
I. Computer History & X-level Parallelism

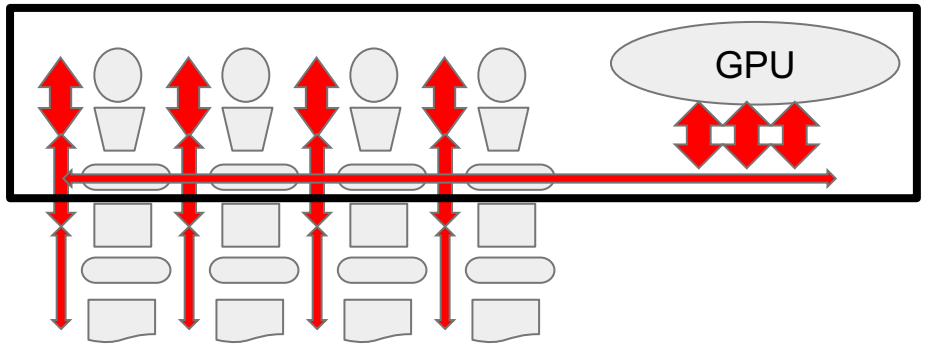
II. Mobile SoCs as ALP Harbinger

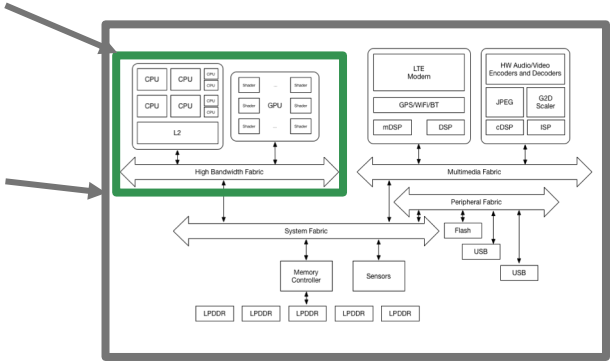
III. Gables ALP SoC Model

IV. Call to Action for Accelerator-level Parallelism

X-level Parallelism in Computer Architecture







1 CPU

BLP+ILP

Bit/Instrn-Level Parallelism

Multicore + **Integrated GPU**

+ TLP

Thread-Level Parallelism

+ DLP

Data-Level Parallelism

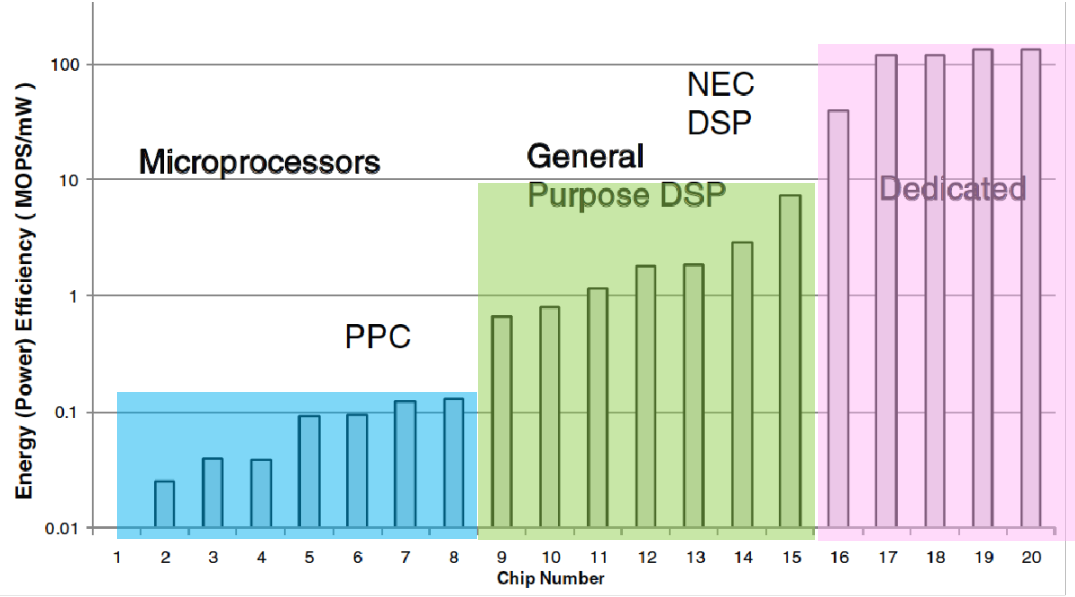
System on a Chip (SoC)

+ ALP

Accelerator-Level Parallelism

Potential for Specialized Accelerators (IPs)

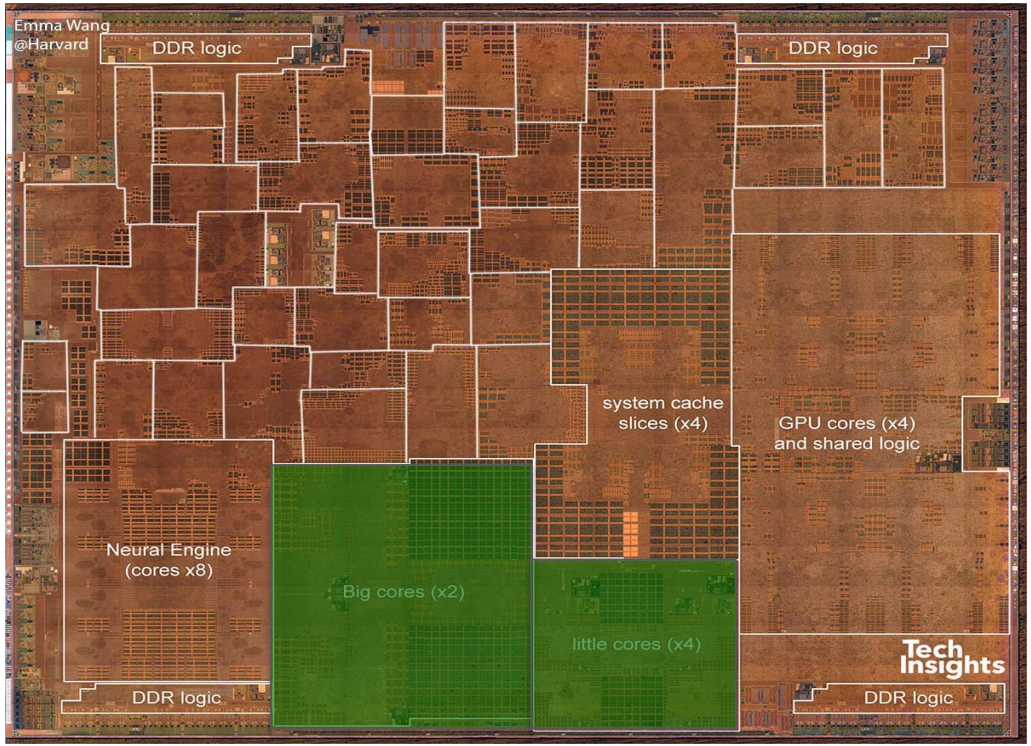
Accelerator is a hardware component that executes a targeted computation class faster & usually with (much) less energy.



16 Encryption
17 Hearing Aid
18 FIR for disk read
19 MPEG Encoder
20 802.11 Baseband

[Brodersen & Meng, 2002]

CPU, GPU, xPU (i.e., Accelerators or IPs)

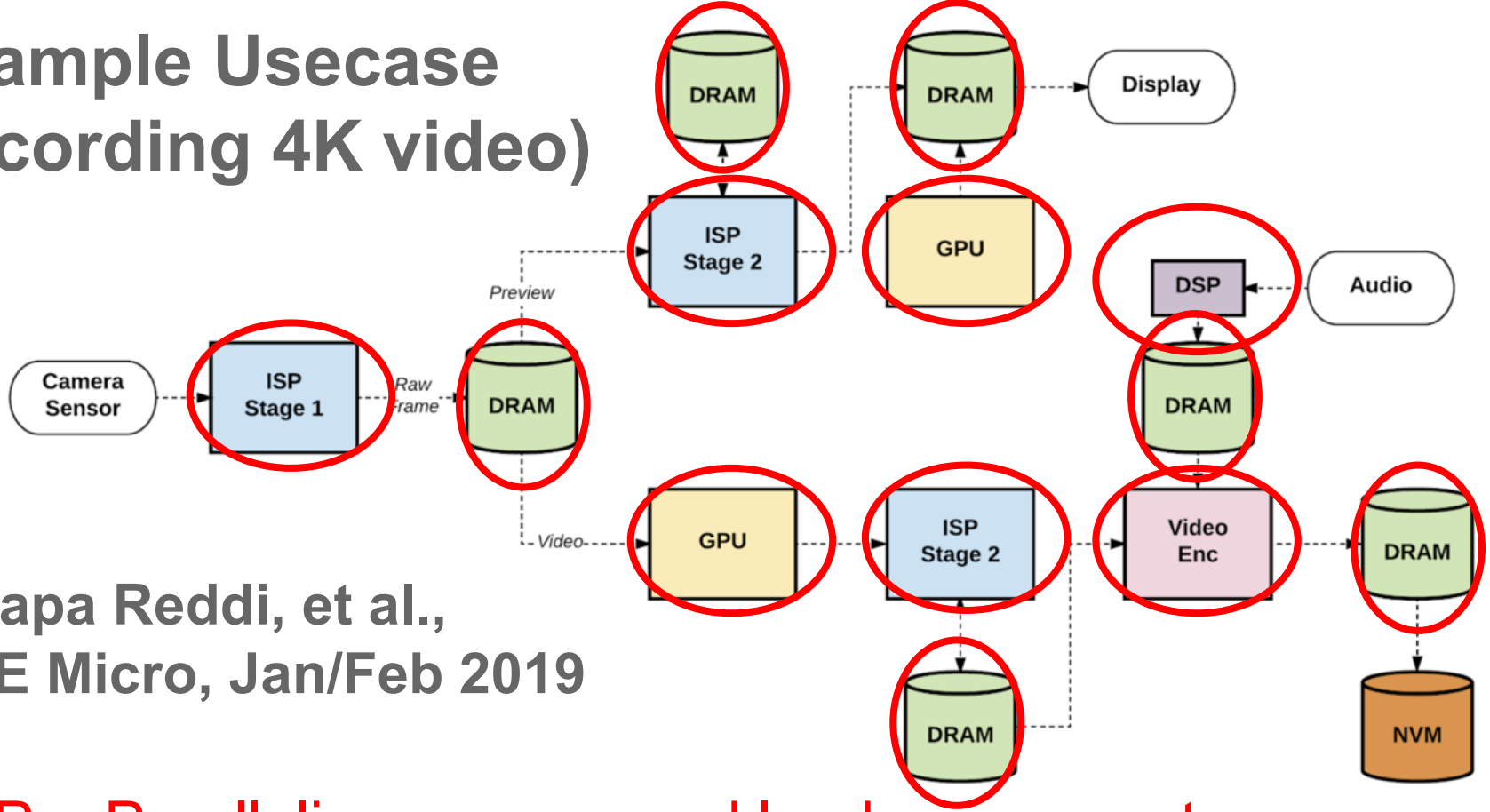


2019 Apple A12 w/ 42 accelerators

42 Really?

The Hitchhiker's Guide to the Galaxy?

Example Usecase (recording 4K video)



Janapa Reddi, et al.,
IEEE Micro, Jan/Feb 2019

ALP = Parallelism among workload components concurrently executing on multiple accelerators (IPs)

Mobile SoCs Run Usecases

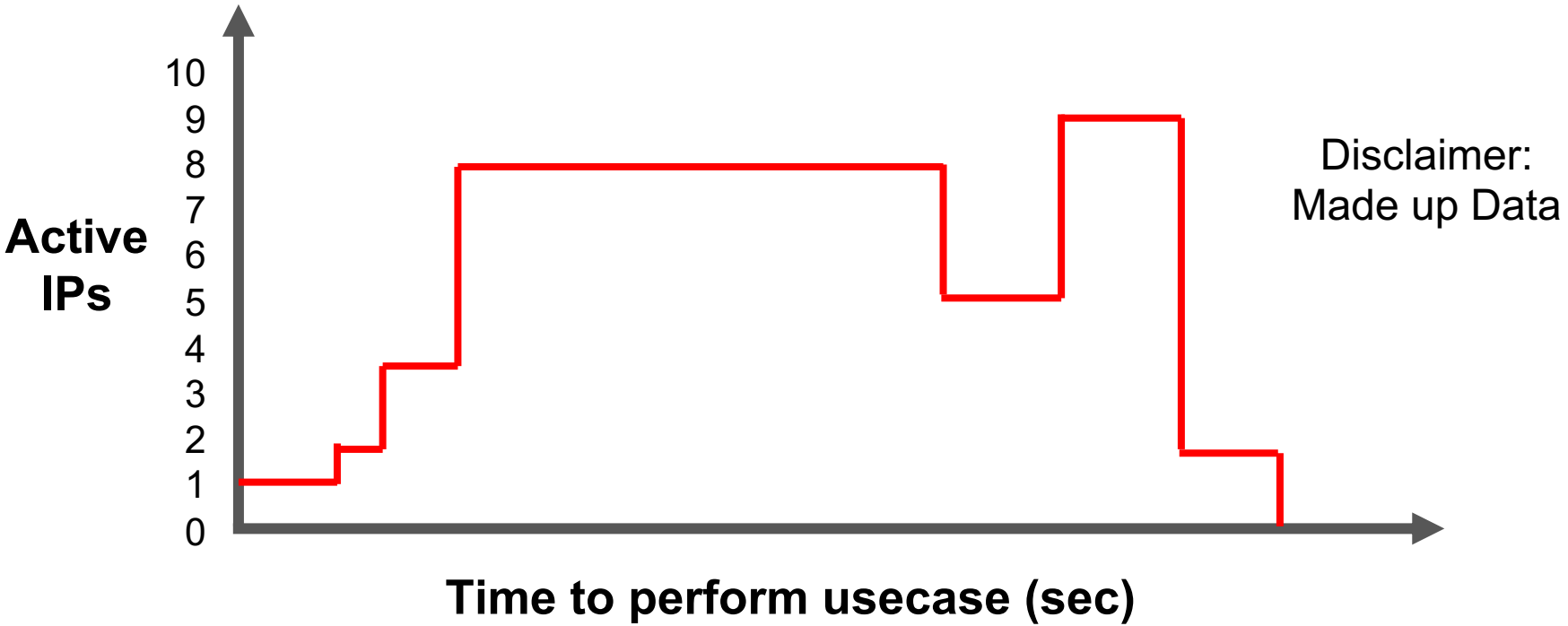
Accelerators (IPs) ➔ Usecases (rows)	CPUs (AP)	Display	Media Scaler	GPU	Image Signal Proc.	JPEG	Pixel Visual Core	Video Decoder	Video Encoder	**Dozens More**
Photo Enhancing	X	X		X	X	X	X			
Video Capture	X	X		X	X				X	
Video Capture HDR	X	X		X	X				X	
Video Playback	X	X	X	X				X		
Image Recognition	X	X	X	X						

Must run each usecase sufficiently fast -- no need faster

A usecase uses IPs concurrently: **more ALP** than serial

For each usecase, how much acceleration for each IP?

ALP(t) = #IPs concurrently active at time t



Outline

I. Computer History & X-level Parallelism

II. Mobile SoCs as ALP Harbinger

III. Gables ALP SoC Model [HPCA'19]

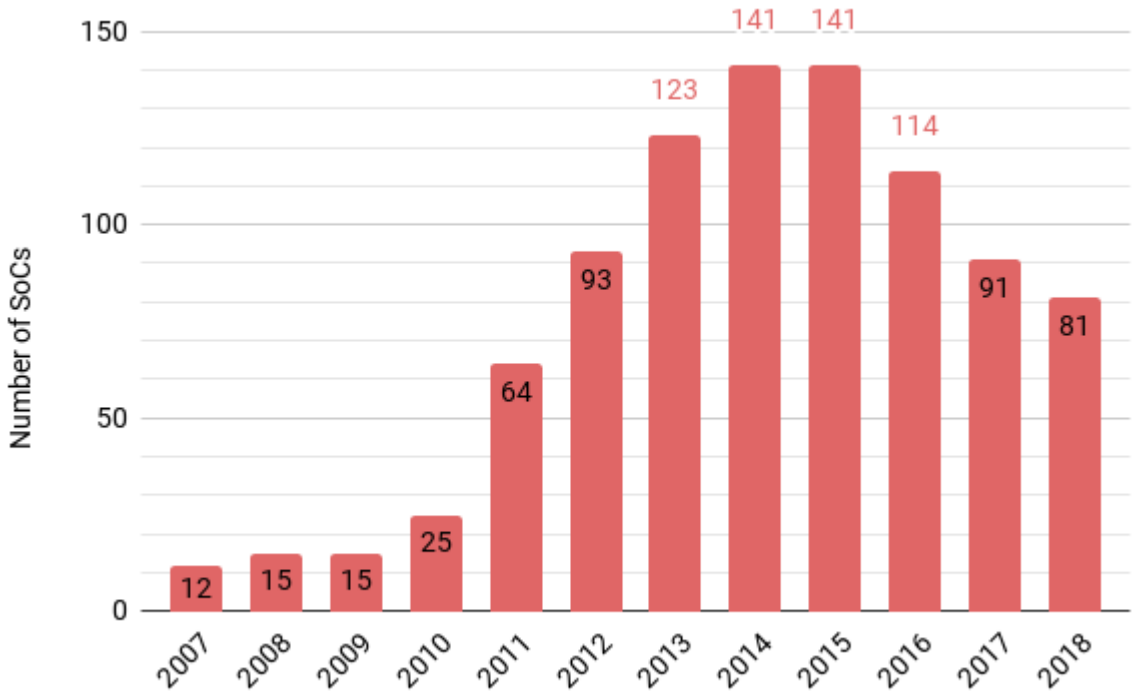
IV. Call to Action for Accelerator-level Parallelism

Mobile SoCs Hard To Program For and Select

Envision usecases (years ahead)
Port to many SoCs??

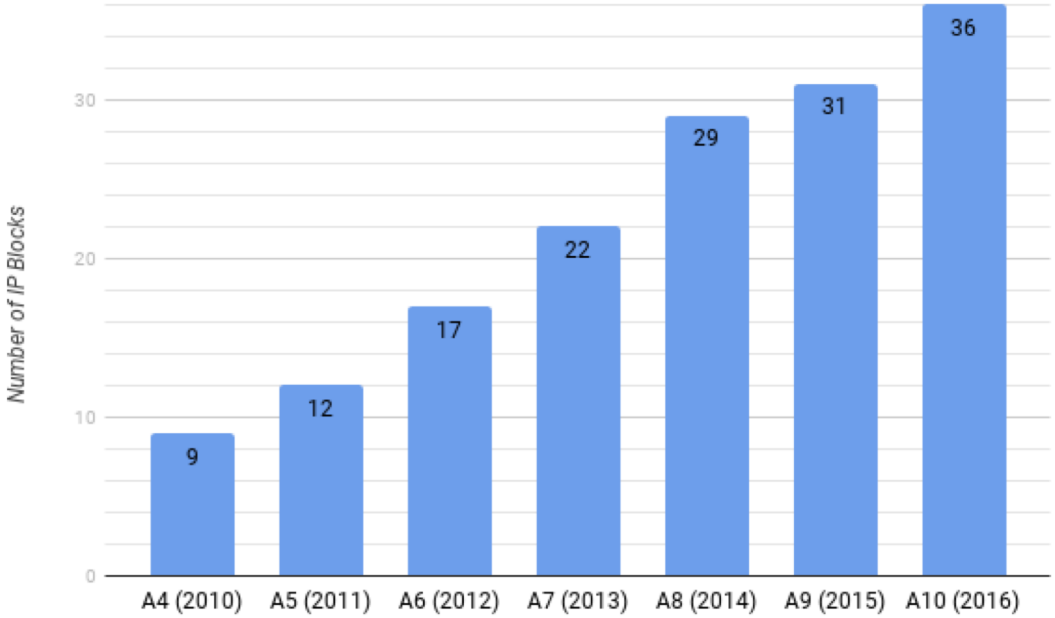
Diversity hinders use [Facebook, HPCA'19]

How to reason about SoC performance?



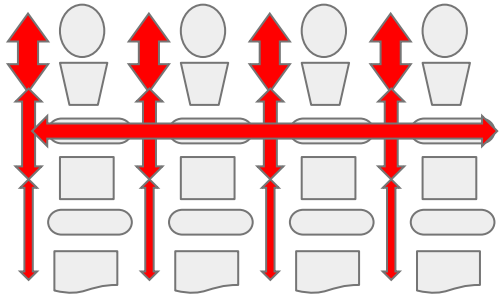
Mobile SoCs Hard To Design

Envision usecases
(2-3 years ahead)
Select IPs
Size IPs
Design Uncore

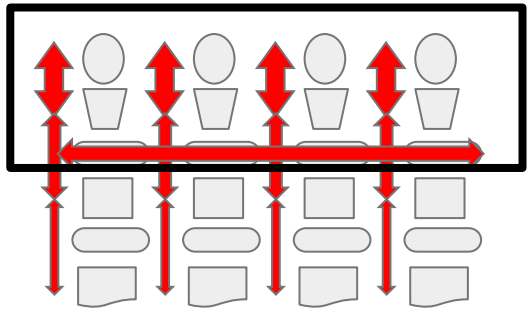


Which accelerators? How big? How to even start?

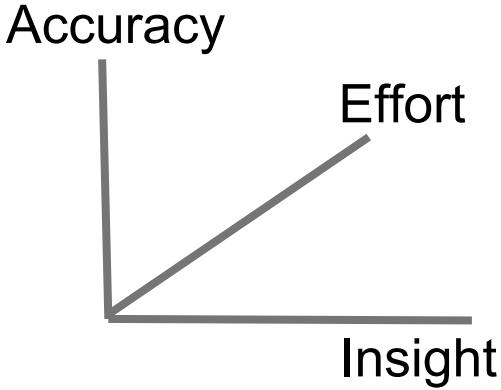
Computer Architecture & Performance Models



Multiprocessor &
Amdahl's Law



Multicore &
Roofline



Models vs Simulation

- More insight
- Less effort
- But less accuracy

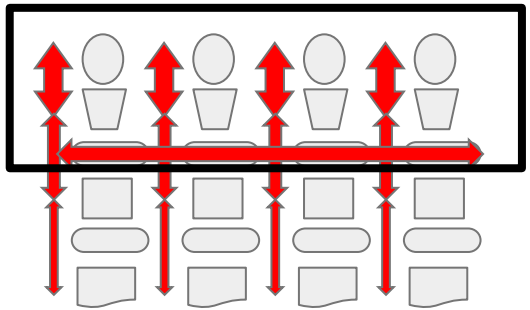
Models give first answer, not final answer

Gables extends Roofline ➔ first answer for SoC ALP

Roofline for Multicore Chips, 2009

Multicore HW

- P_{peak} = peak perf of all cores
- B_{peak} = peak off-chip bandwidth

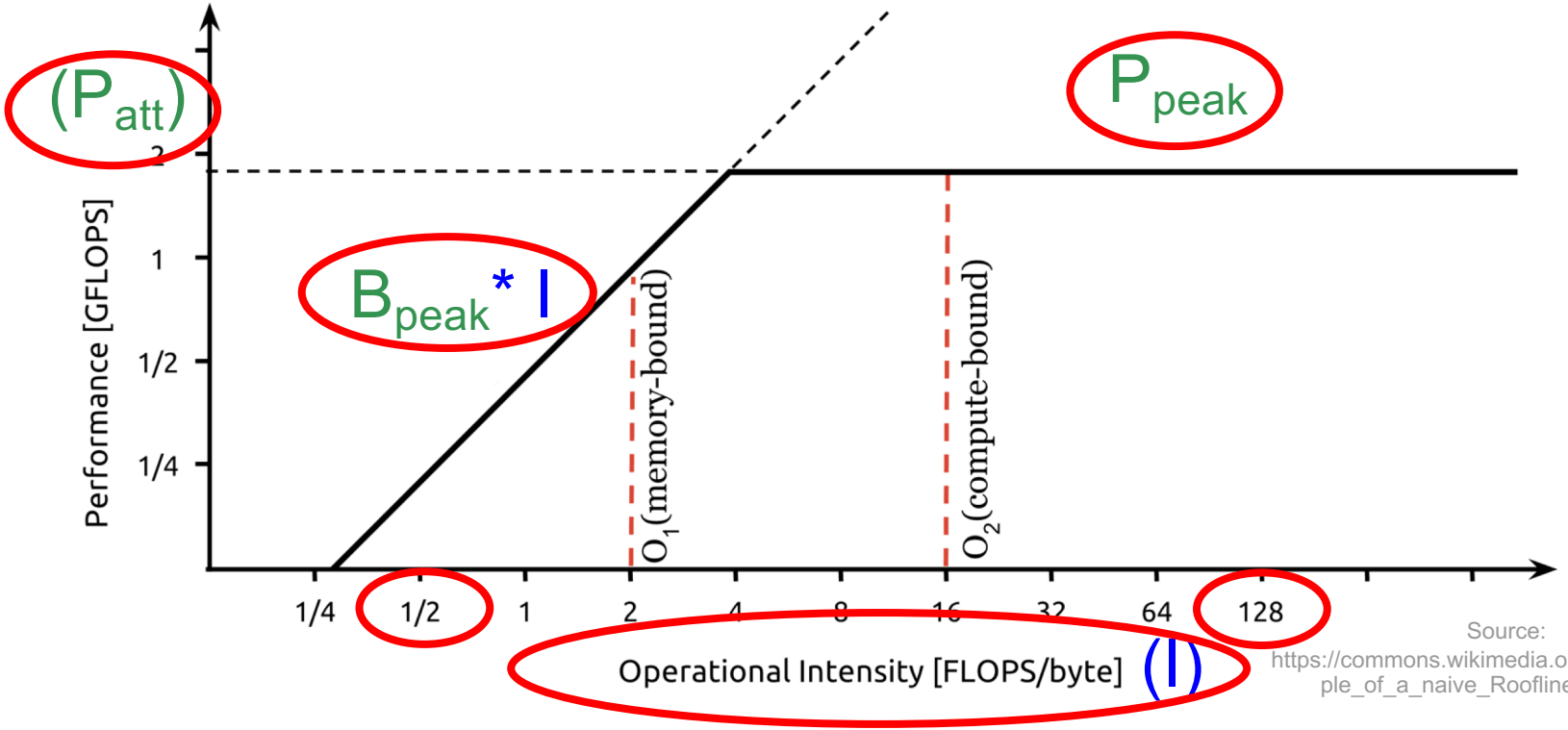


Multicore SW

- I = operational intensity = #operations/#off-chip-bytes
- E.g., 2 ops / 16 bytes → I = 1/8

Output P_{att} = upper bound on performance attainable

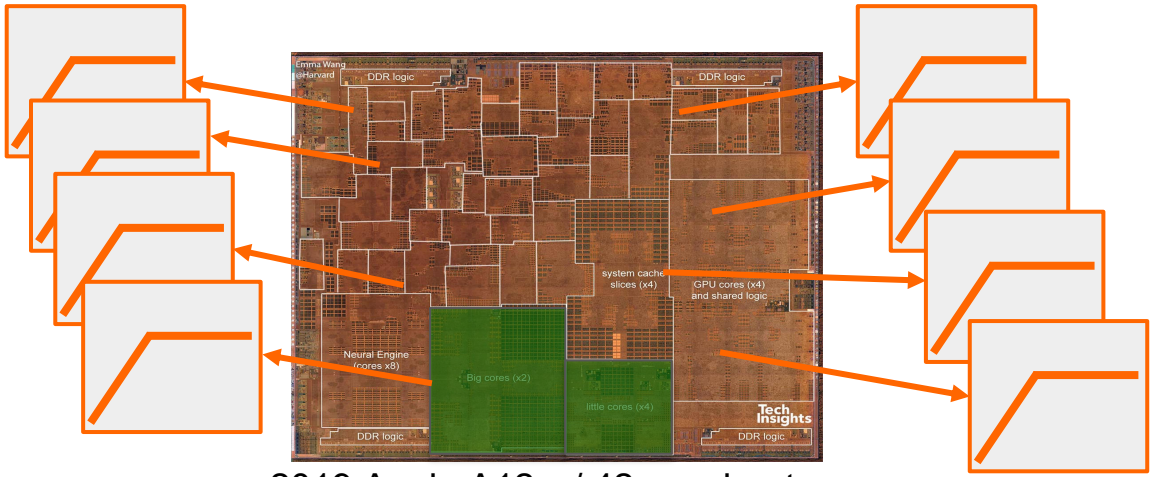
Roofline for Multicore Chips, 2009



Source: https://commons.wikimedia.org/wiki/File:Example_of_a_naive_Roofline_model.svg

Compute v. Communication: Op. Intensity (I) = #operations / #off-chip bytes

ALP System on Chip (SoC) Model: NEW Gables

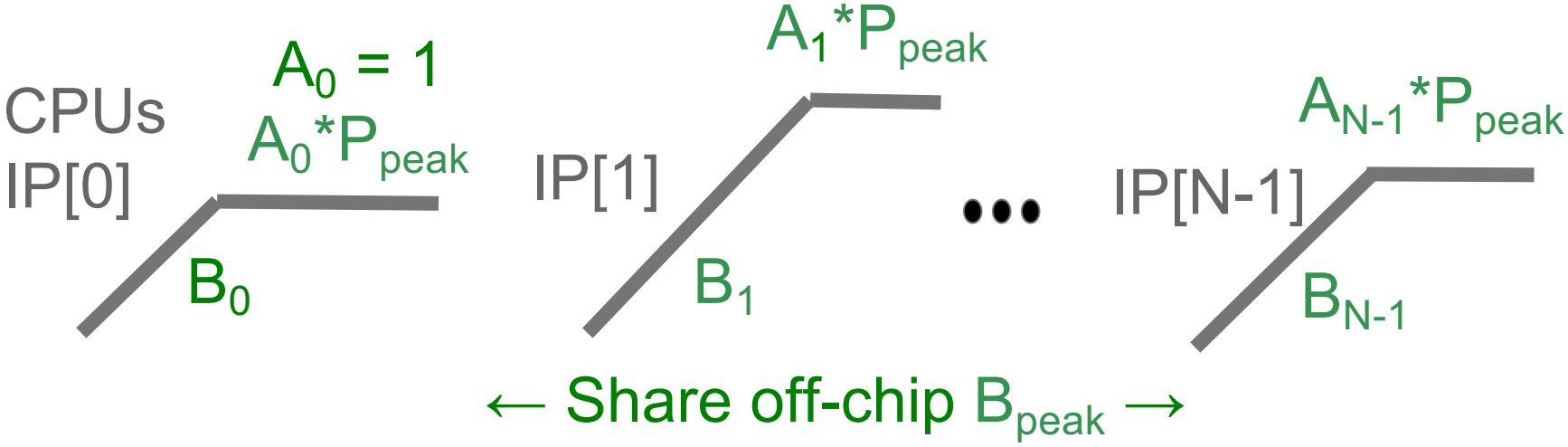


2019 Apple A12 w/ 42 accelerators

Gables uses Roofline per IP to provide first answer!

- SW: performance model of a “gabled roof?”
- HW: select & size accelerators

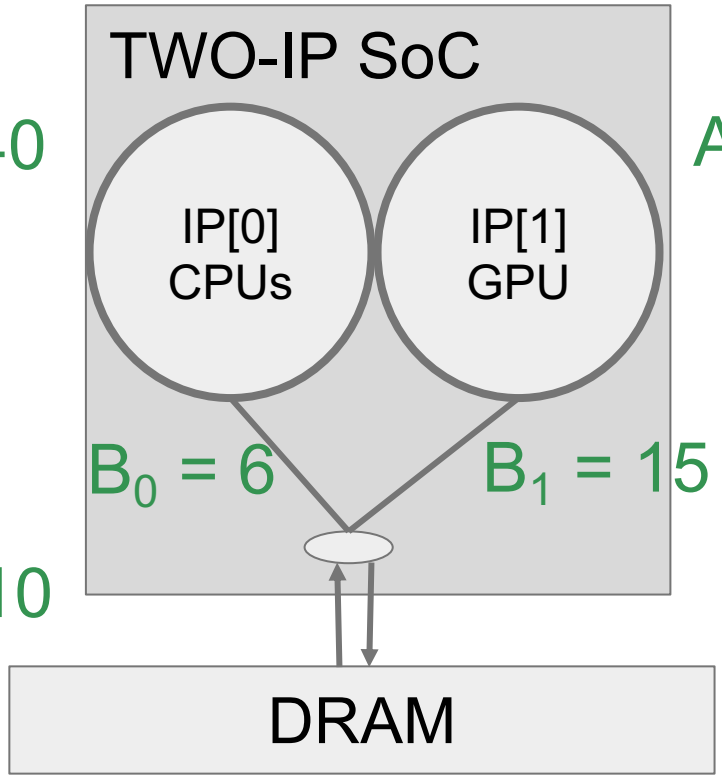
Gables for N IP SoC



Usecase at each IP[i]

- Operational intensity I_i operations/byte
- Non-negative work f_i (f_i's sum to 1) w/ IPs in parallel

Example Balanced Design Start w/ Gables



$P_{peak} = 40$

$A_1 * P_{peak} = 5*40 = 200$

$B_0 = 6$

$B_1 = 15$

$B_{peak} = 10$

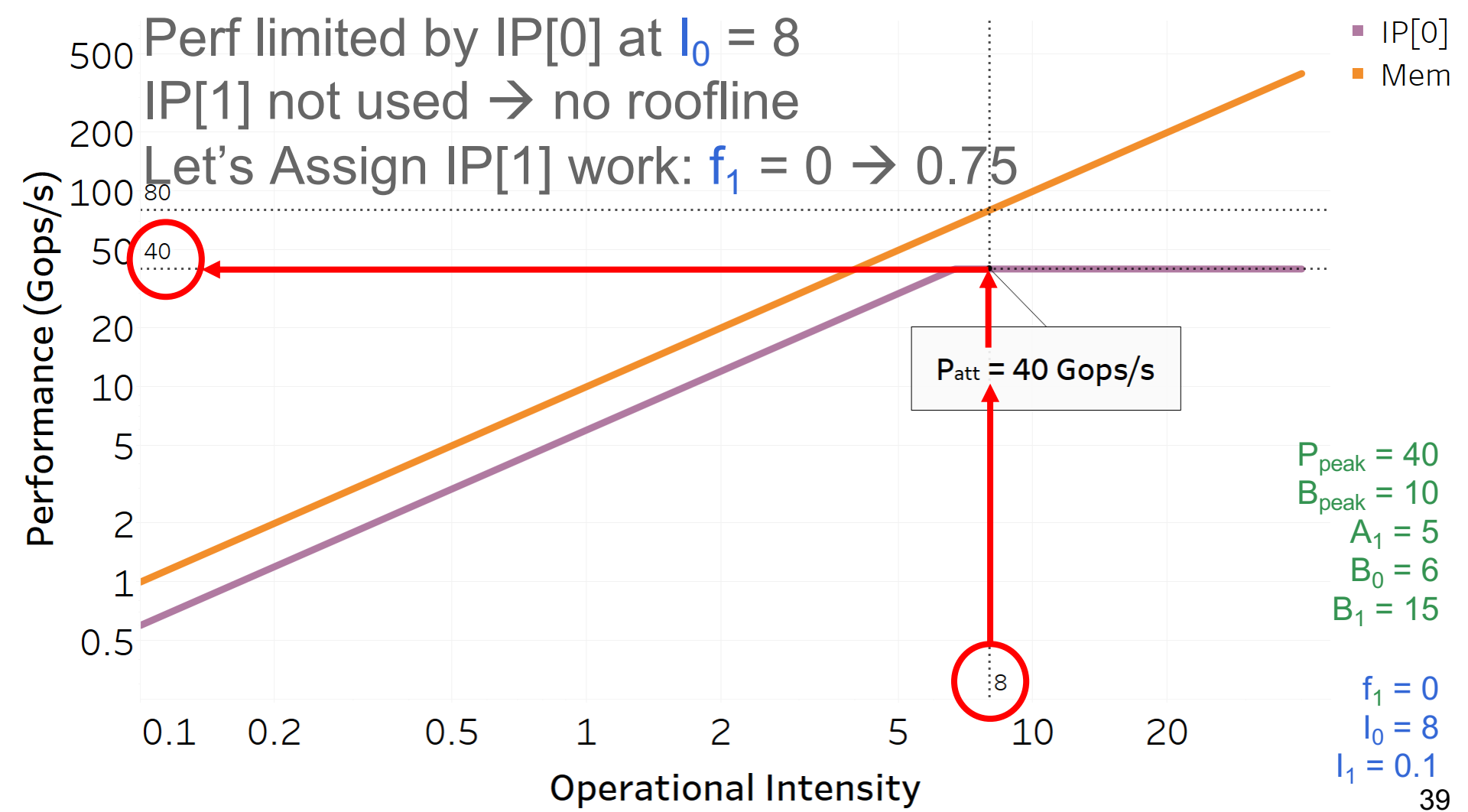
Workload (Usecase):

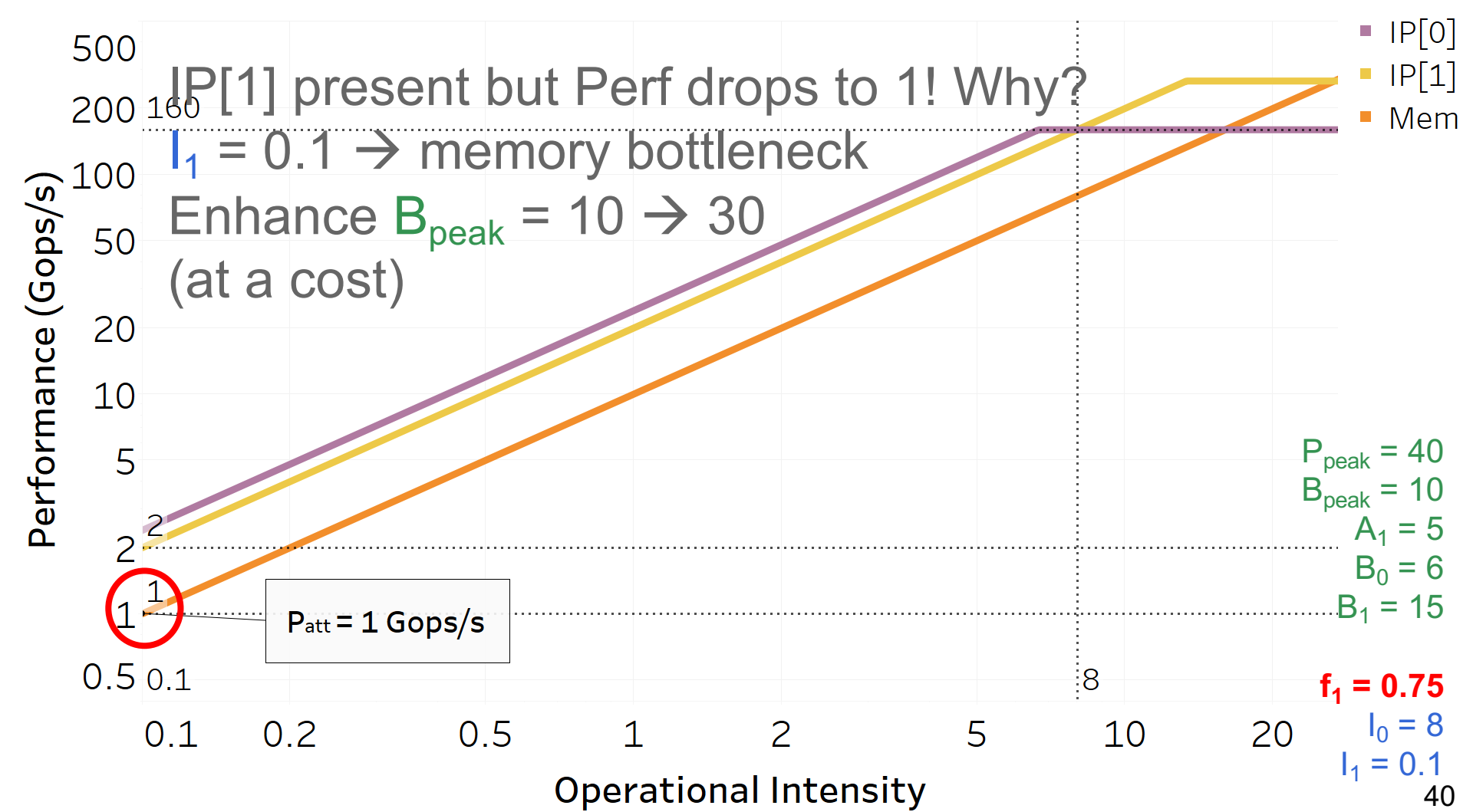
$f_0 = 1$ & $f_1 = 0$

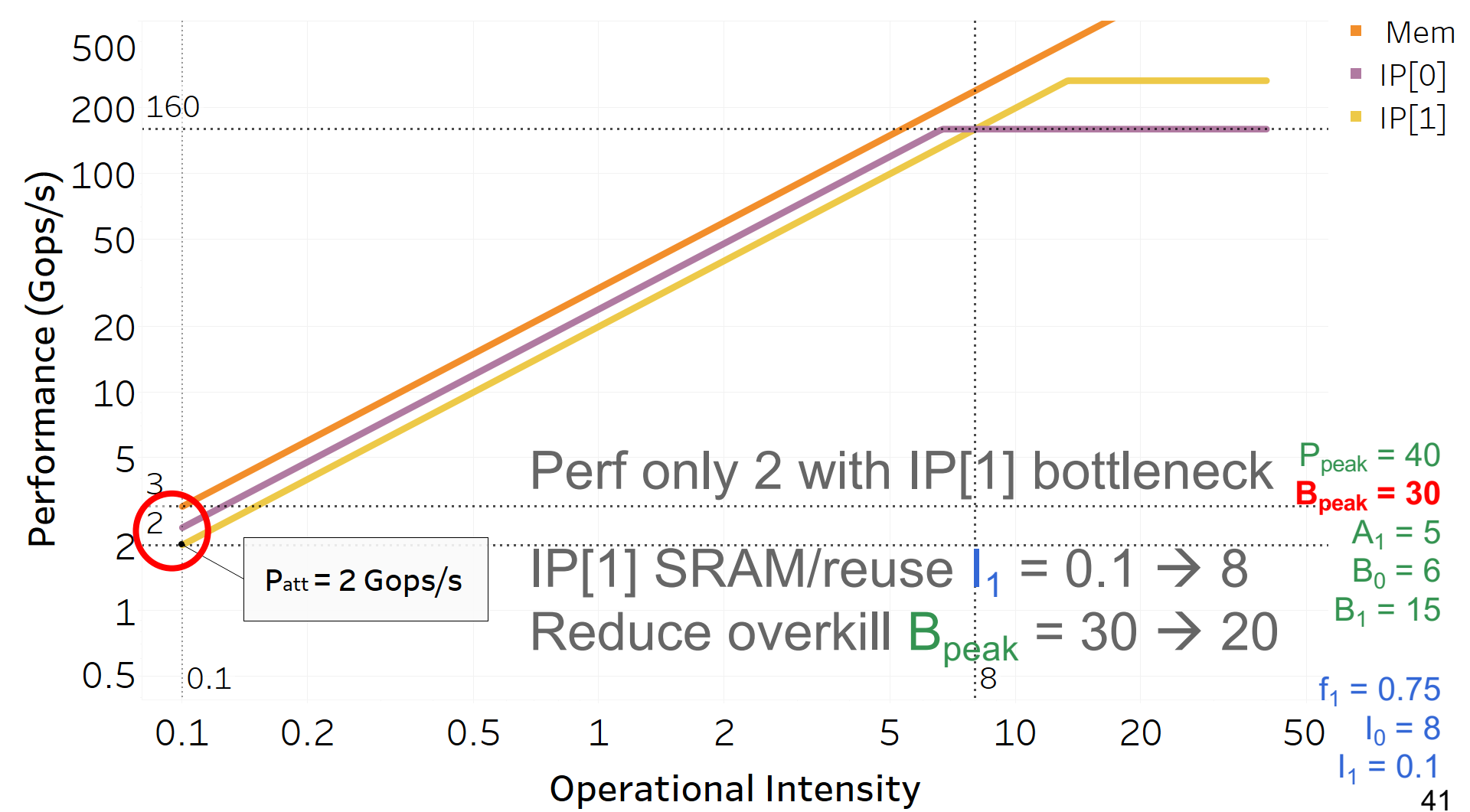
$I_0 = 8$ = good caching

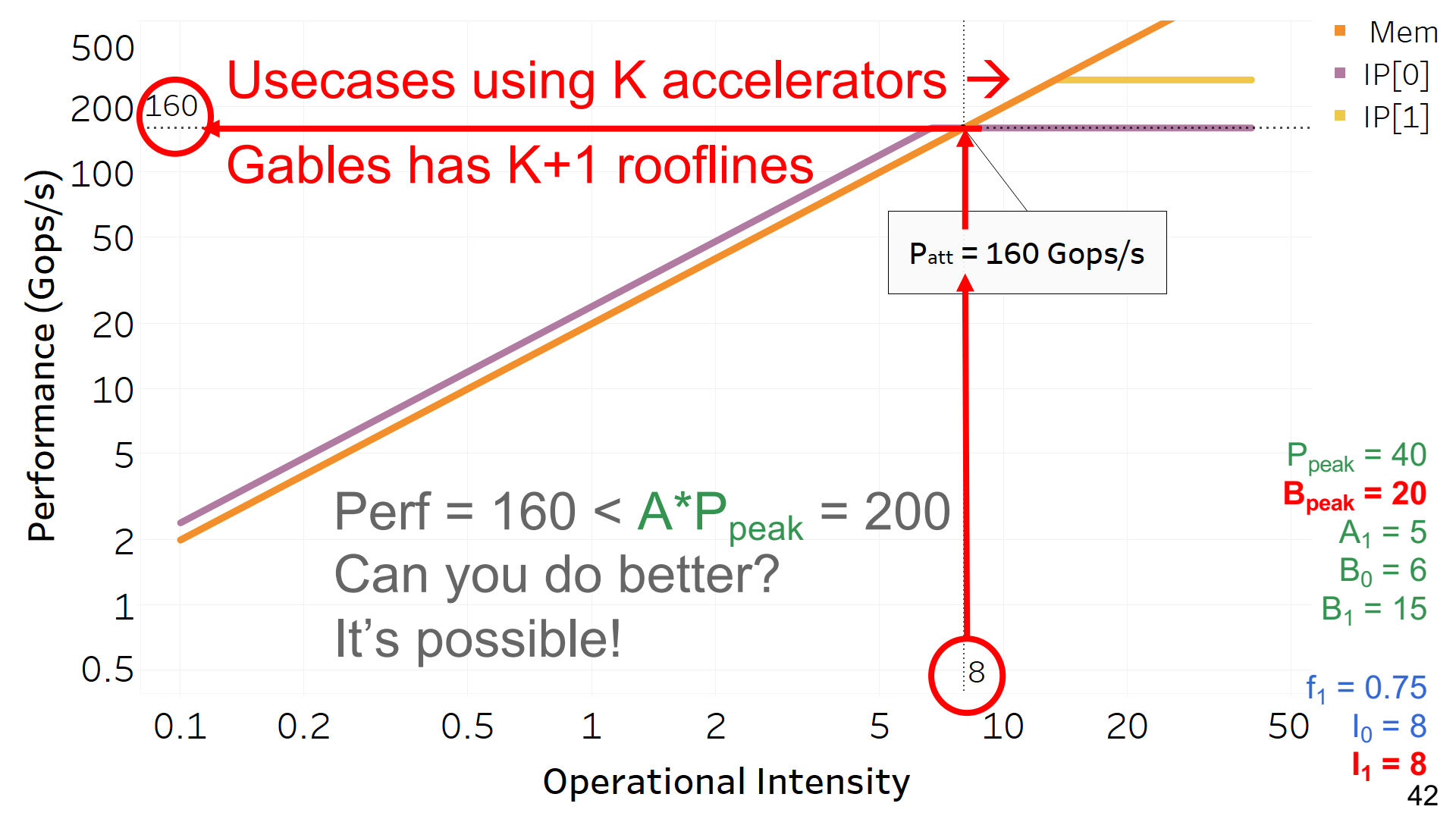
$I_1 = 0.1$ = latency tolerant

Performance?


Perf limited by IP[0] at I_0 = 8
IP[1] not used → no roofline
Let's Assign IP[1] work: f_1 = 0 → 0.75
IP[0]
Mem
Performance (Gops/s)
500
200
100
80
50
40
20
10
5
2
1
0.5
0.1
0.2
0.5
1
2
5
10
20
8
Operational Intensity
Patt = 40 Gops/s
Ppeak = 40
Bpeak = 10
A1 = 5
B0 = 6
B1 = 15
f1 = 0
I0 = 8
I1 = 0.1


IP[1] present but Perf drops to 1! Why?
I_1 = 0.1 → memory bottleneck
Enhance B_{peak} = 10 → 30
(at a cost)
IP[0]
IP[1]
Mem
Performance (Gops/s)
500
200
160
100
50
20
10
5
2
2
1
1
0.5
0.1
P_{att} = 1 Gops/s
P_{peak} = 40
B_{peak} = 10
A_1 = 5
B_0 = 6
B_1 = 15
f_1 = 0.75
I_0 = 8
I_1 = 0.1
8
0.1
0.2
0.5
1
2
5
10
20
Operational Intensity

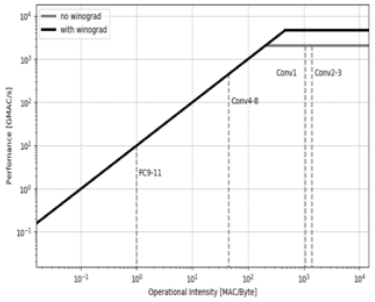

Mem
IP[0]
IP[1]
Performance (Gops/s)
500
200
160
100
50
20
10
5
3
2
2
1
0.5
0.1
0.1
0.2
0.5
1
2
5
8
10
20
50
Operational Intensity
Perf only 2 with IP[1] bottleneck
IP[1] SRAM/reuse I_1 = 0.1 → 8
Reduce overkill B_{peak} = 30 → 20
P_{att} = 2 Gops/s
P_{peak} = 40
B_{peak} = 30
A_1 = 5
B_0 = 6
B_1 = 15
f_1 = 0.75
I_0 = 8
I_1 = 0.1


Usecases using K accelerators →
Gables has K+1 rooflines
160
Patt = 160 Gops/s
Perf = 160 < A*Ppeak = 200
Can you do better?
It's possible!
8
Mem
IP[0]
IP[1]
Ppeak = 40
Bpeak = 20
A1 = 5
B0 = 6
B1 = 15
f1 = 0.75
I0 = 8
I1 = 8
Performance (Gops/s)
500
200
100
50
20
10
5
2
1
0.5
0.1
0.2
0.5
1
2
5
10
20
50
Operational Intensity

Approach: Combine Analytical and Simulation Models

Analytical Models

+ Good first order exploration
+ Results within minutes
~ No dynamic effects



Refine

validate
back-annotate

Architecture Simulation

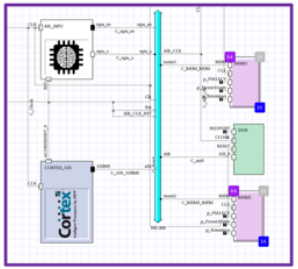
+ Exploration and optimization
+ Simulations in minutes/hours
~ Requires characterization

Refine

validate
back-annotate

Cycle-level Simulation

+ Full cycle-accuracy
- Long turn-around time



Gables: A Roofline Model for Mobile SoCs,
Mark D. Hill and Vijay Janapa Reddi, HPCA, 2019

Into Synopsys design flow << 6 months of publication!

SYNOPSYS®
© 2019 Synopsys, Inc.

Case Study: IT Company + Synopsys

Two cases where: Gables >> Actual

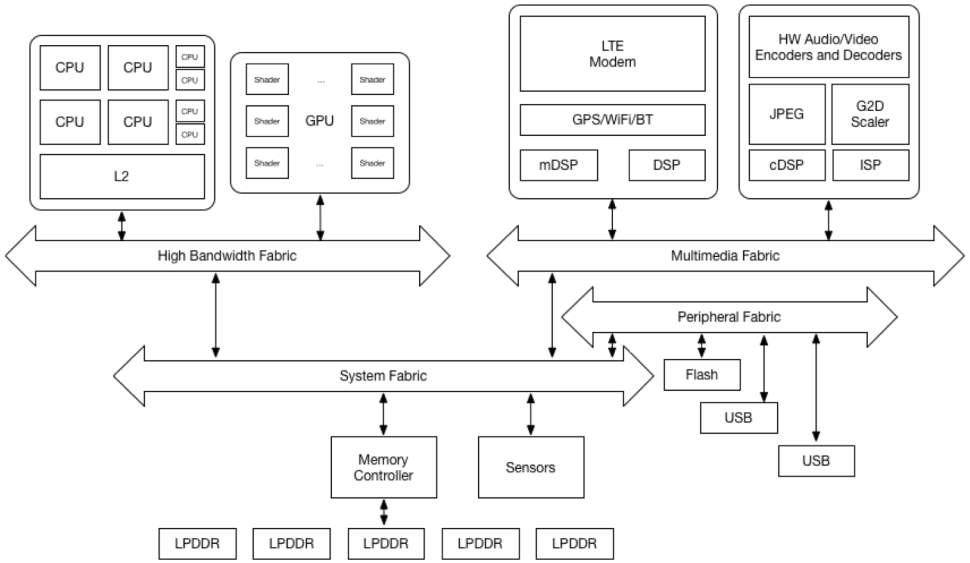
1. Communication between two IP blocks

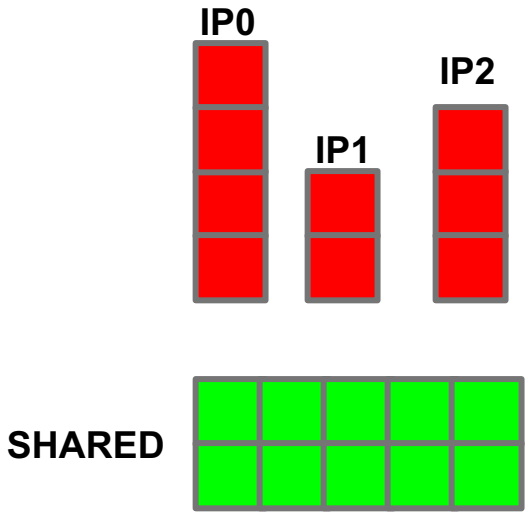
- **Root:** Too few buffers to cover communication latency
- **Little's Law:** # outstanding msgs = avg latency * avg BW
- https://www.sigarch.org/three-other-models-of-computer-system-performance-part-1/
- **Solution:** Add buffers; actual performance → Gables

2. More complex interaction among IP blocks

- **Root:** Usecase work (task graph) not completely parallel
- **Solution:** No change, but useful double-check

Case Study: Allocating SRAM



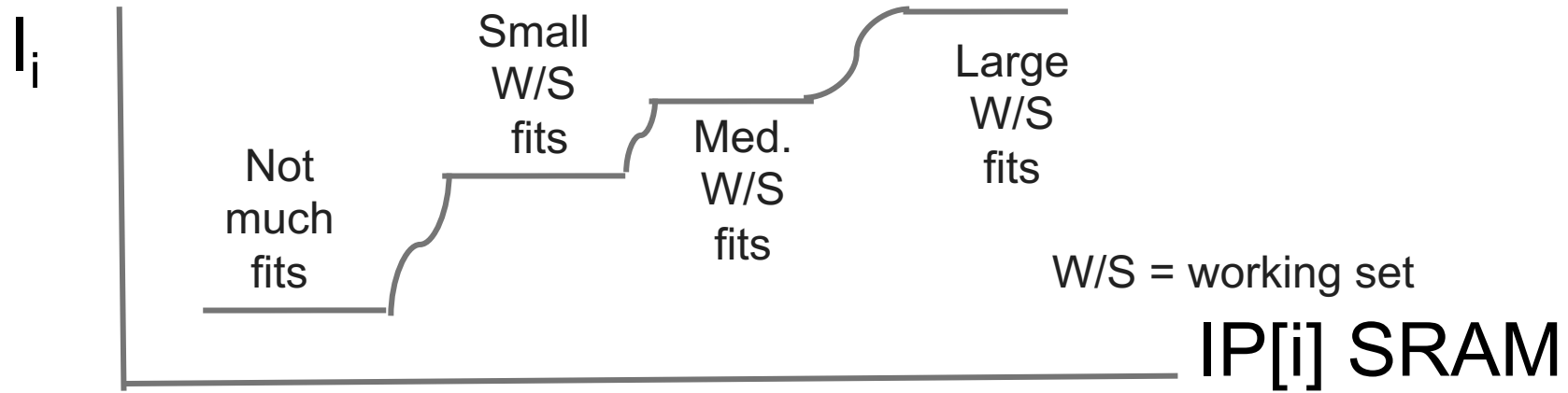


Where SRAM?

- Private w/i each IP
- Shared resource

Does more IP[i] SRAM help Op. Intensity (I_i)?

Compute v. Communication: Op. Intensity (I) = #operations / #off-chip bytes



Non-linear function that increases when new footprint/working-set fits

Should consider these plots when sizing IP[i] SRAM

Later evaluation can use simulation performance on y-axis

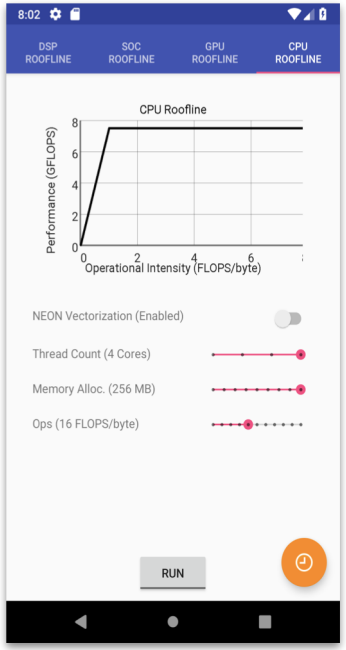
Gables Home Page

[HPCA'19]

Model Extensions

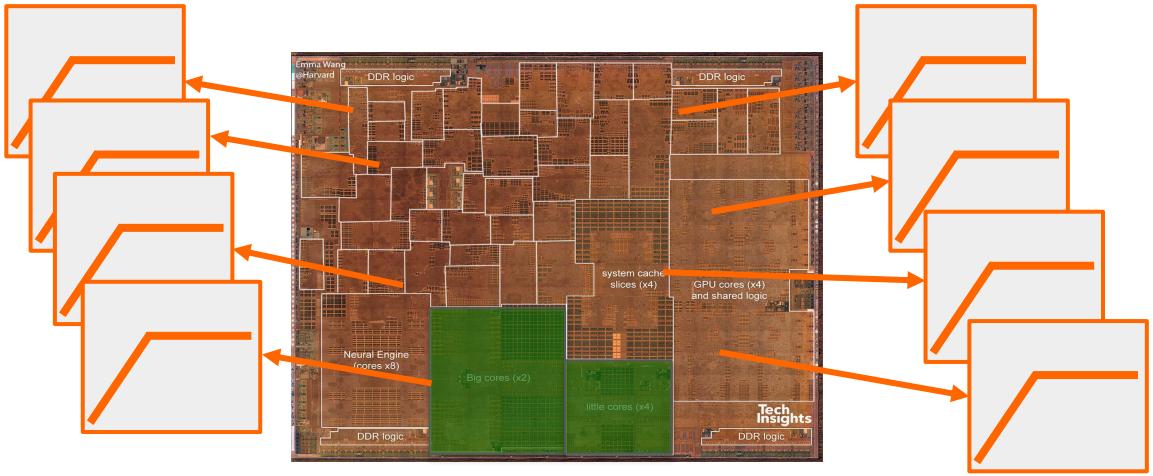
Interactive tool

Gables Android Source at GitHub



http://research.cs.wisc.edu/multifacet/gables/

Mobile System on Chip (SoC) & Gables



SW: Map usecase to IP's w/ many BWs & acceleration

HW: IP[i] under/over-provisioned for BW or acceleration?

Gables—like Amdahl's Law—gives intuition & a first answer

But still missing is SoC "architecture" & programming model

Outline

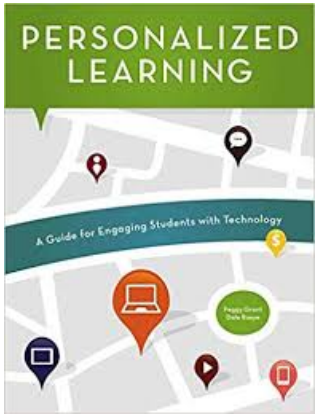
I. Computer History & X-level Parallelism

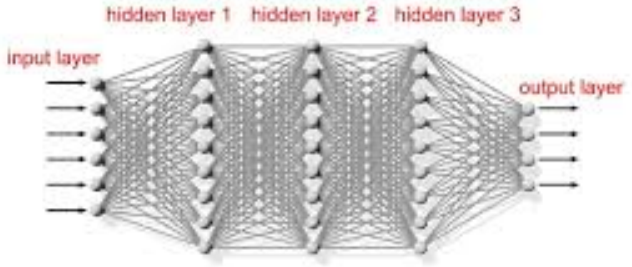
II. Mobile SoCs as ALP Harbinger

III. Gables ALP SoC Model

IV. Call to Action for Accelerator-level Parallelism

Future Apps Demand Much More Computing





deep neural network

Accelerator-level Parallelism Call to Action

Future apps demand much more computing

Standard tech scaling & architecture NOT sufficient

Mobile SoCs show a promising approach:

ALP = Parallelism among workload components concurrently executing on multiple accelerators (IPs)

Call to action to develop "science" for ubiquitous ALP

- **An SoC architecture that exposes & hides?**
- ***A whole SoC programming model/runtime?***

ALP/SoC Software Descent to Hellfire!

Local SW stack **abstracts** each accelerator.

But no good, general SW abstraction for SoC ALP!

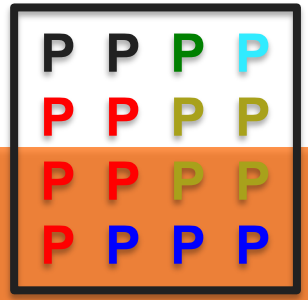
No visible parallelism

Uniprocessor

Any thread-level parallelism, e.g., homogeneous

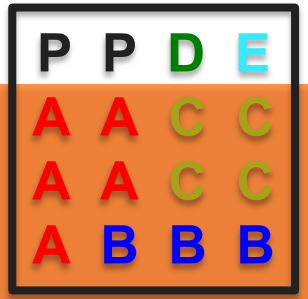
Homogeneous Multicore

Thought bridge: Must divide work heterogeneously



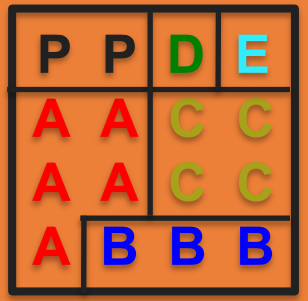
Heterogeneous Multicore

Accelerate each differently with unique HLLs (DSLs) & SDKs



Heterogeneous Accelerators

All of above & hide in many kernel drivers ☹



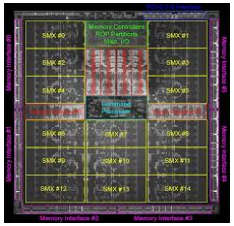
Today: Device Accelerators

Hellfire!

Key: P == processor core; A-E == accelerators

SW+HW Lessons from GP-GPUs?

Programming for data-level parallelism: **four decades**
SIMD→Vectors→SSE→SIMT!



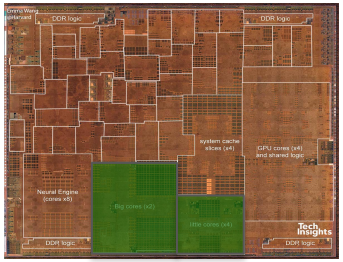
Nvidia GK110
BLP+TLP+DLP

Feature	**Then**
1. Programming	Graphics OpenGL
2. Concurrency	Either CPU or GPU only; Intra-GPU mechanisms
3. Communication	Copy data between host & device memories
4. Design	Driven by graphics only; GP: $0B market

SW+HW Directions for ALP?

Need programmability for broad success!!!!

In less than four decades?



Apple A12: BLP+ILP+TLP+DLP+ALP

Feature	**Now**
1. Programming	Local: Per-IP DSL & SDK Global: Ad hoc
2. Concurrency	Ad hoc
3. Communication	SW: Up/down OS stack HW: Via off-chip memory
4. Design, e.g., select, combine, & size IPs	Ad hoc

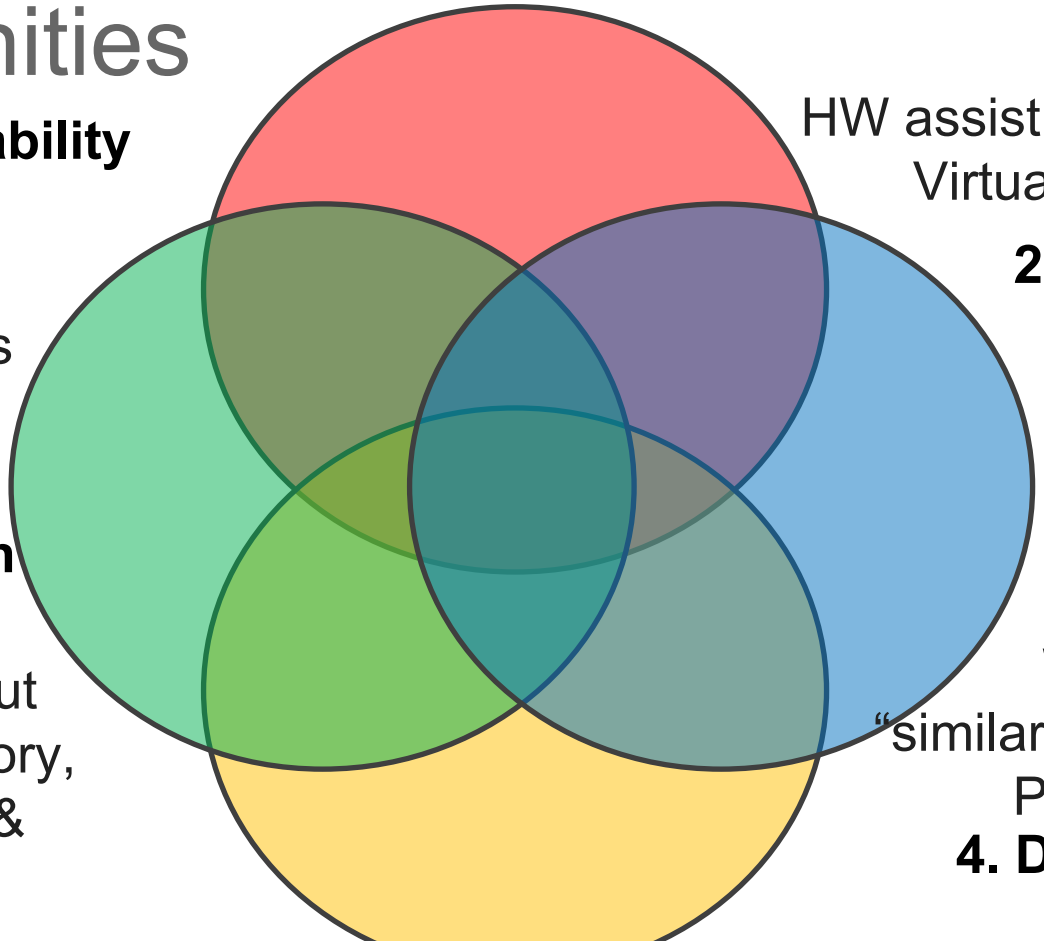
Opportunities

1. Programmability

Whither global model/runtime? DAG of streams for SoCs?

HW assist for scheduling? Virtualize & partition?

2. Concurrency



3. Communication

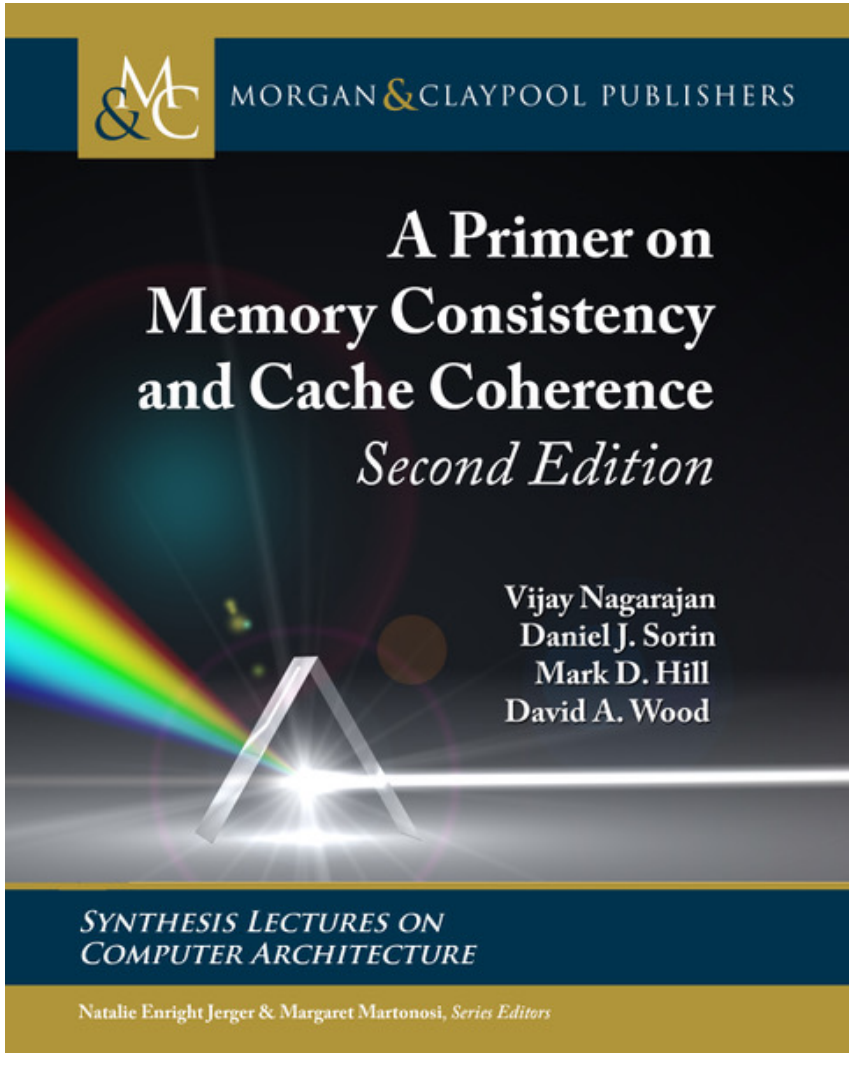
How should SW stack reason about local/global memory, caches, queues, & scratchpads?

When combine "similar" accelerators? Power vs. area?

4. Design Space

Hennessy & Patterson: A New Golden Age for Computer ~~Architecture~~ Science

New Feb 2020!


MORGAN & CLAYPOOL PUBLISHERS
A Primer on Memory Consistency and Cache Coherence
Second Edition
Vijay Nagarajan
Daniel J. Sorin
Mark D. Hill
David A. Wood
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE
Natalie Enright Jerger & Margaret Martonosi, Series Editors