Accelerator-level Parallelism

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

@ Yale University, September 2019

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

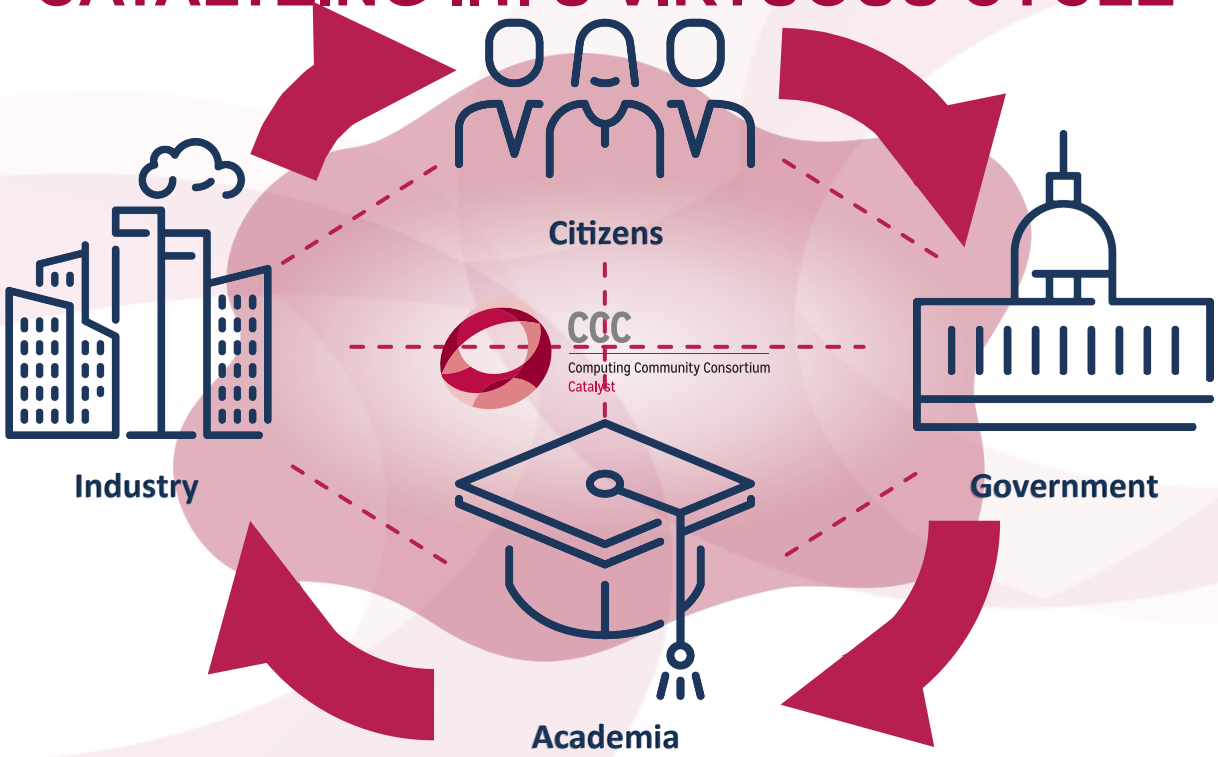
Mark D. Hill
University of Wisconsin-Madison

Abstract:
Computer system performance has improved due to creatively using more transistors (Moore's Law) in parallel via bit-, instruction-, thread-, and data-level parallelism. With the slowing of technology scaling, the only known way to further improve computer system performance under energy constraints is to employ hardware **accelerators**. Each accelerator is a hardware component that executes a targeted computation class faster and usually with (much) less energy. Already today, many chips in mobile, edge and cloud computing concurrently employ multiple accelerators in what we call **accelerator-level parallelism (ALP).**

This talk develops our hypothesis that ALP will spread to computer systems more broadly. ALP is a promising way to dramatically improve power-performance to enable broad, future use of deep AI, virtual reality, self-driving cars, etc. To this end, we review past parallelism levels and the ALP already present in mobile systems on a chip (SoCs). We then aid understanding of ALP with the **Gables** model and charge computer science researchers to develop better ALP "best practices" for: targeting accelerators, managing accelerator concurrency, choreographing inter-accelerator communication, and productively programming accelerators. This joint work with Vijay Janapa Reddi of Harvard is at: https://arxiv.org/abs/1907.02064

Biography:
Mark D. Hill (http://www.cs.wisc.edu/~markhill) is John P. Morgridge Professor and Gene M. Amdahl Professor of Computer Sciences at the University of Wisconsin-Madison, where he also has a courtesy appointment in Electrical and Computer Engineering. His research interests include parallel-computer system design, memory system design, and computer simulation. He received the 2019 Eckert-Mauchly Award and is a fellow of IEEE and the ACM. He serves as Chair of the Computer Community Consortium (2018-19) and served as Wisconsin Computer Sciences Department Chair 2014-2017. Hill has a PhD in computer science from the University of California, Berkeley.

COMPUTING COMMUNITY CONSORTIUM (CCC): CATALYZING I.T.'S VIRTUOUS CYCLE



Get involved w/ white papers, workshops, & advocating I.T. research (to do good & make your research better)

Community Consortium

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>

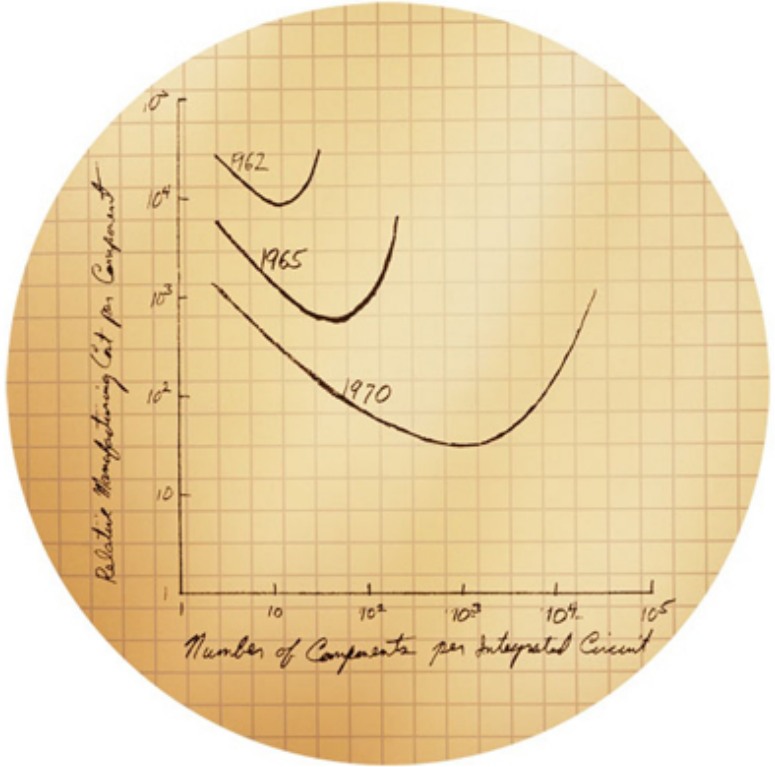
Required innovations in algorithms, applications, programming languages, … , & system software

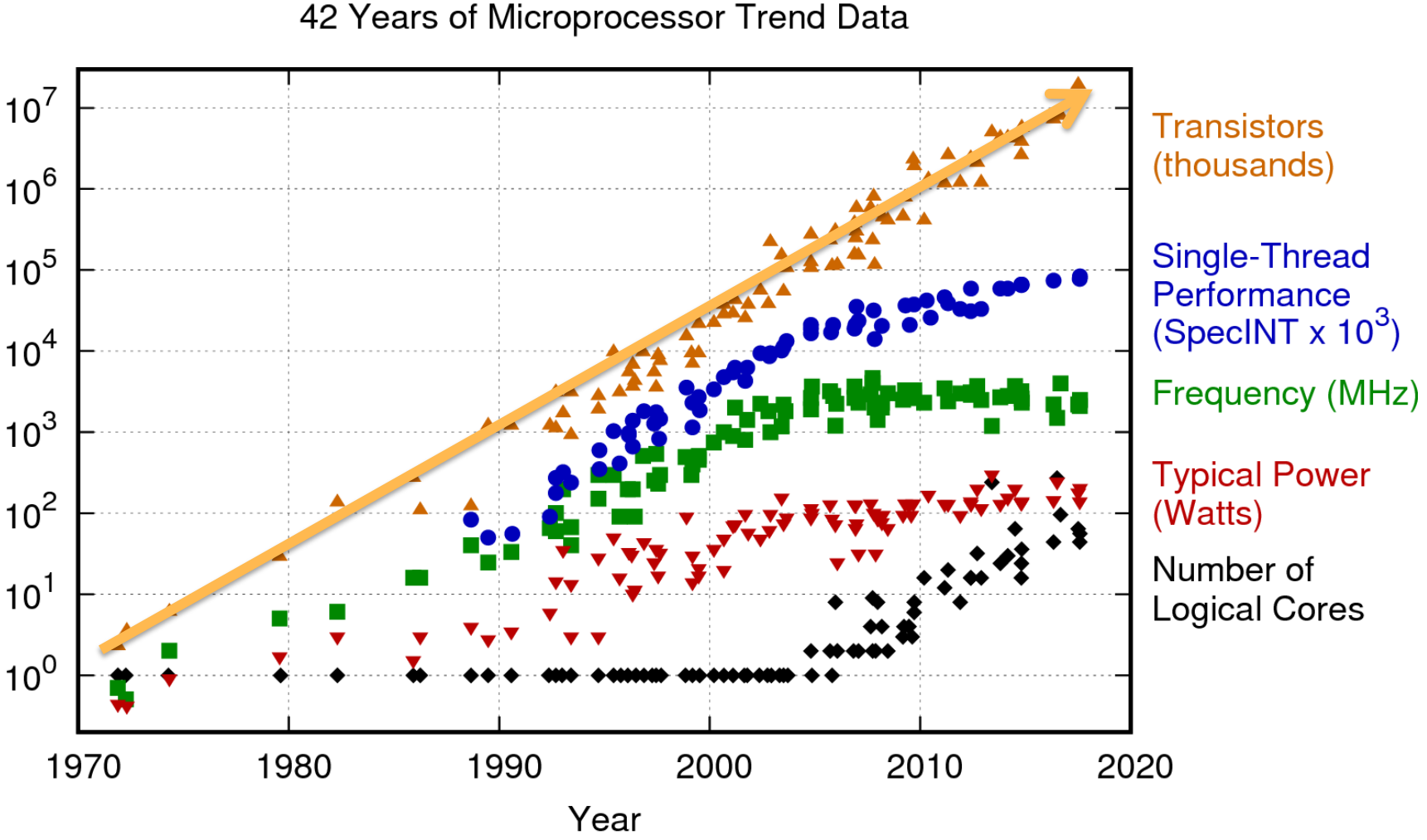
Key (invisible) enablers (cost-)performance gains

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

Moore's Law – 1965 Paper

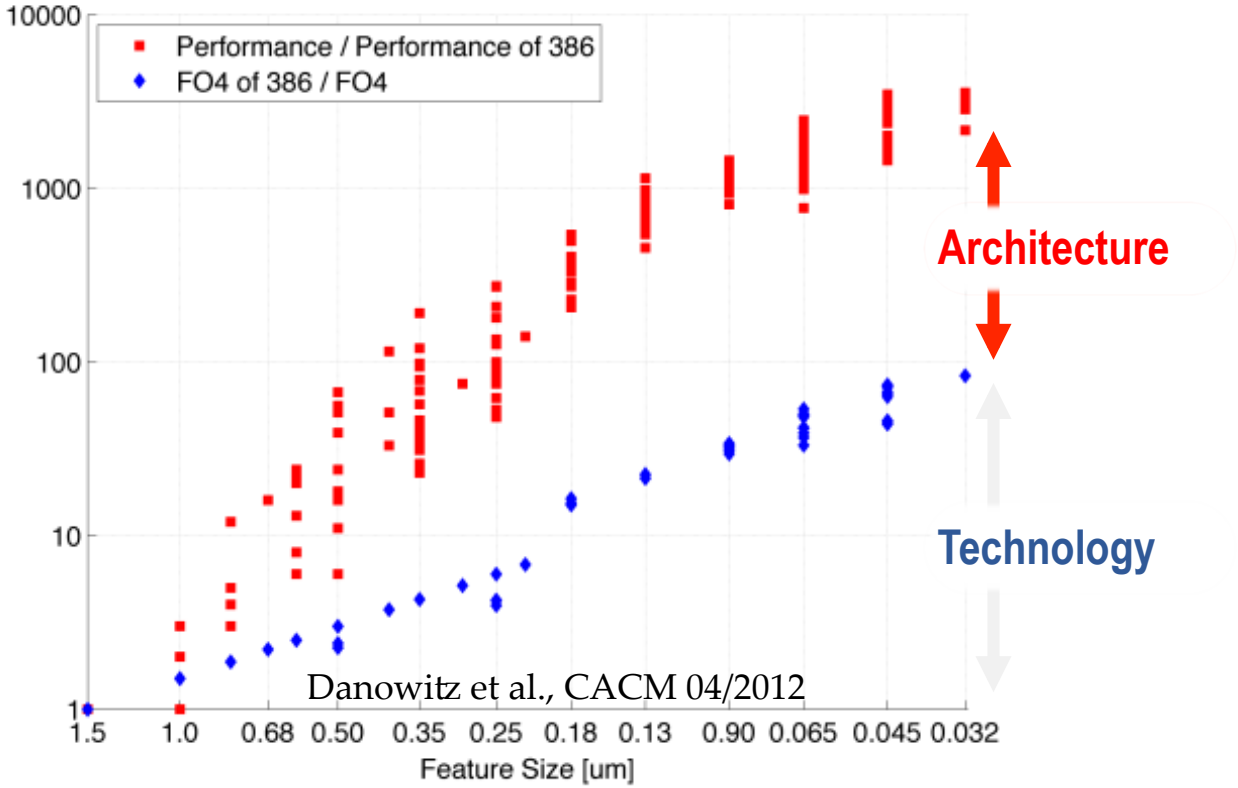
- Optimal number of transistors per chip with increase with time
- Became self-fulfilling prophesy with doubling transistor count every ~ two years
- Note that transistor gain in two years equals all past gain – the power of an exponential!





Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp https://www.karlrupp.net/wp-content/uploads/2018/02/42-years-processor-trend.png

Enablers: Technology + Architecture



How did Architecture Exploit Moore's Law?

MORE (& faster) transistors ➔ even faster computers

Memory – transistors in parallel

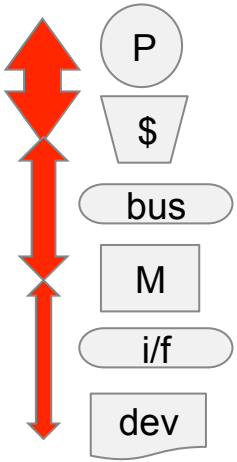
- Vast semiconductor memory (DRAM)
- Cache hierarchy for fast memory illusion

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)

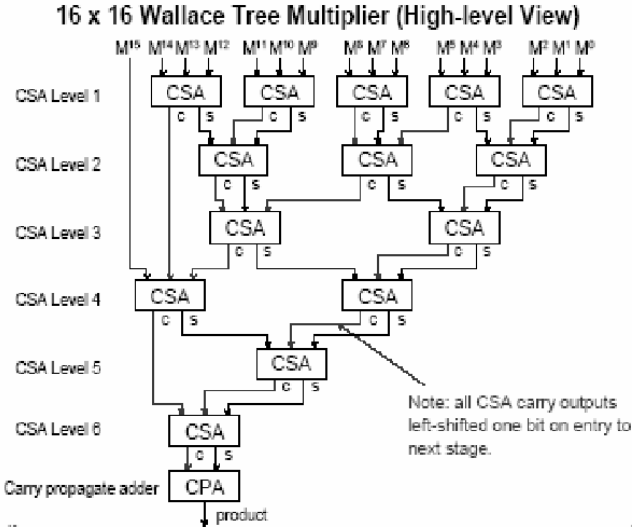
Early computers: few switches (transistors)

- ➔ 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



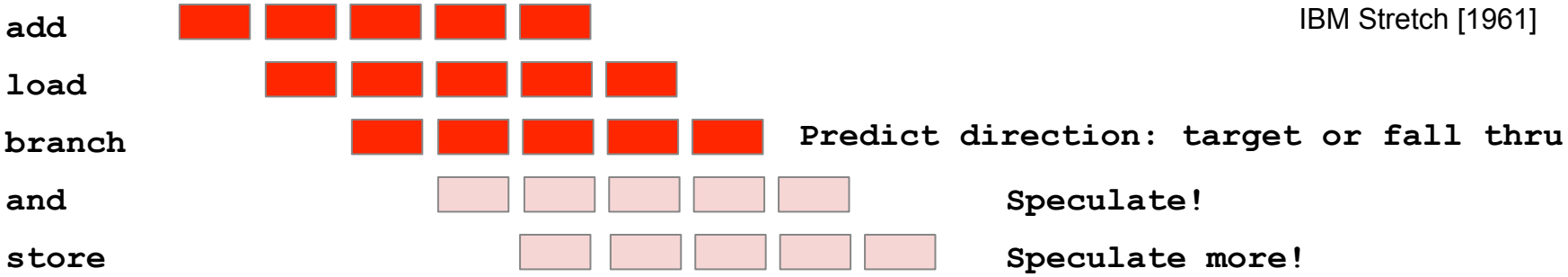
Important: Easy for software

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

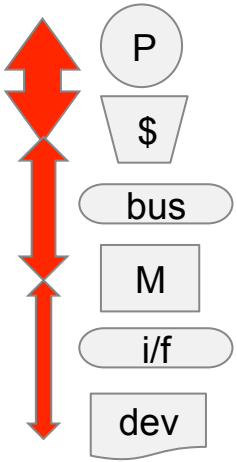


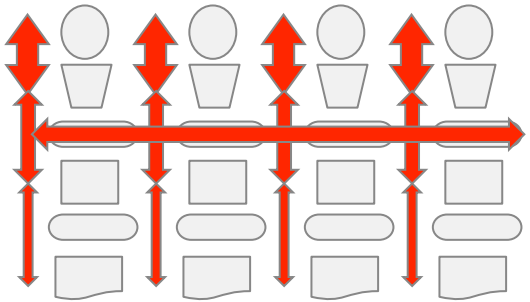
IBM Stretch [1961]

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**

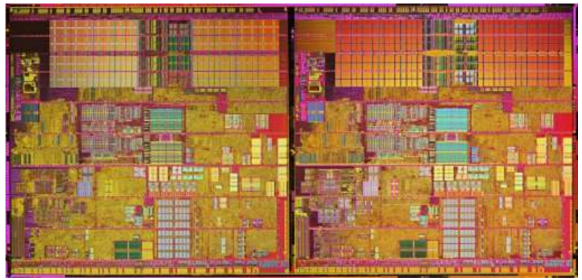
- E.g. pThreads, OpenMP, MPI

On-chip TLP called “multicore” – forced choice

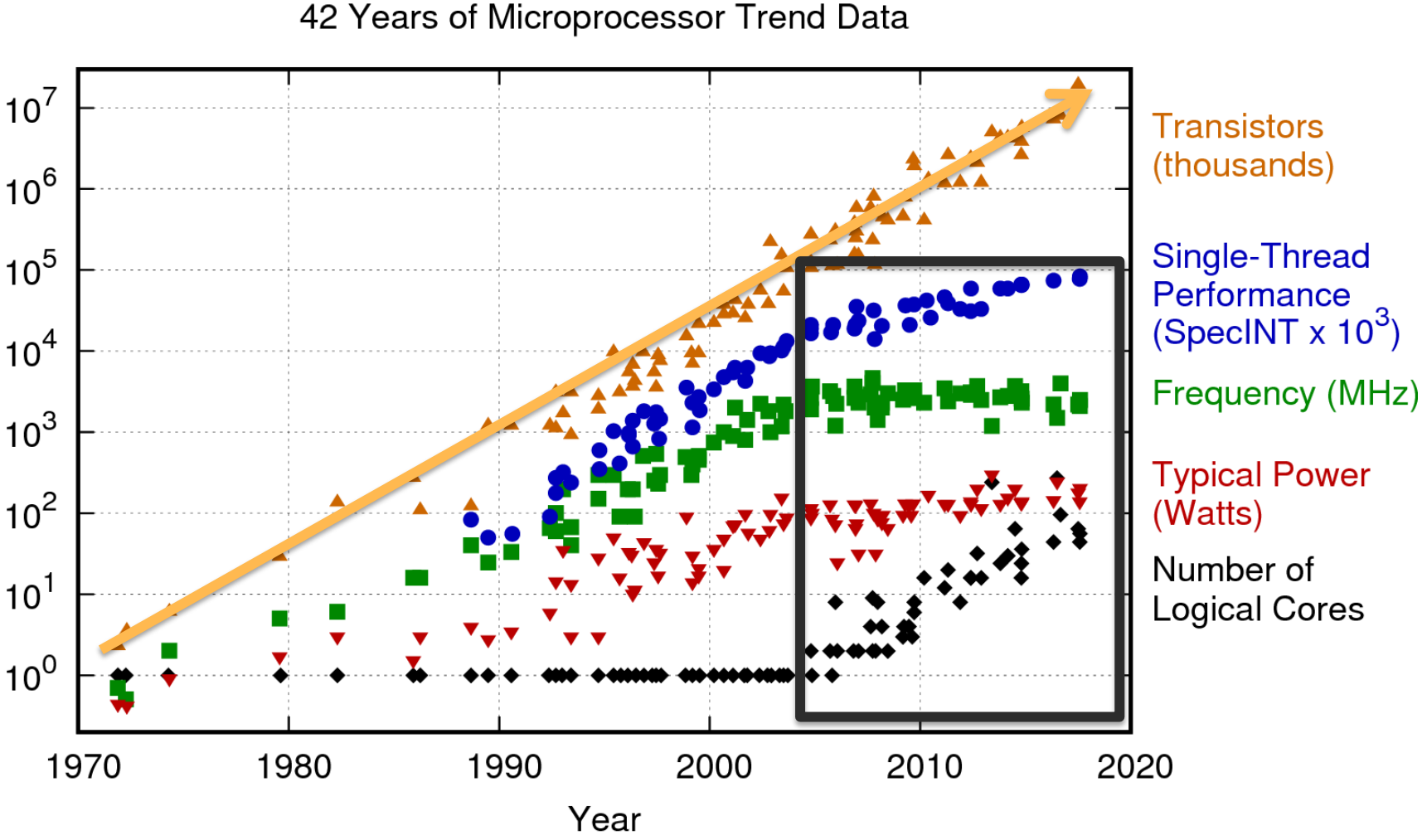
Less easy for software but

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

CDC 6600, 1964,
(TLP via multithreaded processor)

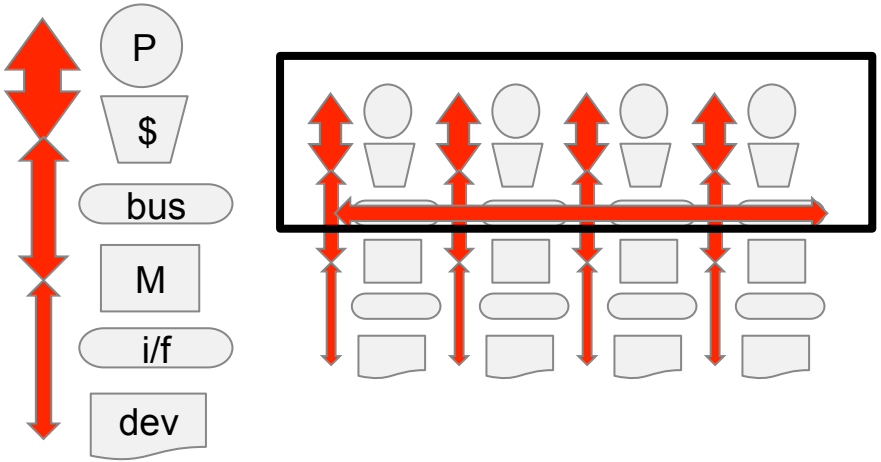


Intel Pentium Pro Extreme Edition,
early 2000s



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp https://www.karlrupp.net/wp-content/uploads/2018/02/42-years-processor-trend.png

X-level Parallelism in Computer Architecture



1 CPU

BLP+ILP

Bit/Instrn-Level Parallelism

Multicore

+ TLP

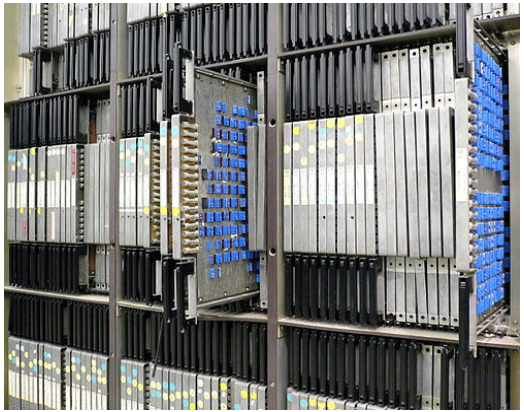
Thread-Level Parallelism

Data-level Parallelism (DLP)

Need same operation on many data items

Do with parallelism ➔ DLP

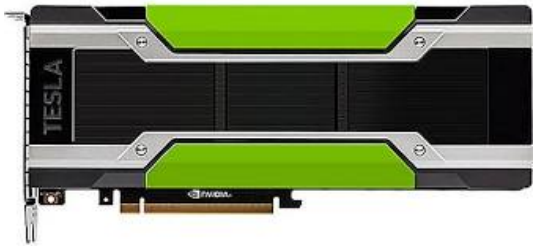
- Array of single instruction multiple data (SIMD)
- Deep pipelines like Cray vector machines
- Intel-like Streaming SIMD Extensions (SSE)



Illinois ILLIAC IV, 1966

Broad DLP success awaited General-Purpose GPUs

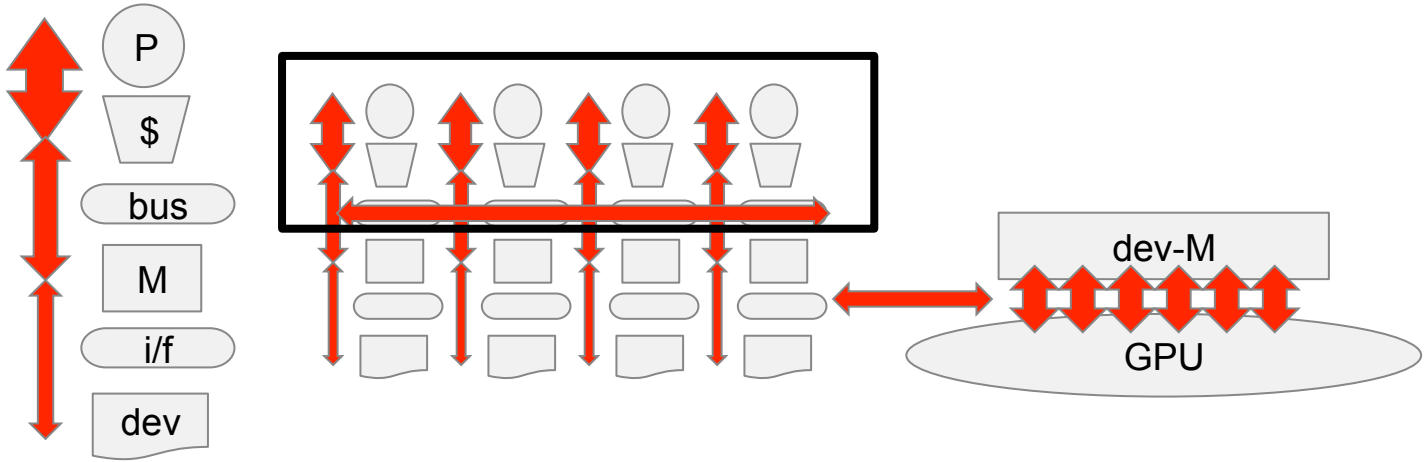
1. **Single Instruction Multiple Thread (SIMT)**
2. **SW (CUDA) & libraries (math & ML)**
3. **Experimentation as $1-10K not $1-10M**



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

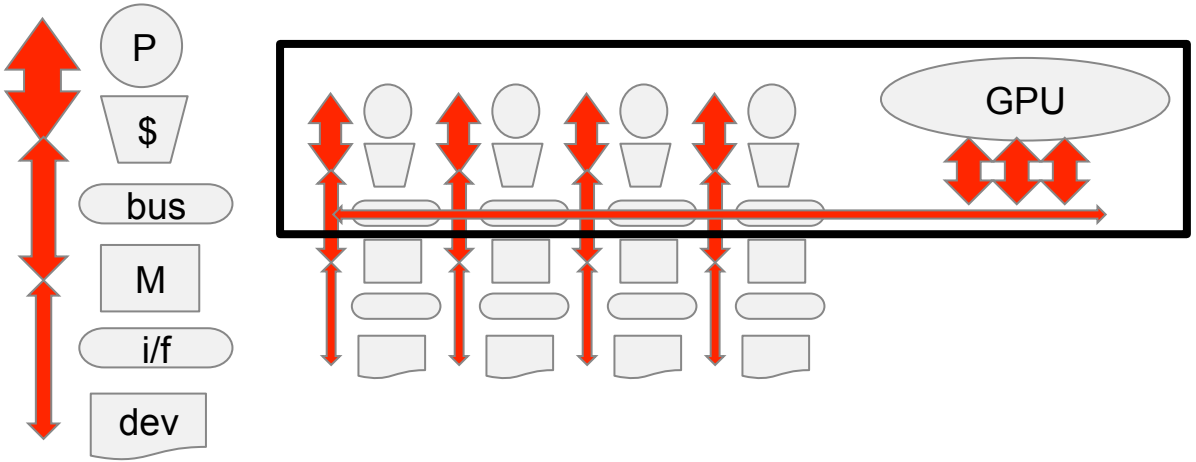
Thread-Level Parallelism

\+ **Discrete GPU**

+ DLP

Data-Level Parallelism

X-level Parallelism in Computer Architecture

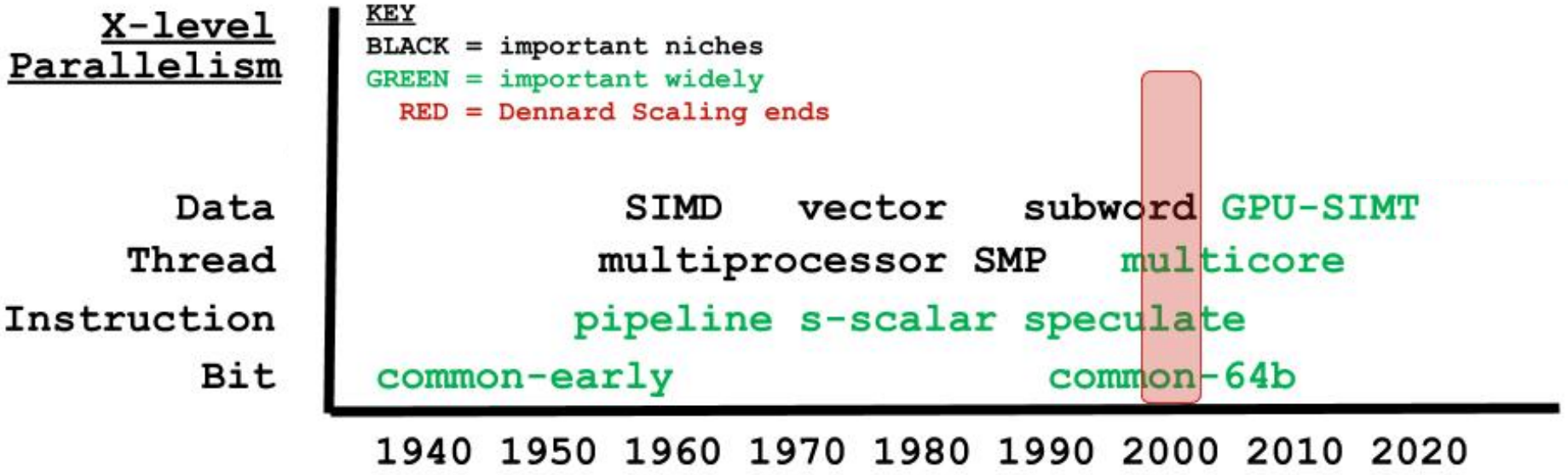


1 CPU | **Multicore** + **Integrated GPU**

BLP+ILP | + TLP | + DLP

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

X-level Parallelism in Computer Architecture



Outline

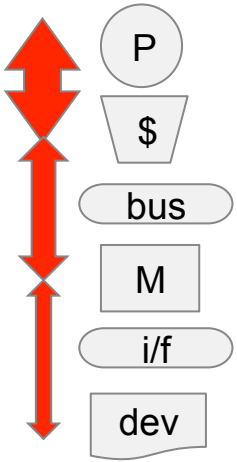
I. Computer History & X-level Parallelism

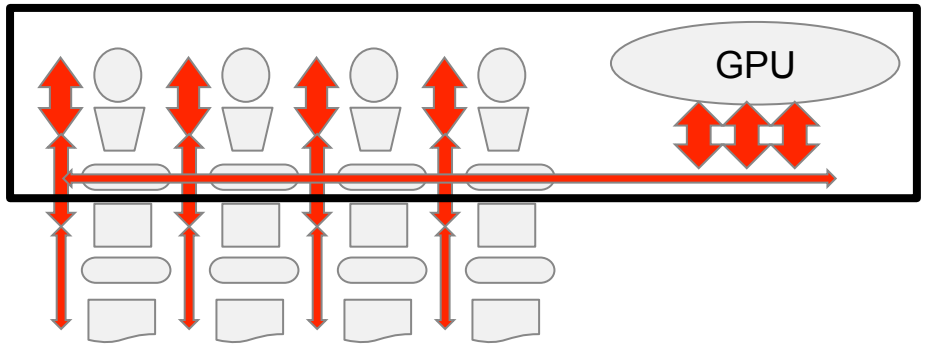
II. Mobile SoCs as ALP Harbinger

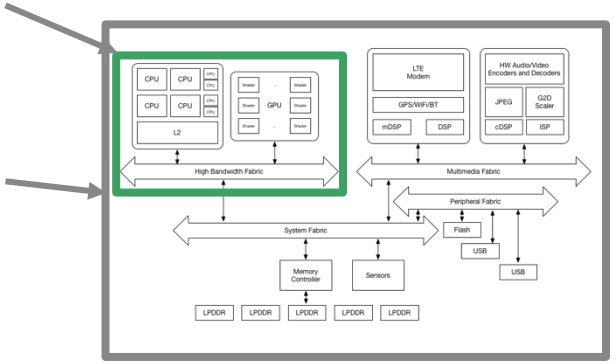
III. Gables ALP SoC Model

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

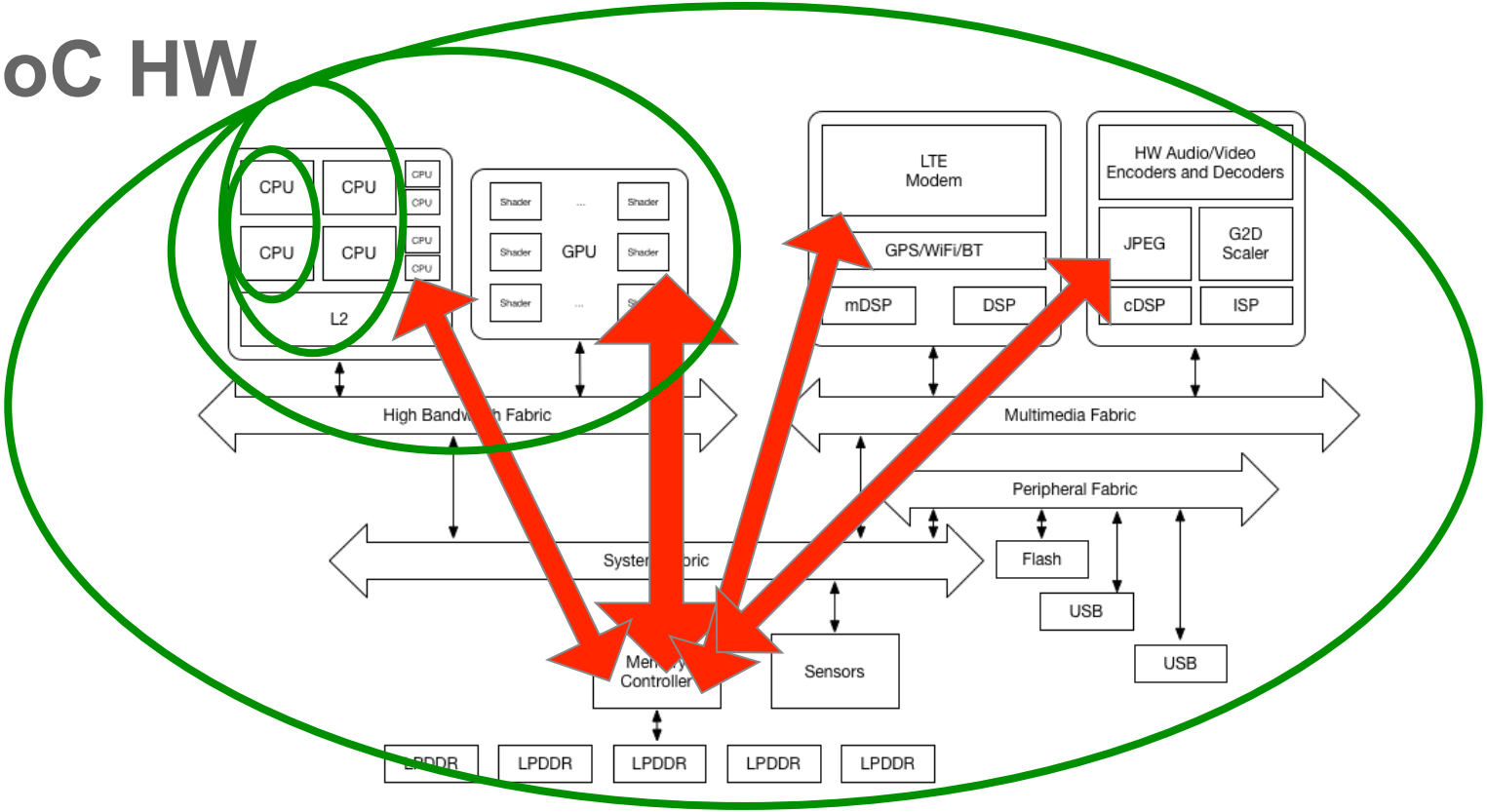
Data-Level Parallelism

System on a Chip (SoC)

+ ALP

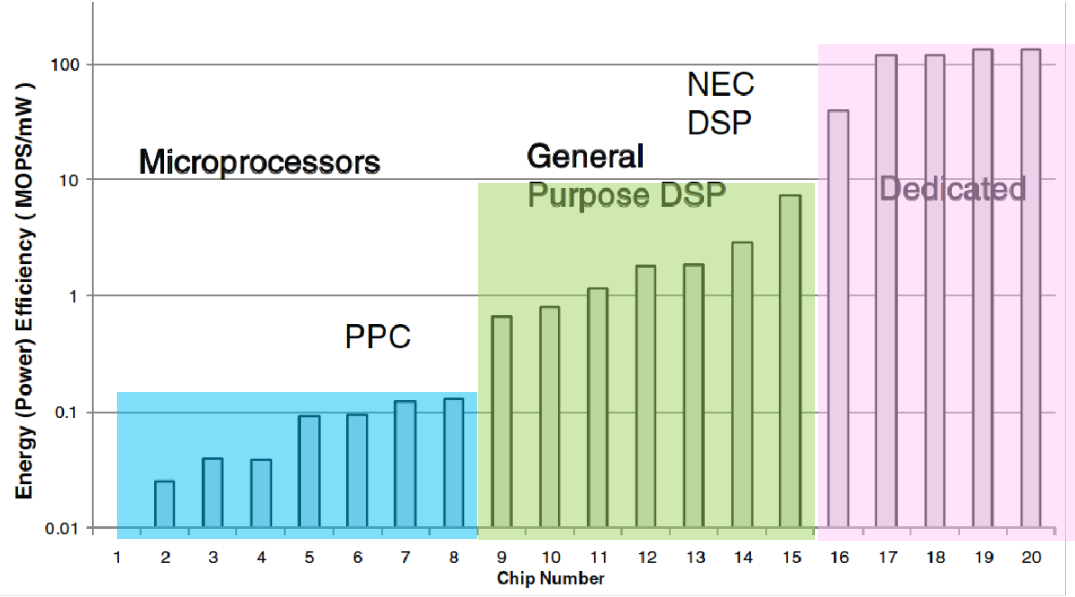
Accelerator-Level Parallelism

Mobile SoC HW



Potential for Specialized Accelerators (IPs)

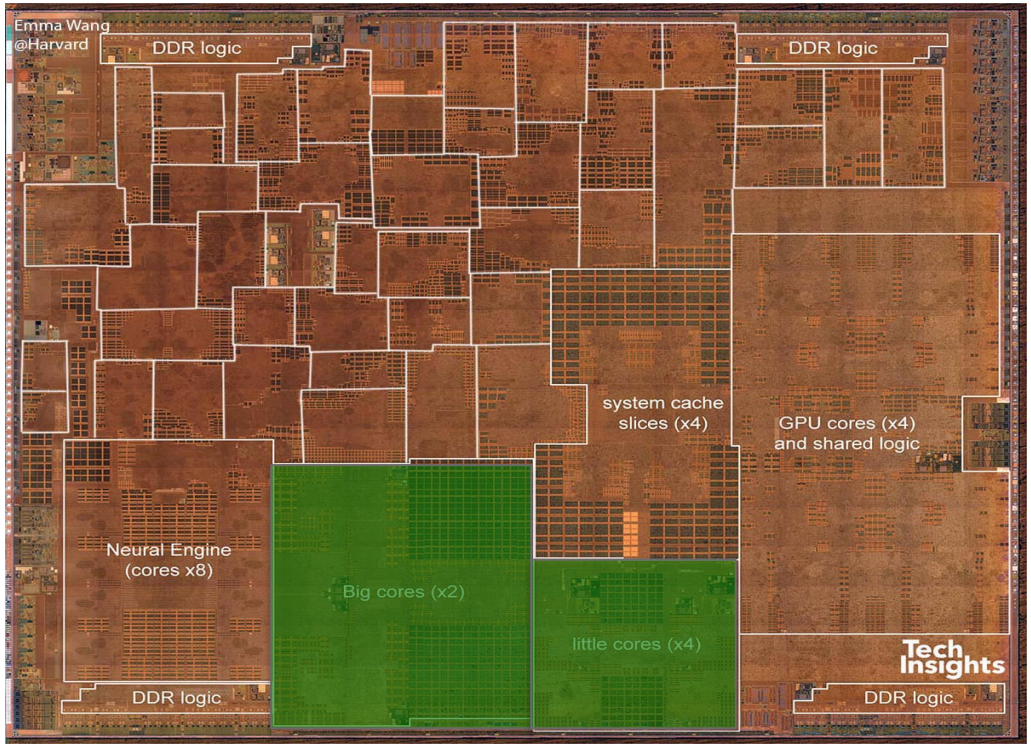
Accelerator is a hardware component that executes a targeted computation class faster & usually with (much) less energy.



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)

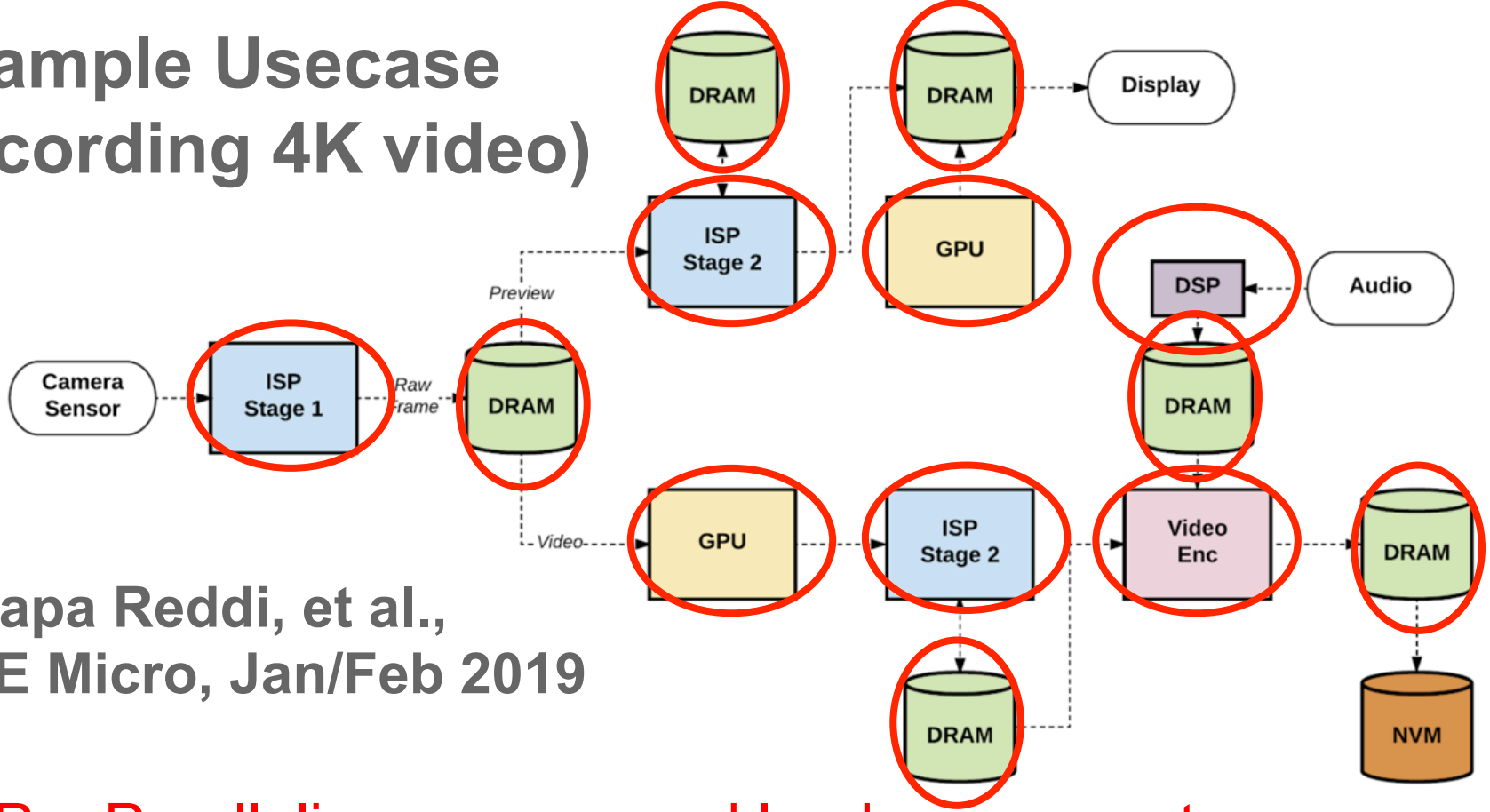


2019 Apple A12 w/ 42 accelerators

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						

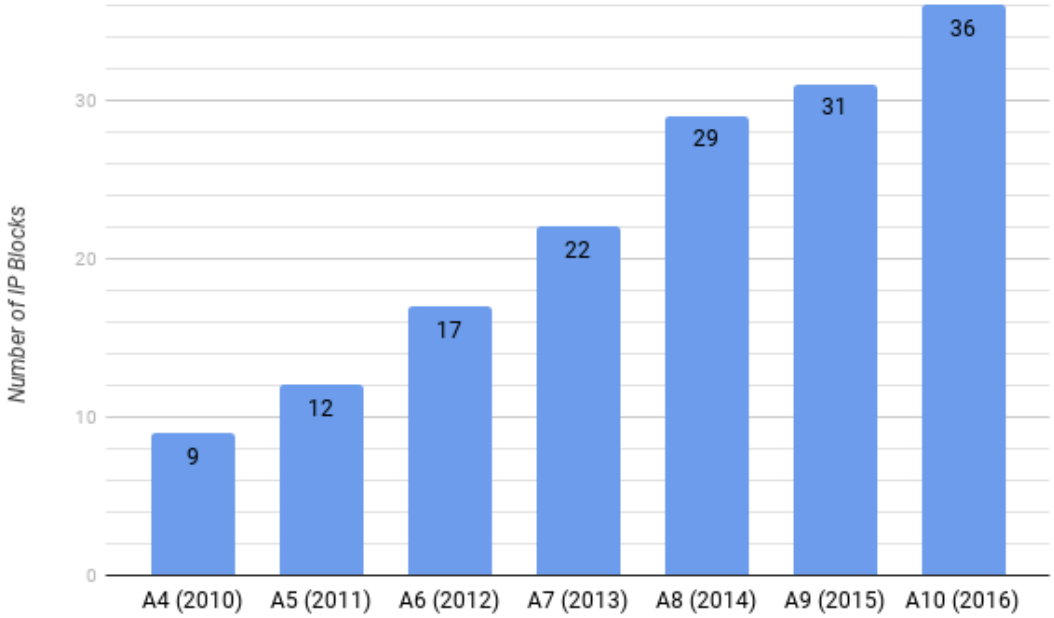
Must run each usecase sufficiently fast -- no need faster

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

For each usecase, how much acceleration for each IP?

Mobile SoCs Hard To Design

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



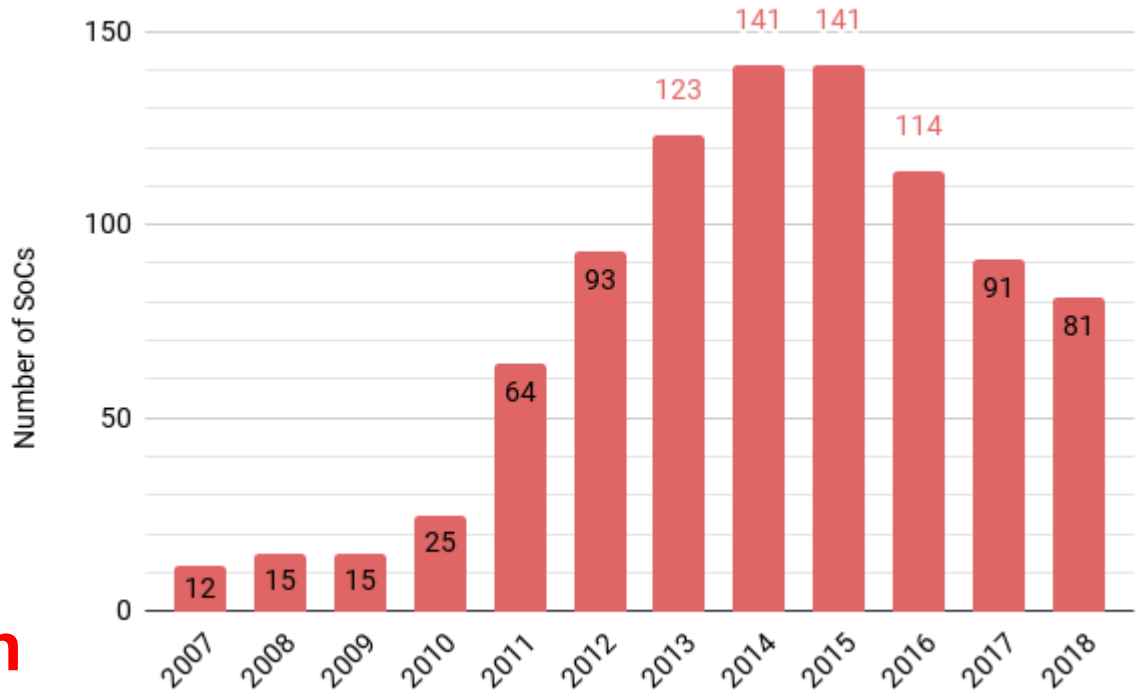
Which accelerators? How big? How to even start?

Mobile SoCs Hard To Program For and Select

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

Diversity hinders use [Facebook, HPCA'19]

What SoC abstraction should SW use?



Outline

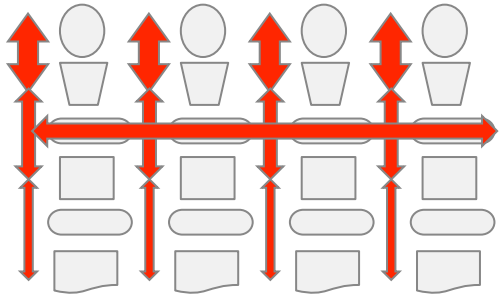
I. Computer History & X-level Parallelism

II. Mobile SoCs as ALP Harbinger

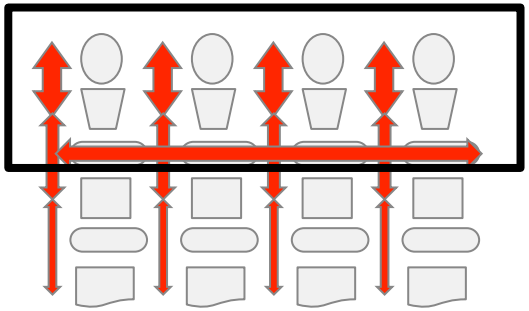
III. Gables ALP SoC Model (ok to get lost)

IV. Call to Action for Accelerator-level Parallelism

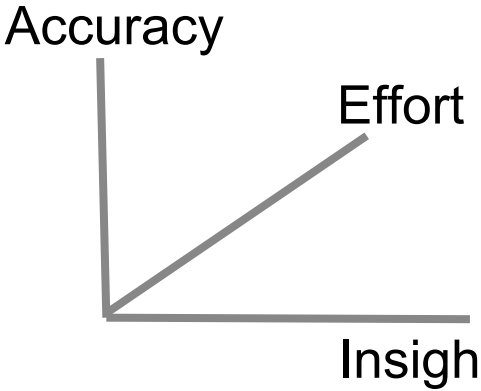
Computer Architecture & Models



Multiprocessor &
Amdahl's Law



Multicore &
Roofline



Models vs Simulation

- More insight
- Less effort
- But less accuracy

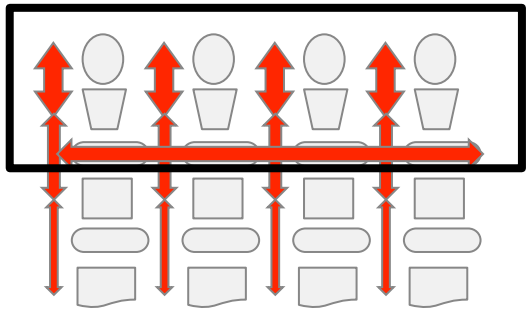
Models give first answer, not final answer

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

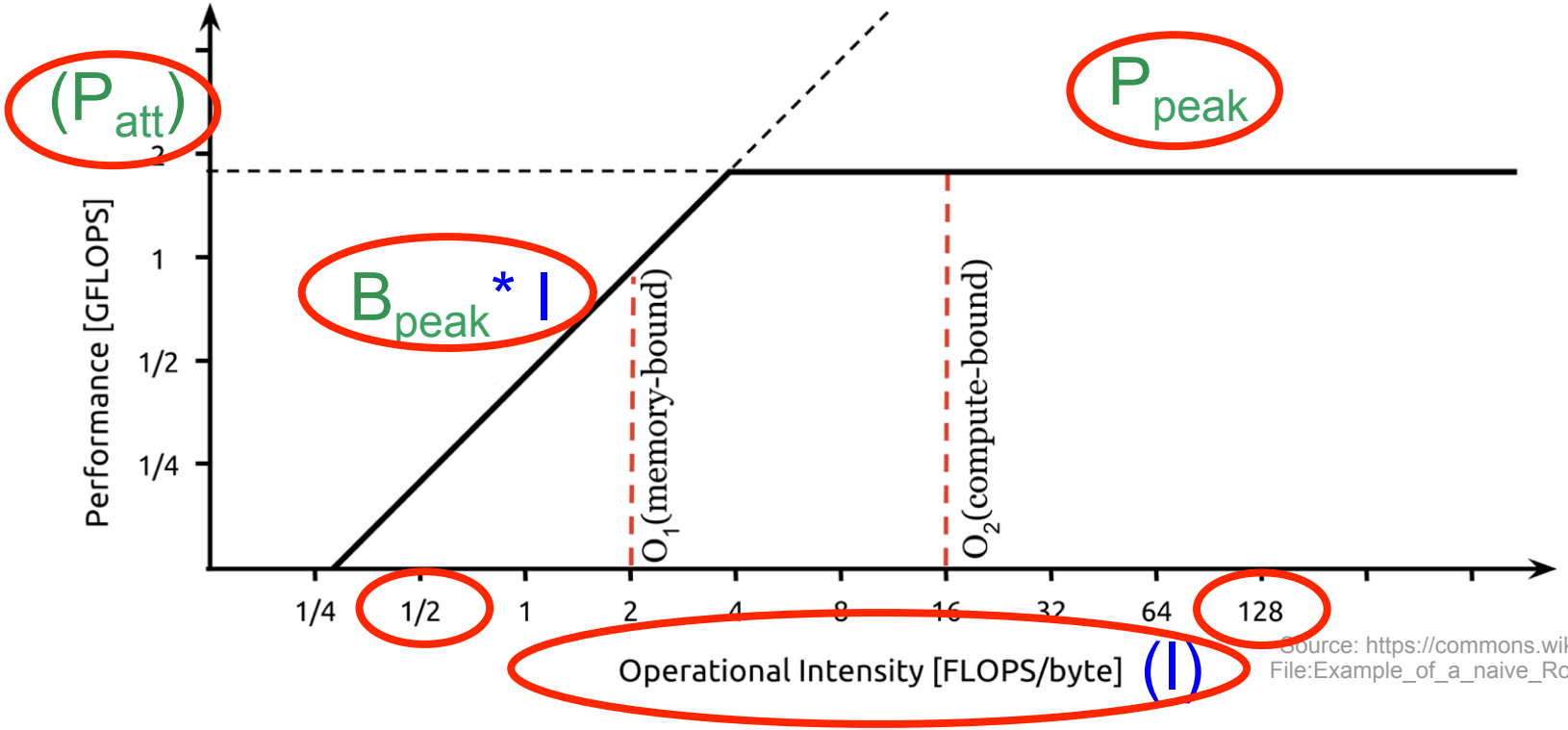


Multicore SW

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

Output P_{att} = upper bound on performance attainable

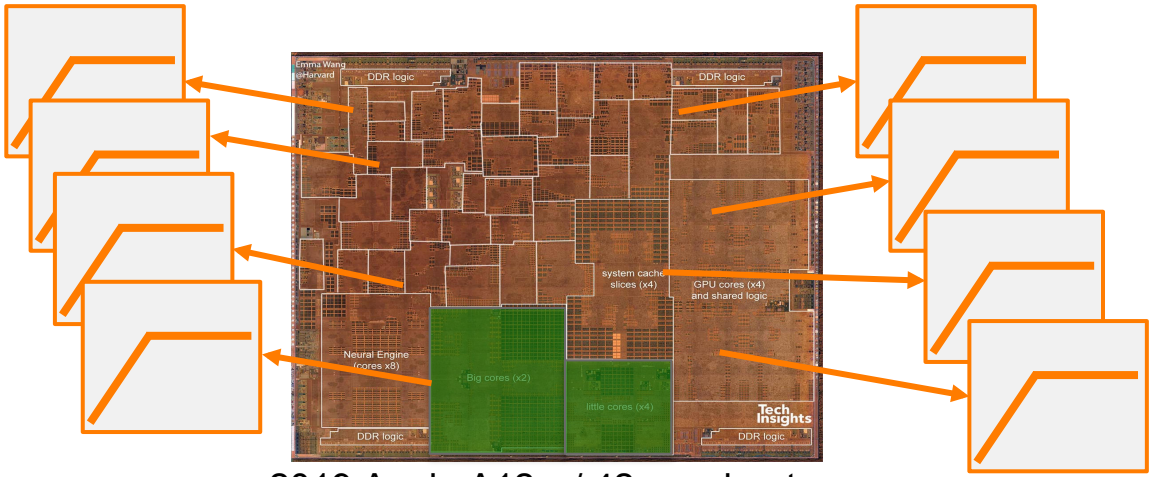
Roofline for Multicore Chips, 2009



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

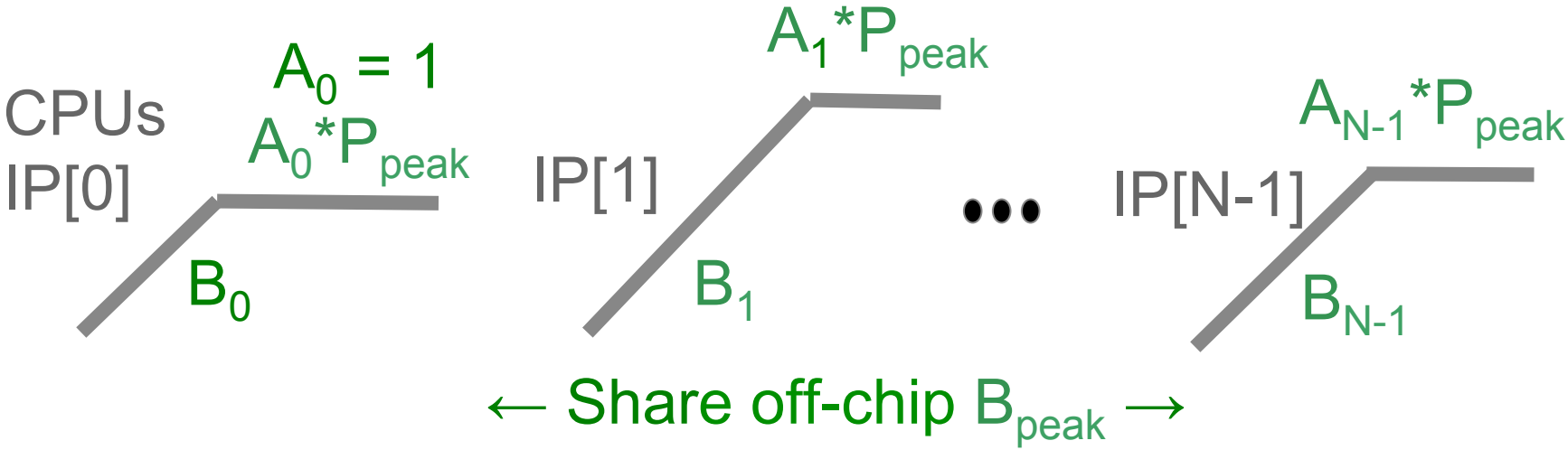


2019 Apple A12 w/ 42 accelerators

Gables uses Roofline per IP to provide first answer!

- HW: select & size accelerators
- SW: optimize for a “gabled roof?”

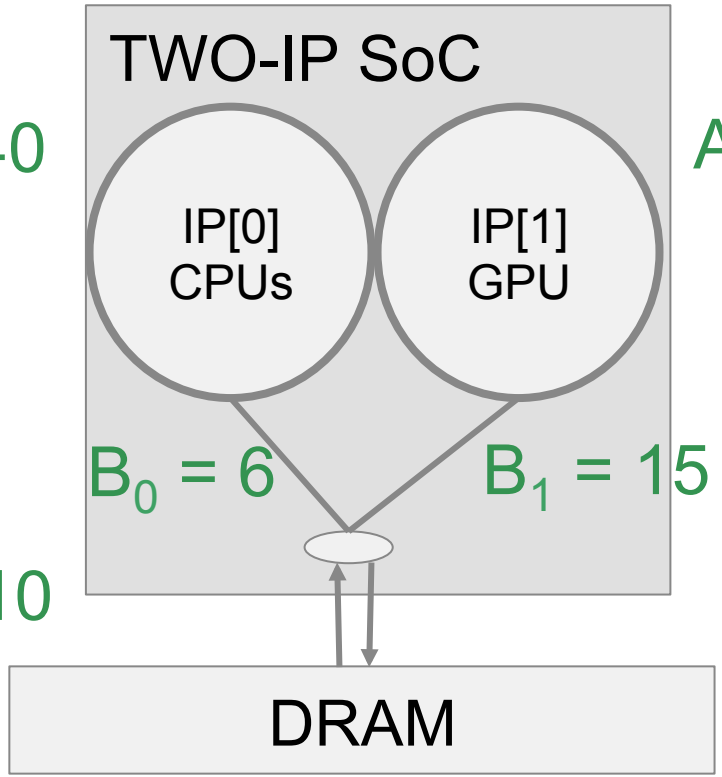
Gables for N IP SoC



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



$P_{peak} = 40$

$B_{peak} = 10$

$B_0 = 6$

$B_1 = 15$

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

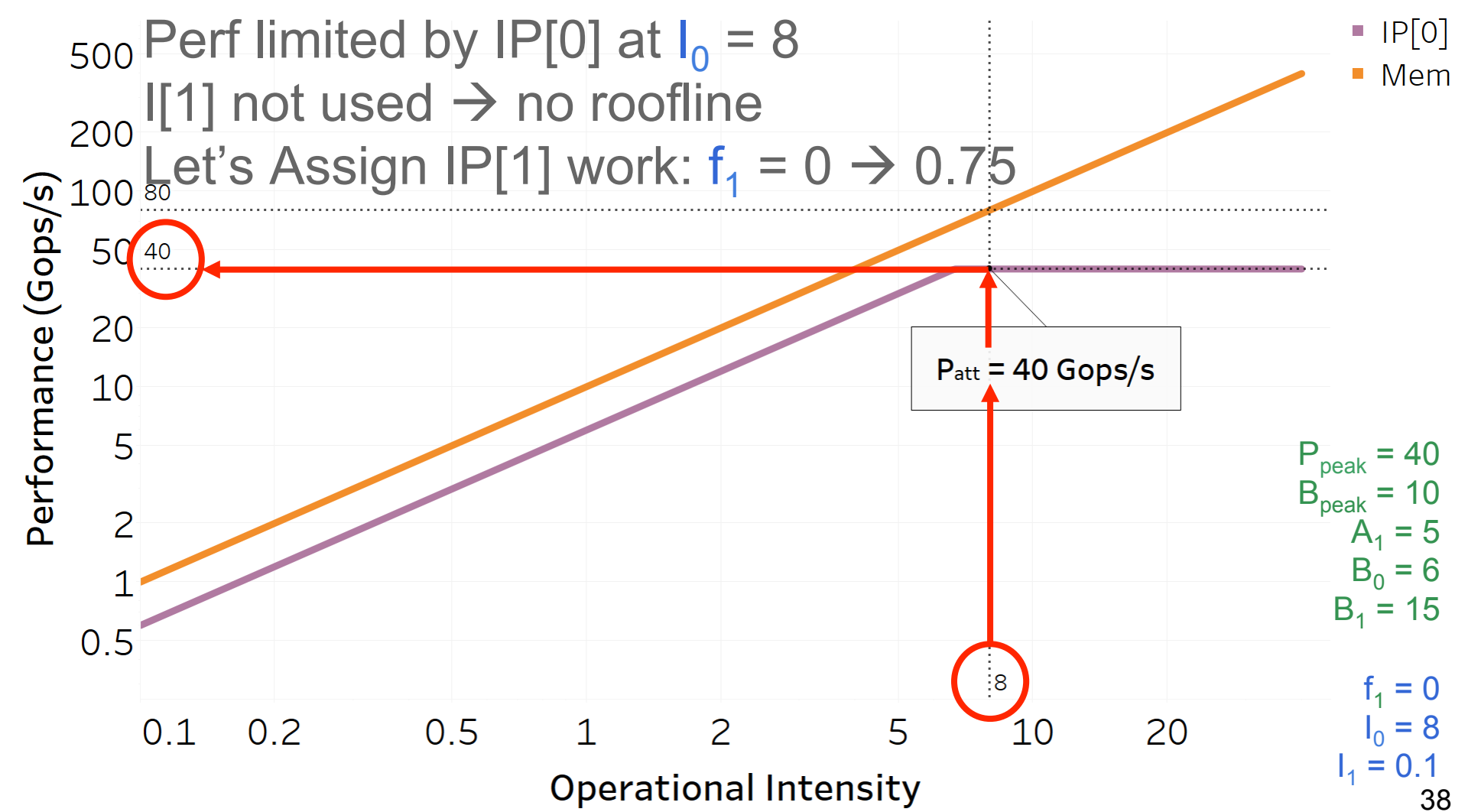
Workload (Usecase):

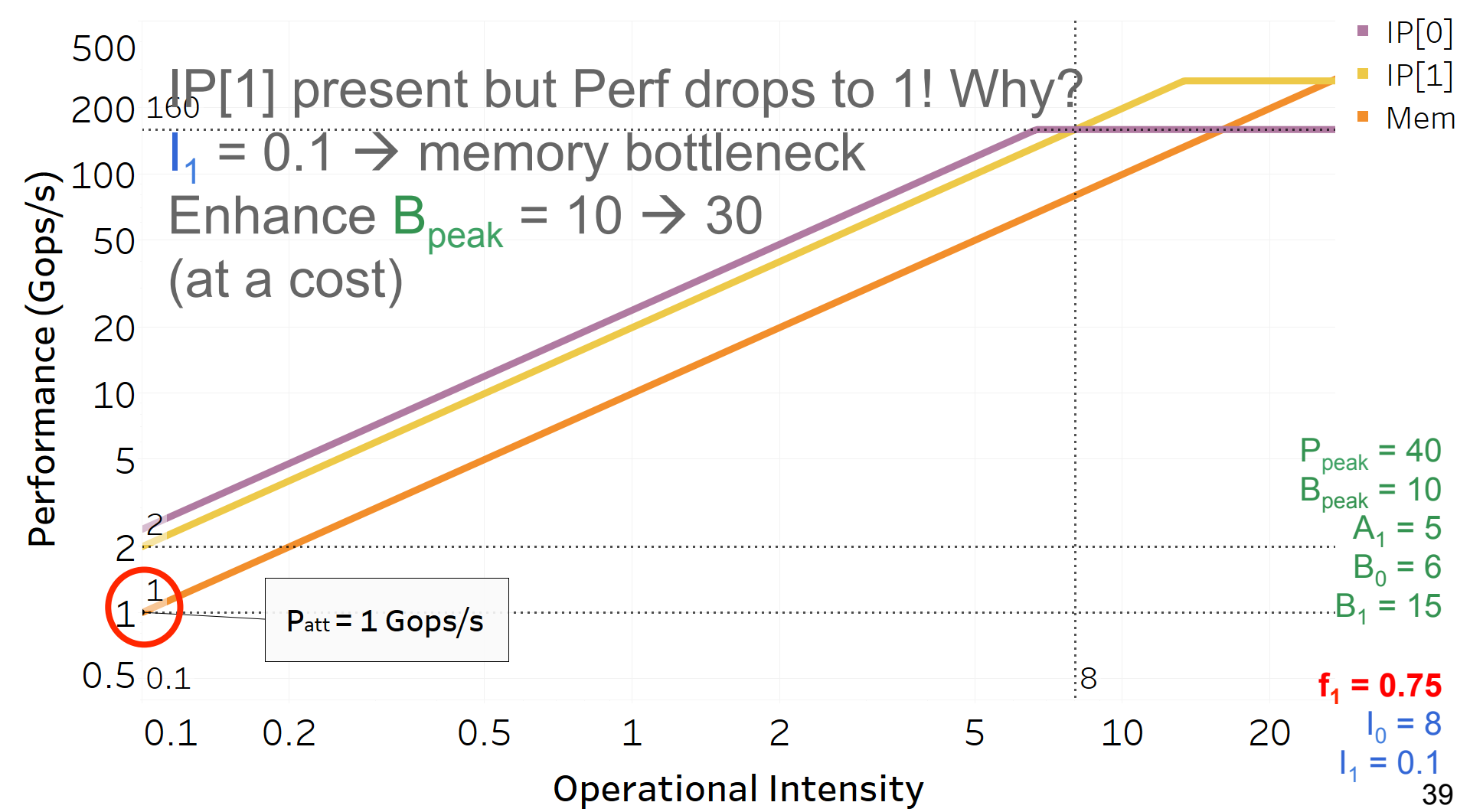
$f_0 = 1$ & $f_1 = 0$

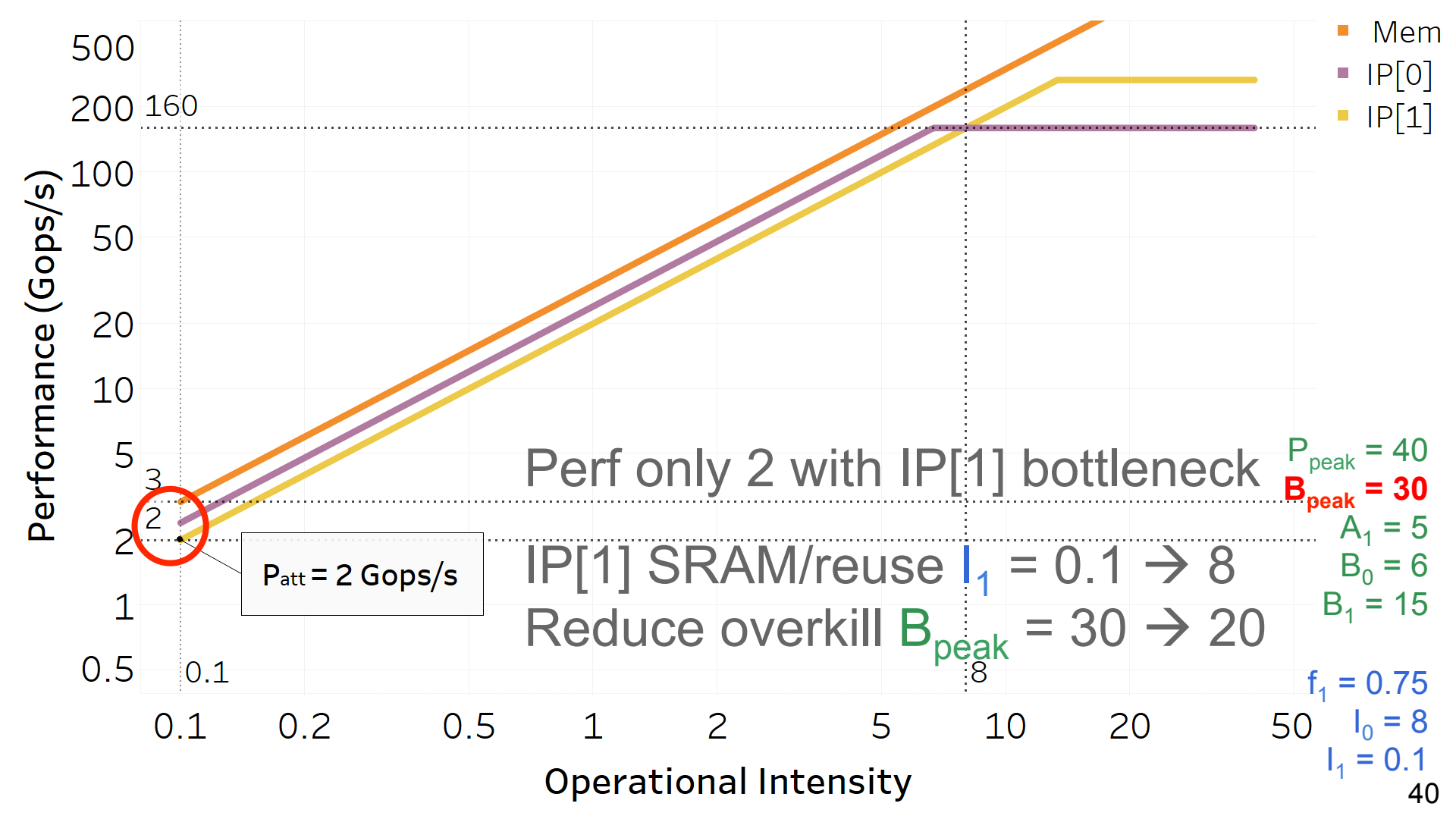
$I_0 = 8$ = good caching

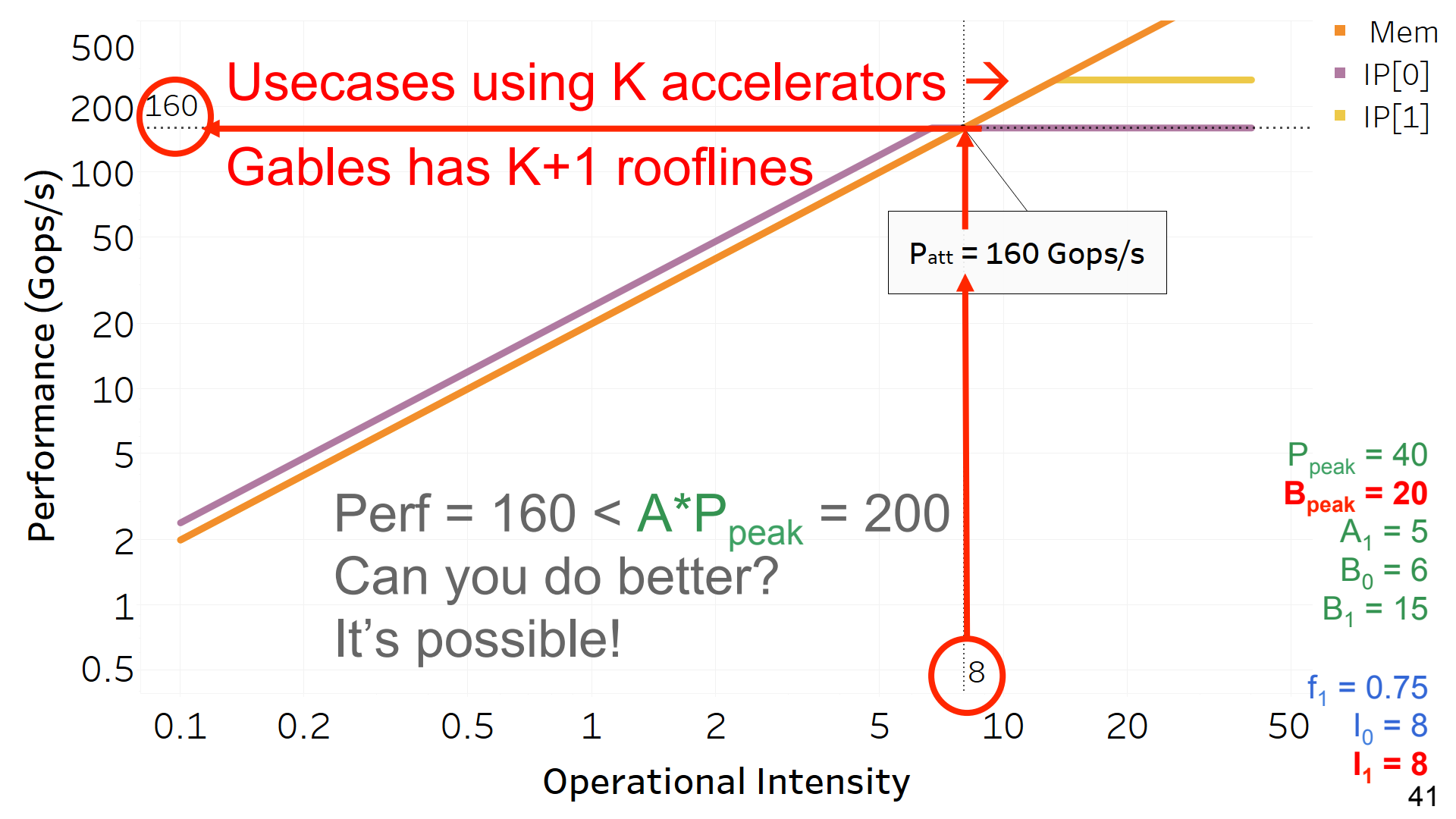
$I_1 = 0.1$ = latency tolerant

Performance?


Perf limited by IP[0] at I_0 = 8
I[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
P_{peak} = 40
B_{peak} = 10
A_1 = 5
B_0 = 6
B_1 = 15
f_1 = 0
I_0 = 8
I_1 = 0.1

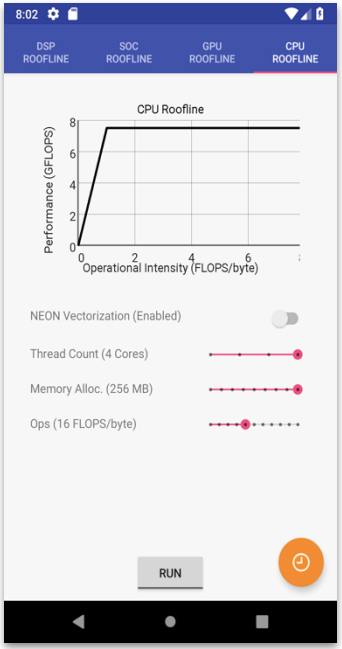

IP[1] present but Perf drops to 1! Why?
I_1 = 0.1 → memory bottleneck
Enhance B_{peak} = 10 → 30
(at a cost)
IP[0]
IP[1]
Mem
Performance (Gops/s)
500
200
160
100
50
20
10
5
2
2
1
1
0.5
0.1
P_{att} = 1 Gops/s
P_{peak} = 40
B_{peak} = 10
A_1 = 5
B_0 = 6
B_1 = 15
f_1 = 0.75
I_0 = 8
I_1 = 0.1
8
0.1
0.2
0.5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

Gables Home Page

Model Extensions



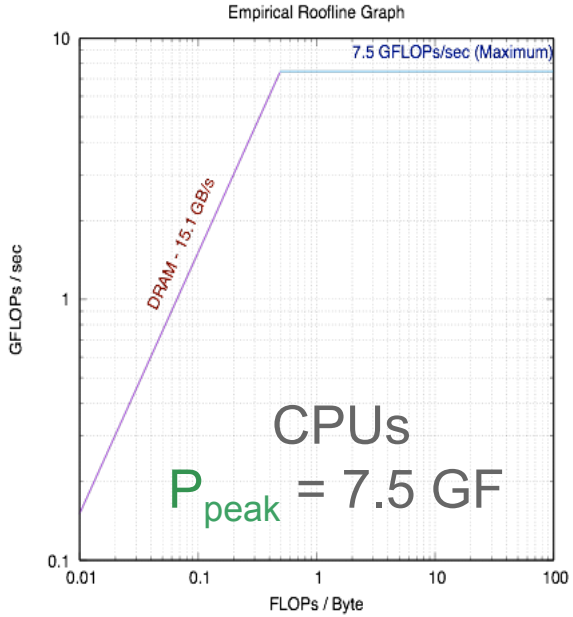
Interactive tool

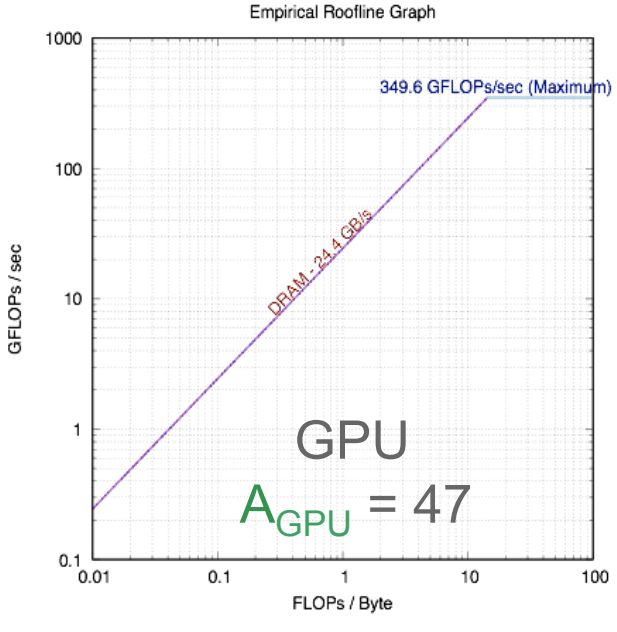
Gables Android Source at GitHub

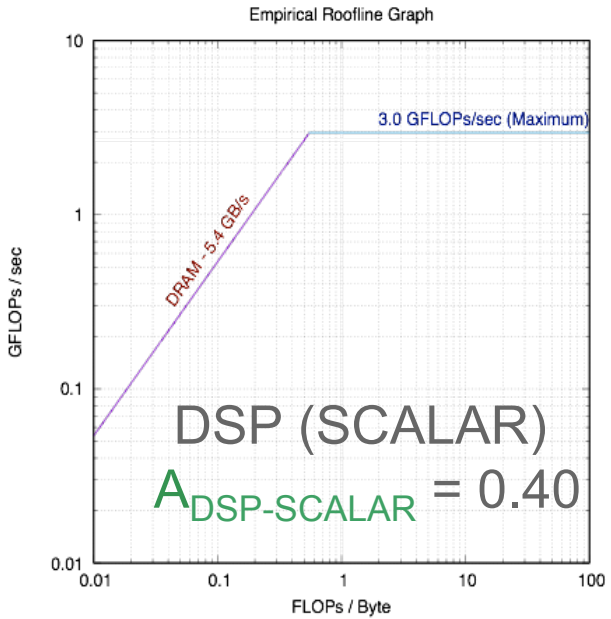
http://research.cs.wisc.edu/multifacet/gables/

µBenchmark w/ Qualcomm Snapdragon™ 835

- All elements load from array & vary FP SP op intensity
- Finds empirical lower bound on rooflines

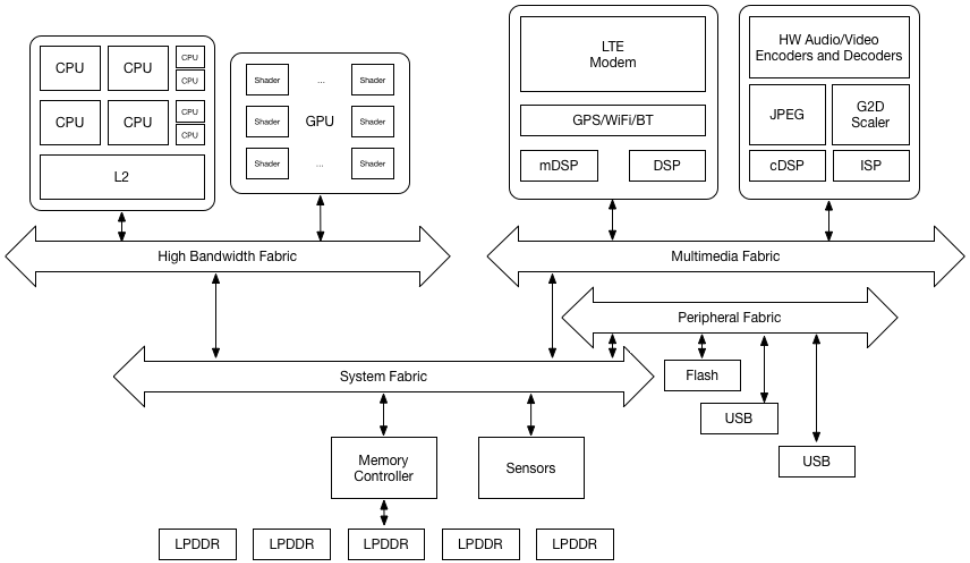


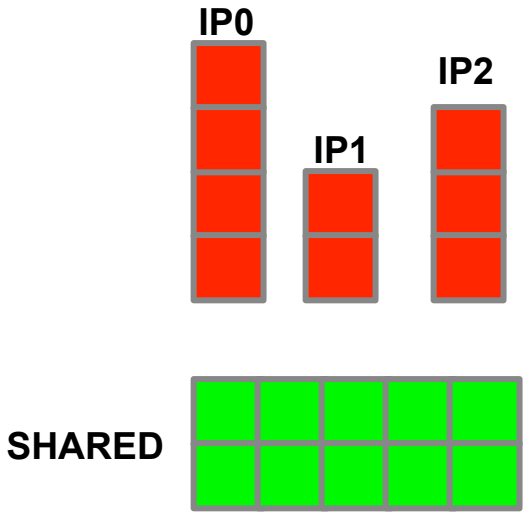




- Preliminary evidence that multiple rooflines useful

Case Study: Allocating SRAM

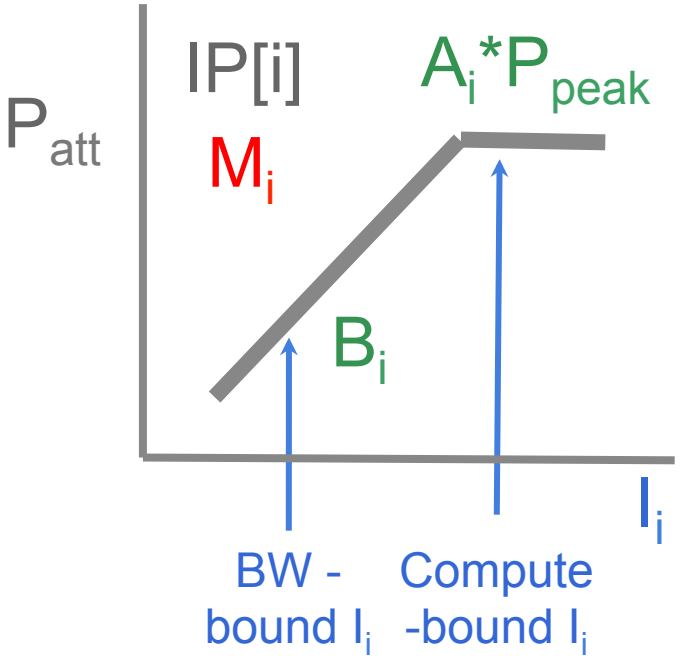




Where SRAM?

- Private w/i each IP
- Shared resource

What determines I_i?



SW Usecase (most important)

- Dense v. sparse matrices
- E.g. vision v. audio ML

Hardware

More A_i toward BW-bound (recall f_i too!)

More B_i toward compute-bound

More M_i toward compute-bound **if reuse**

Whither I_i as function of M_i?

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

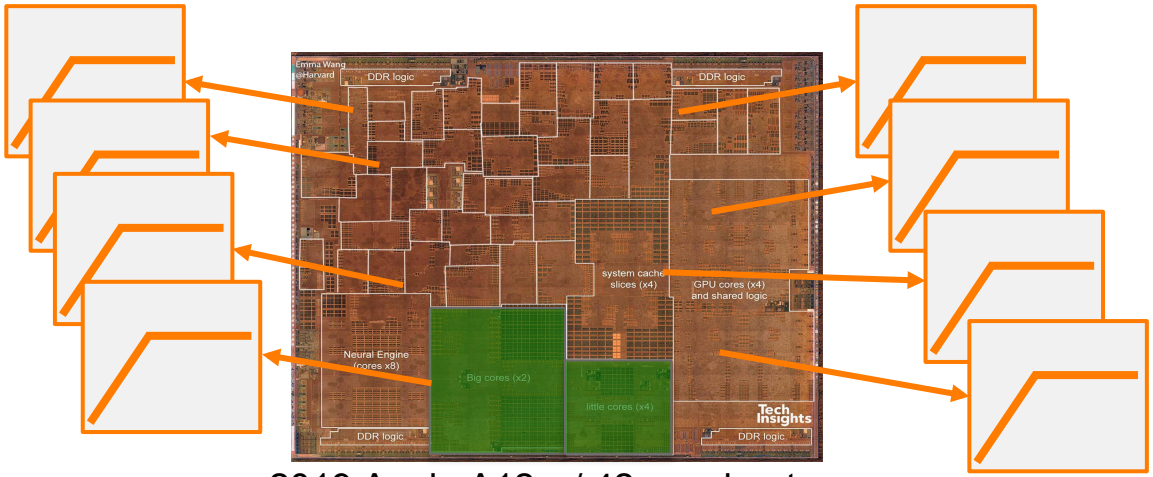
Compute v. Communication: Op. Intensity (I) = #operations / #off-chip bytes

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

Mobile System on Chip (SoC) & Gables



2019 Apple A12 w/ 42 accelerators

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

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

Gables is perhaps a first answer, but not a final answer

Outline

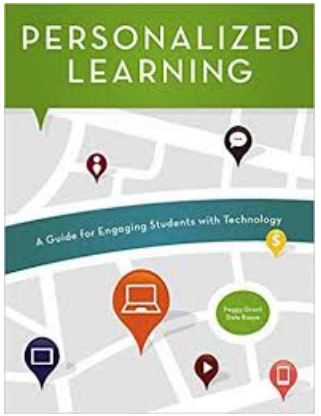
I. Computer History & X-level Parallelism

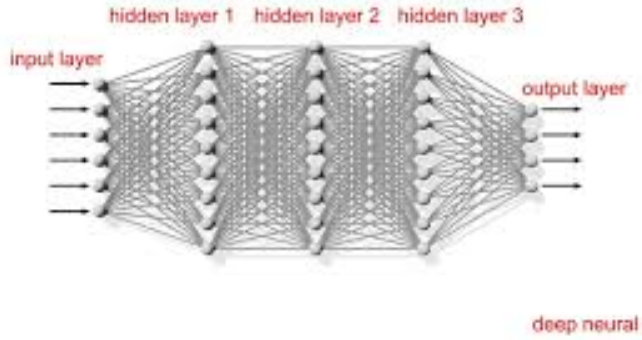
II. Mobile SoCs as ALP Harbinger

III. Gables ALP SoC Model

IV. Call to Action for Accelerator-level Parallelism

Future Apps Demand Much More Computing





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

- **It's the SW stupid!**
- **What SW (model) for a gabled-roof SoC?**

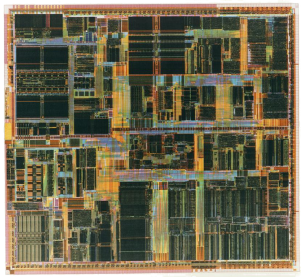
X-level Parallel Hardware + Software (Model)

Single-threaded languages, compilers, runtimes, etc. **hides** parallelism

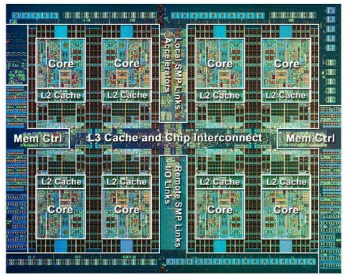
Software **abstracts** TLP, e.g., pThreads, OpenMP, & MPI. Also **hidden** in cloud, etc.

Software **abstracts** TLP+DLP, e.g., CUDA, OpenCL, & graphics OpenGL Also **hidden** in cloud, etc.

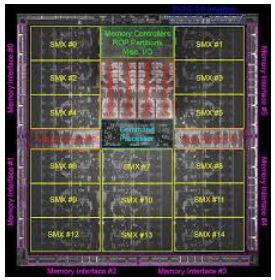
Local software stack **abstracts** each accelerator. **But no good, general software abstraction for SoC ALP!**



Intel Pentium Pro
BLP+ILP



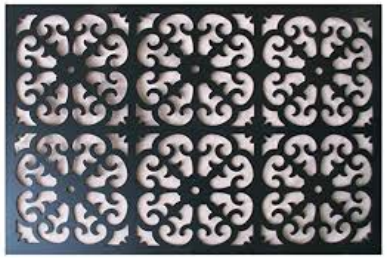
IBM Power 7
BLP+ILP+TLP



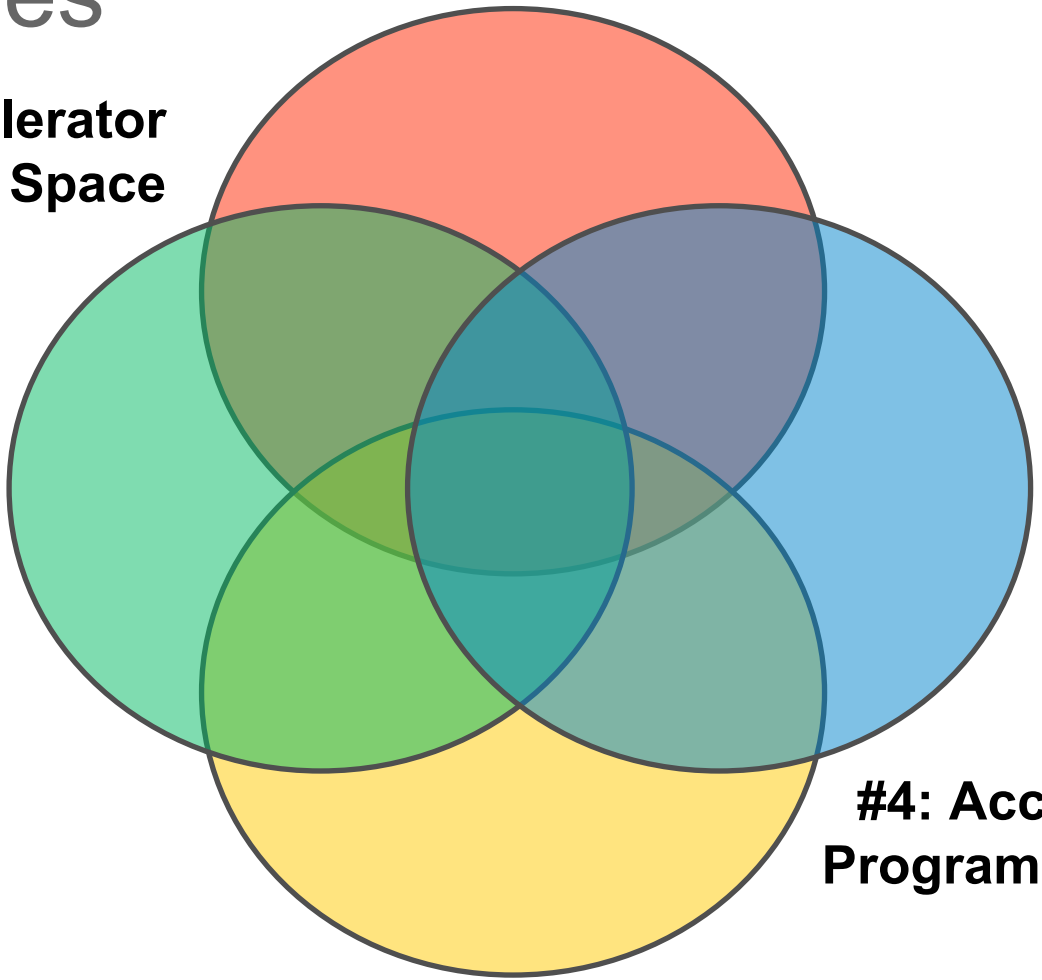
Nvidia GK110
BLP+TLP+DLP

Apple A12
BLP+ILP+TLP+DLP+ALP

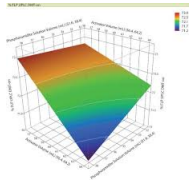
A Parallelism Lattice



Challenges


#1: Accelerator Design Space
#2: Accelerator Concurrency
#3: Accelerator Communication
#4: Accelerator Programmability

#1: Accelerator Design Space



What is the “right” set of accelerators? For HW? For SW?

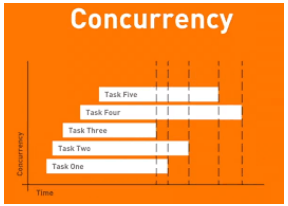
When should “similar” accelerators be combined?

When should accelerators share resources (e.g., SRAM)?

How to future-proof for change (e.g., machine learning)?

How should tools/frameworks speed accelerator design?

#2: Accelerator Concurrency



How to cooperatively schedule accelerators?
By OS or runtime? As devices or processor peers? What HW mechanisms? ***Note: GPU tasks use runtime & HW***

Policies/mechanisms **manage/partition/virtualize** shared resources (compute, cache/memory, interconnect)?

Whither OS/runtime *Hardware Abstraction Layer (HAL)?*
What should HALs hide/expose?

#3: Accelerator Communication

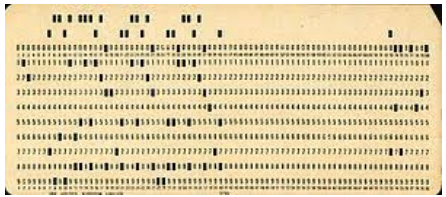
How do accelerators communicate data? Through memory, shared cache, queues, scratchpads? #copies? Abstraction(s)? **Stream dataflow? Idempotent (RDDs)?**

Note: GPU memory: discrete→shared→coherent

How do accelerators communicate control? Through interrupts or polling (both bad) Via CPUs? Other?

Separate or unified drivers?

#4: Accelerator Programmability



Each accelerator has **domain-specific language** (DSL) w/ SDK, JIT, runtime, etc.? Phone ➔ more generally?

Unify multiple accelerator SW "stacks" somehow?

Tools/frameworks to speed SW development?
"**SW is behind HW**" true since 1940s if new SW required

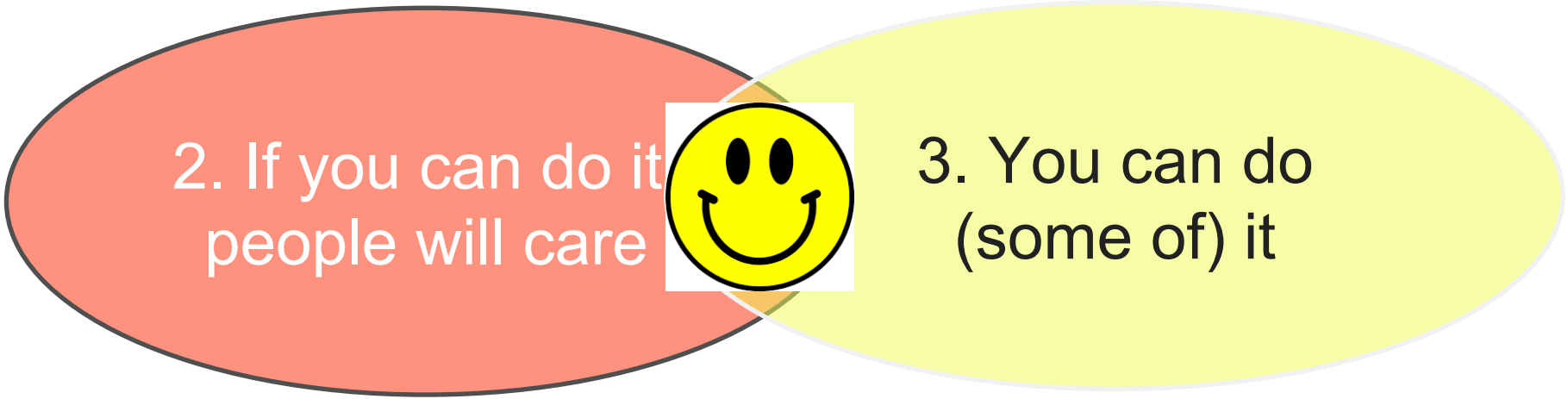
#1-4 needed to delivery ubiquitous ALP to future apps!

Do for ALP what SIMT/runtimes did for GPU TLP+DLP!

Picking Research Problems & ALP

1. Look for **change** – gives fresh opportunity

➔ ALP applications, software, & hardware will explode



➔ True for general ALP Must answer for yourself!

Accelerator-level Parallelism Call to Action

Future apps demand much more computing

Standard tech scaling & architecture NOT sufficient

Mobile SoCs show most/only promising approach:

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

Call to action to develop "science" for ubiquitous ALP

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

Backup Slides

COMPUTING COMMUNITY CONSORTIUM

The **mission** of Computing Research Association's Computing Community Consortium (CCC) is to **catalyze** the computing research community and **enable** the pursuit of innovative, high-impact research.

point successes, lacking SW/HW science

Who

- **Council ~24 members**
- **CCC/CRA Staff**
- **Chair, VC, & Director**

Inputs: Bottom-up, Internal, & Top-Down

What:

- **Workshops & Conf. Blue Sky Tracks**
- **Whitepapers & Social Media**
- **Reports Out (esp. to government)**
- **Biannual Symposium to DCers**

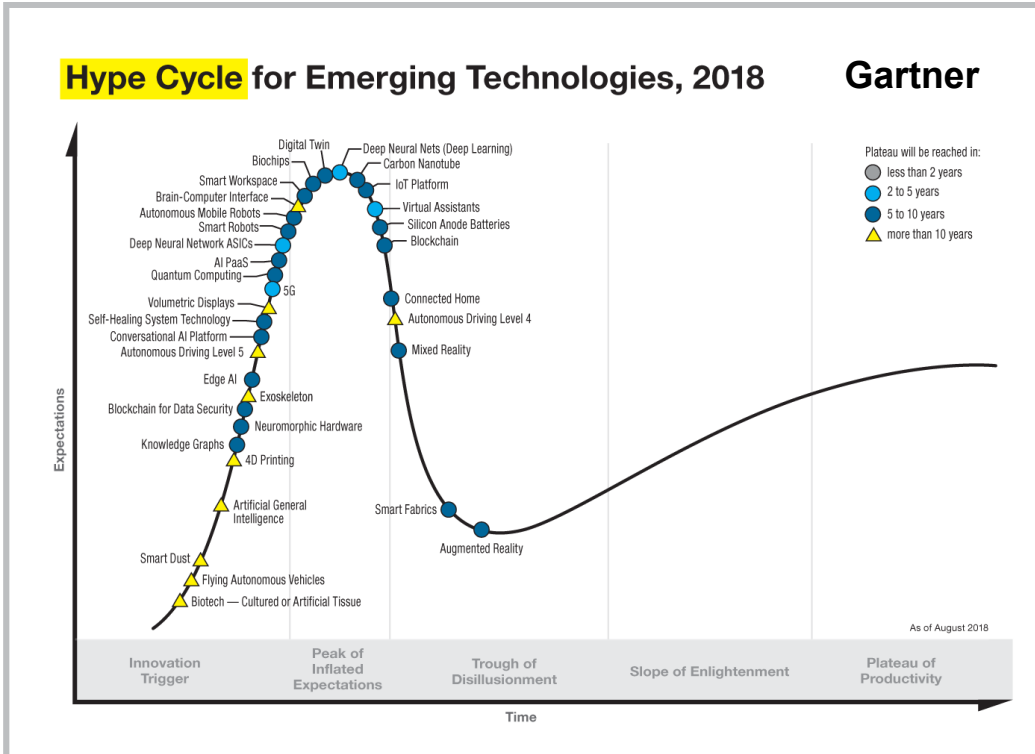
Human Development

- **Early Career Workshops & Participation**
- **Council Membership**
- **Leadership w/ Gov't (LISPI)**

Pitfall X: Design for (Hyped) Importance

IPs should target important workloads, but …

Recommend: Provision IP resources (compute & SRAM) only as needed for important usecases



Gables Math: Roofline / Work Fraction

Roofline: $MIN(B_{peak} * I, P_{peak})$

$MIN(B_{peak} * I, 1 * P_{peak}) \;/\; 1$

$1 / T_{IP[i]} = MIN(B_i * I_i, A_i * P_{peak}) \;/\; f_i \qquad f_i \neq 0$

$1 / T_{memory} = B_{peak} * I_{avg} \qquad I_{avg} = 1 / \Sigma_{i=1,N-1}(f_i / I_i)$

$Perf = MIN(1/T_{IP[0]}, \ldots 1/T_{IP[N-1]}, 1/T_{memory})$