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!
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
20\textsuperscript{th} 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

Danowitz et al., CACM 04/2012
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

$P$

$M$

bus

i/f

dev
Bit-level Parallelism (BLP)

Early computers: few switches (transistors)
- 
  \(\Rightarrow\) compute a result in many steps
- E.g., 1 multiplication partial product per cycle

Bit-level parallelism
- More transistors \(\Rightarrow\) compute more in parallel
- E.g., Wallace Tree multiplier (right)

Larger words help: 8b \(\rightarrow\) 16b \(\rightarrow\) 32b \(\rightarrow\) 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→)

- add
- load

Actually do instructions in parallel → ILP

- add
- load
- branch
- and
- store

Predict direction: target or fall thru
Speculate!
Speculate more!

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**
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)

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

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
BLP+ILP Bit/Instrn-Level Parallelism
+ TLP Thread-Level Parallelism
+ DLP Data-Level Parallelism

Multicore + Integrated GPU
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
+ TLP
Thread-Level Parallelism

Integrated GPU
+ 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

<table>
<thead>
<tr>
<th>Usecases (rows)</th>
<th>CPUs (AP)</th>
<th>Display</th>
<th>Media Scaler</th>
<th>GPU</th>
<th>Image Signal Proc.</th>
<th>JPEG</th>
<th>Pixel Visual Core</th>
<th>Video Decoder</th>
<th>Video Encoder</th>
<th>Dozens More</th>
</tr>
</thead>
<tbody>
<tr>
<td>Photo Enhancing</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video Capture</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Video Capture HDR</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Video Playback</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Recognition</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
Envision use cases (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
Multicore HW
- $P_{\text{peak}} = \text{peak perf of all cores}$
- $B_{\text{peak}} = \text{peak off-chip bandwidth}$

Multicore SW
- $I = \text{operational intensity} = \#\text{operations}/\#\text{off-chip-bytes}$
- E.g., 2 ops / 16 bytes $\rightarrow I = 1/8$

Output $P_{\text{att}} = \text{upper bound on performance attainable}$
Compute v. Communication: Op. Intensity ($I$) = \#operations / \#off-chip bytes
ALP System on Chip (SoC) Model: **NEW Gables**

Gables uses Roofline per IP to provide first answer!

- **SW**: performance model of a “gabled roof?”
- **HW**: select & size accelerators
Use case at each IP[i]

- Operational intensity $l_i$ operations/byte
- Non-negative work $f_i$ (f_i’s sum to 1) w/ IPs in parallel
Example Balanced Design Start w/ Gables

\[ P_{\text{peak}} = 40 \]

\[ A_1 \cdot P_{\text{peak}} = 5 \cdot 40 = 200 \]

Workload (Usecase):

\[ f_0 = 1 \quad \& \quad f_1 = 0 \]

\[ I_0 = 8 = \text{good caching} \]

\[ I_1 = 0.1 = \text{latency tolerant} \]

Performance?
Perf limited by IP[0] at $I_0 = 8$
IP[1] not used $\rightarrow$ no roofline
Let’s Assign IP[1] work: $f_1 = 0 \rightarrow 0.75$

$P_{\text{peak}} = 40$
$B_{\text{peak}} = 10$
$A_1 = 5$
$B_0 = 6$
$B_1 = 15$

$f_1 = 0$
$I_0 = 8$
$I_1 = 0.1$

$P_{\text{att}} = 40 \text{ Gops/s}$
IP[1] present but Perf drops to 1! Why?

$I_1 = 0.1 \rightarrow$ memory bottleneck

Enhance $B_{\text{peak}} = 10 \rightarrow 30$
(at a cost)
Perf only 2 with IP[1] bottleneck

IP[1] SRAM/reuse $I_1 = 0.1 \rightarrow 8$
Reduce overkill $B_{\text{peak}} = 30 \rightarrow 20$

$\begin{align*}
P \text{att} &= 2 \text{ Gops/s} \\
A_1 &= 5 \\
B_0 &= 6 \\
B_1 &= 15 \\
f_1 &= 0.75 \\
I_0 &= 8 \\
I_1 &= 0.1 \\
P_{\text{peak}} &= 40 \\
B_{\text{peak}} &= 30
\end{align*}$
Perf = 160 < A*\(P_{\text{peak}}\) = 200

Can you do better?
It’s possible!

Gables has K+1 rooflines

\(P_{\text{att}} = 160 \text{ Gops/s}\)
Approach: Combine Analytical and Simulation Models

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

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

Cycle-level Simulation
+ Full cycle-accuracy
- Long turn-around time

Refine
validate back-annotate

validate back-annotate

Into Synopsys design flow << 6 months of publication!

Gables: A Roofline Model for Mobile SoCs,
Mark D. Hill and Vijay Janapa Reddi, HPCA, 2019
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:** \( \text{# outstanding msgs} = \text{avg latency} \times \text{avg BW} \)
   • Solution: Add buffers; actual performance → Gables

2. More complex interaction among IP blocks
   • **Root:** Use case 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
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!

No visible parallelism

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

Thought bridge: Must divide work heterogeneously

Accelerate each differently with unique HLLs (DSLs) & SDKs

Local SW stack abstracts each accelerator.

But no good, general SW abstraction for SoC ALP!

Uniprocessor

Homogeneous Multicore

Heterogeneous Multicore

Heterogeneous Accelerators

Today: Device Accelerators

Key: P == processor core; A-E == accelerators
SW+HW Lessons from GP-GPUs?

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Programming</td>
<td>Graphics OpenGL</td>
</tr>
<tr>
<td>2. Concurrency</td>
<td>Either CPU or GPU only; Intra-GPU mechanisms</td>
</tr>
<tr>
<td>3. Communication</td>
<td>Copy data between host &amp; device memories</td>
</tr>
<tr>
<td>4. Design</td>
<td>Driven by graphics only; GP: $0B market</td>
</tr>
</tbody>
</table>
**SW+HW Directions for ALP?**

Need programmability for broad success!!!! In less than four decades?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Programming</td>
<td>Local: Per-IP DSL &amp; SDK Global: Ad hoc</td>
</tr>
<tr>
<td>2. Concurrency</td>
<td>Ad hoc</td>
</tr>
<tr>
<td>3. Communication</td>
<td>SW: Up/down OS stack HW: Via off-chip memory</td>
</tr>
<tr>
<td>4. Design, e.g., select, combine, &amp; size IPs</td>
<td>Ad hoc</td>
</tr>
</tbody>
</table>

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

1. Programmability
   Whither global model/runtime?
   DAG of streams for SoCs?

2. Concurrency
   HW assist for scheduling?
   Virtualize & partition?

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

4. Design Space
   When combine "similar" accelerators?
   Power vs. area?

Hennessy & Patterson: A New Golden Age for Computer Architecture
New Feb 2020!