Faster Computing

Better Algorithms

More Data

Value
We need to continue delivering improved performance and perf/W
But Process Technology isn’t Helping us Anymore

Moore’s Law is Dead
Accelerators can continue scaling perf and perf/W
Fast Accelerators since 1985


- **Darwin**: Turakhia, Bejerano, and Dally, “Darwin: A Genomics Co-processor provides up to 15,000 × acceleration on long read assembly”, ASPLOS 2018.

- **SATiN**: Zhuo, Rucker, Wang, and Dally, “Hardware for Boolean Satisfiability Inference,” Under Review.
Accelerators Employ:

- Massive **Parallelism** – >1,000x, not 16x – with **Locality**

- Special **Data Types and Operations**
  - Do in 1 cycle what normally takes 10s or 100s

- Optimized **Memory**
  - High bandwidth (and low energy) for specific data structures and operations

- Reduced or Amortized **Overhead**

- Algorithm-Architecture **Co-Design**
Specialized Hardware is Everywhere

- Does most of the work
- But is mostly invisible

Cell phones
- Software on ARM cores
  - High-complexity, low-compute work
- Accelerators do the heavy lifting
  - MODEMs
  - CODECs
  - Camera Image Processing
  - DNNs
  - Graphics

GPUs
- Rasterizer
- Texture filter
- Compositing
- Compression/Decompression
- Tensor computations
- BVH Traversal
- Ray-Triangle Intersection
Specialized Operations
Orders of Magnitude Efficiency
Moderate Speedup
Specialized Operations

Dynamic programming for gene sequence alignment (Smith-Waterman)

$$I(i, j) = \max \{H(i, j - 1) - o, I(i, j - 1) - e\}$$

$$D(i, j) = \max \{H(i - 1, j) - o, D(i - 1, j) - e\}$$

$$H(i, j) = \max \left\{ \begin{array}{c}
0 \\
I(i, j) \\
D(i, j) \\
H(i - 1, j - 1) + W(r_i, q_j)
\end{array} \right\}$$

On 14nm CPU
35 ALU ops, 15 load/store
37 cycles
81nJ

On 40nm Special Unit
1 cycle (37x speedup)
3.1pJ (26,000x efficiency)
300fJ for logic (remainder is memory)
Why is a Specialized PE 26,000x More Efficient?

Area is proportional to energy – all 28nm

16b Int Add, 32fJ


Specialization -> Efficiency
Efficiency -> Parallelization
Parallelization -> Speedup
Specialized Operations

\[
I(i, j) = \max \{ H(i, j - 1) - o, I(i, j - 1) - e \}
\]

\[
D(i, j) = \max \{ H(i - 1, j) - o, D(i - 1, j) - e \}
\]

\[
H(i, j) = \max \begin{cases} 
0 \\
I(i, j) \\
D(i, j) \\
H(i - 1, j - 1) + W(r_i, q_j)
\end{cases}
\]

Dynamic programming for gene sequence alignment (Smith-Waterman)

Specialization -> 37x speedup, 26,000x efficiency, 270,000x for logic

Efficiency -> Parallelism 64 PE arrays x 64 PEs per array, 4,096x total

Speedup = 37 (Specialization) \times 4,034 (Parallelism) = 150,000x total
Specialized Operations

\[
I(i, j) = \max \{H(i, j - 1) - o, I(i, j - 1) - e\}
\]

\[
D(i, j) = \max \{H(i - 1, j) - o, D(i - 1, j) - e\}
\]

\[
H(i, j) = \max \begin{cases} 
0 \\
I(i, j) \\
D(i, j) \\
H(i - 1, j - 1) + W(r_i, q_j) 
\end{cases}
\]

Dynamic programming for gene sequence alignment (Smith-Waterman)

Specialization -> 37x speedup, 26,000x efficiency, 270,000x for logic

Efficiency -> Parallelism 64 PE arrays x 64 PEs per array, 4,096x total

Speedup = 37 (Specialization) x 4,034 (Parallelism) = 150,000x total
Accelerator Design is Guided by Cost

Arithmetic is Free
(particularly low-precision)

Memory is expensive

Communication is prohibitively expensive
## Need to Understand Cost of Operations And Communication

### Relative Energy Cost

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b Add</td>
<td>0.03</td>
</tr>
<tr>
<td>16b Add</td>
<td>0.05</td>
</tr>
<tr>
<td>32b Add</td>
<td>0.1</td>
</tr>
<tr>
<td>16b FP Add</td>
<td>0.4</td>
</tr>
<tr>
<td>32b FP Add</td>
<td>0.9</td>
</tr>
<tr>
<td>8b Mult</td>
<td>0.2</td>
</tr>
<tr>
<td>32b Mult</td>
<td>3.1</td>
</tr>
<tr>
<td>16b FP Mult</td>
<td>1.1</td>
</tr>
<tr>
<td>32b FP Mult</td>
<td>3.7</td>
</tr>
<tr>
<td>32b SRAM Read (8KB)</td>
<td>5</td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>640</td>
</tr>
</tbody>
</table>

### Relative Area Cost

<table>
<thead>
<tr>
<th>Area (µm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>137</td>
</tr>
<tr>
<td>1360</td>
</tr>
<tr>
<td>4184</td>
</tr>
<tr>
<td>282</td>
</tr>
<tr>
<td>3495</td>
</tr>
<tr>
<td>1640</td>
</tr>
<tr>
<td>7700</td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

Energy numbers are from Mark Horowitz “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014. Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.
Communication is Expensive, Be Small, Be Local

LPDDR DRAM
GB

640pJ/word

On-Chip SRAM
MB

50pJ/word

Local SRAM
KB

5pJ/word
Scaling of Communication

The Algorithm Often Has to Change To Avoid Being Global Memory Limited
Algorithm-Architecture Co-Design for Darwin
Start with Graphmap

1. Graphmap (software)

Graphmap

~10K seeds
~440M hits

**Filtration**

~3 hits

**Alignment**

~1 hits

**Time/Read (ms)**

- Filtration
- Alignment

<table>
<thead>
<tr>
<th>0.1</th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm-Architecture Co-Design for Darwin
Replace Graphmap with Hardware-Friendly Algorithms
Speed up Filtering by 100x, but 2.1x Slowdown Overall

Graphmap
- ~10K seeds
- ~440M hits

Darwin
- ~2K seeds
- ~1M hits

Time/read (ms)

1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
Algorithm-Hardware Co-Design for Darwin Accelerate Alignment – 380x Speedup

1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration

<table>
<thead>
<tr>
<th>Time/read (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 1 10 100 1000 10000 100000</td>
</tr>
</tbody>
</table>

- Filtration
- Alignment

1. 2.1X slowdown
2. 380X speedup
Algorithm-Hardware Co-Design for Darwin
4x Memory Parallelism – 3.9x Speedup

1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration
4. Four DRAM channels for D-SOFT
Algorithm-Hardware Co-Design for Darwin
Specialized Memory for D-Soft Bin Updates – 15.6x Speedup

1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration
4. Four DRAM channels for D-SOFT
5. Move bin updates in D-SOFT to SRAM (ASIC)
Algorithm-Hardware Co-Design for Darwin
Pipeline D-Soft and GACT – now completely D-Soft limited – 1.4x
Overall 15,000x

1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration
4. Four DRAM channels for D-SOFT
5. Move bin updates in D-SOFT to SRAM (ASIC)
6. Pipeline D-SOFT and GACT
Memory Dominates
Memory dominates power and area
<table>
<thead>
<tr>
<th></th>
<th>Unit</th>
<th>Area (mm²)</th>
<th>(%)</th>
<th>Power (W)</th>
<th>(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GACT</td>
<td>Logic</td>
<td>17.6</td>
<td>20.5%</td>
<td>1.04</td>
<td>23.6%</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>68.0</td>
<td>79.5%</td>
<td>3.36</td>
<td>76.4%</td>
</tr>
<tr>
<td>D-SOFT</td>
<td>Logic</td>
<td>6.2</td>
<td>1.8%</td>
<td>0.41</td>
<td>4.4%</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>320.3</td>
<td>98.2%</td>
<td>8.80</td>
<td>95.6%</td>
</tr>
<tr>
<td>EIE</td>
<td>Logic</td>
<td>2.8</td>
<td>6.9%</td>
<td>0.23</td>
<td>40.3%</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>38.0</td>
<td>93.1%</td>
<td>0.34</td>
<td>59.7%</td>
</tr>
</tbody>
</table>
Algorithms must be memory optimized
Minimize global memory accesses
Keep local memory footprint small
GACT Alignment

- 15M Reads, 10k bases each, ~2k hits each
  - ~300T Alignments to be done
  - Additional parallelism within each alignment
- But long reads have large (10M) memory footprint
- Solution: GACT (Tiling)
GACT Alignment

- 15M Reads, 10k bases each, ~2k hits each
  - ~300T Alignments to be done
  - Additional parallelism within each alignment
- But long reads have large (10M) memory footprint
- Solution: GACT (Tiling)

Darwin GACT hardware
4k PEs - 64 PEs per Array x 64 Arrays
~50 operations per cycle per PE
200k operations per cycle
Specialized memory
150,000x speedup vs CPU
On-Chip Memory
Cost per Bit is 10-100x Commodity DRAM
And It’s Often Less Expensive
D-SOFT: Algorithm Overview

Slope=1

Bin 1 Bin 2 Bin 3 Bin 4 Bin 5

<table>
<thead>
<tr>
<th>Bin count (bases)</th>
<th>Last hit offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-inf</td>
</tr>
<tr>
<td>0</td>
<td>-inf</td>
</tr>
<tr>
<td>0</td>
<td>-inf</td>
</tr>
<tr>
<td>0</td>
<td>-inf</td>
</tr>
<tr>
<td>0</td>
<td>-inf</td>
</tr>
<tr>
<td>0</td>
<td>-inf</td>
</tr>
</tbody>
</table>
D-SOFT: Algorithm Overview
D-SOFT: Algorithm Overview

- Pointer Table:
  - GA: 17
  - GC: 21
  - GG: 21

- Position Table:
  - 17: 1
  - 18: 15
  - 19: 18
  - 20: 38

- Bin count (bases) Last hit offset
  - 0: -inf
  - 4: 2
  - 2: 2
  - 2: 0
  - 2: 2
D-SOFT: Algorithm Overview

- **GTGCTGGATATA**
- **AGCTTTCCCTACGATAGCTGCATCTATTCTCGTATTTAGC**
- **Pointer Table**
  - CG: 12
  - CT: 17
  - GA: 17
- **Position Table**
  - 12: 2
  - 13: 8
  - 14: 16
  - 15: 22
  - 16: 28
  - 17: 1

- **Bin count (bases)**
  - 2: 3
  - 5: 3
  - 3: 3
  - 4: 3
  - 2: 2

- **Last hit offset**
  - 3
  - 3
  - 3
  - 3
  - 2
D-SOFT: Algorithm Overview

![Diagram of D-SOFT algorithm showing pointer and position tables with bin counts and last hit offsets.]

- **Pointer Table**
  - TC: 32
  - TG: 33
  - TT: 39

- **Position Table**
  - Bin counts (bases) and Last hit offset:
    - 2: 3
    - 5: 3
    - 4: 4
    - 5: 4
    - 2: 2
D-SOFT: Algorithm Overview

- AGCTTTCCCTACGTAGC
- TG CATCTATTTCTCGTATTTAGC
- GTGCT

32
33
39

- Bin count (bases)
- Last hit offset

<table>
<thead>
<tr>
<th>Bin count (bases)</th>
<th>Last hit offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
D-SOFT: Algorithm Overview

Parameters:

**k**: seed size

**N**: number of seeds

**h**: threshold on non-overlapping bases

**B**: bin size (number of bases, fixed to 128)

(k=2, N=6, h=6)
D-SOFT: Hardware-acceleration

Network-on-chip (16-endpoint Butterfly)

- Bin-count SRAM 1
- Update-bin logic (UBL)
- NZ bins SRAM
- Bin-count SRAM 16
- Update-bin logic (UBL)
- NZ bins SRAM

Arbiter

DRAM

Seed-position lookup (SPL)

(seed, j)

(candidate_pos)

GTGCTTGGATATA

AGCTTTCCCTACGTAGCTGCATCTATTTCTCGTATTTAGC
Cost has a Time Component

\[ C = T(B_1N_1 + B_2N_2 + \ldots + P) \]

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>B_1</th>
<th>N_1</th>
<th>B_2</th>
<th>N_2</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Darwin Filter</td>
<td>1</td>
<td>100</td>
<td>64M</td>
<td>1</td>
<td>128G</td>
<td>134G</td>
</tr>
<tr>
<td>All DRAM</td>
<td>15.6</td>
<td></td>
<td></td>
<td>1</td>
<td>128G</td>
<td>1,997G</td>
</tr>
</tbody>
</table>
Hardware Enables Irregular, Compressed Data Structures (reduces memory footprint)
Dynamic Sparse Activations, Static Sparse Weights

\[
\tilde{\alpha} \begin{pmatrix} a_0 & a_1 & 0 & a_3 \end{pmatrix} \times PE \begin{pmatrix} w_{0,0}, 0, 0, 0, 0, w_{0,3} \\ 0, 0, w_{1,2}, 0, 0, 0 \\ 0, w_{2,1}, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0 \\ w_{5,0}, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, w_{6,3} \\ 0, w_{7,1}, 0, 0, 0, 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} \text{ReLU} \\ \text{ReLU} \\ \text{ReLU} \end{pmatrix} \Rightarrow \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \end{pmatrix}
\]

<table>
<thead>
<tr>
<th>Virtual Weight</th>
<th>(W_{0,0})</th>
<th>(W_{0,1})</th>
<th>(W_{4,2})</th>
<th>(W_{0,3})</th>
<th>(W_{4,3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Column Pointer</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Trained Quantization

Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, arXiv 2015
EIE Hardware

Central Control

SpMat

PTR Even
Arithm
PTR Odd

SpMat

Act Queue

Even Ptr SRAM

Odd Ptr SRAM Bank

Col Start/End Addr

Sparse Matrix Access

Weight Decoder

Address Accum

Absolute Address

Arithmetic Unit

Act R/W

Act SRAM

Act Value

Encoded Weight

Col

Dest Act Regs

Src Act Regs

Leading NZero

ReLU

Act Index

Pointer Read
Simple Parallelism Often Beats a “Better” Algorithm
Broadcast Literals to Associative Memory of Clauses
More Comparisons but Lower Latency
Need to Accelerate the Whole Problem
Implication 300-500x Faster than CPU

Number of Cycles per Boolean Constraint Propagation

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>CPU Cycles</th>
<th>Accelerator Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>bench_7666.smt2.cnf</td>
<td>2105</td>
<td>0.58</td>
</tr>
<tr>
<td>bench_7222.smt2.cnf</td>
<td>1180</td>
<td>0.60</td>
</tr>
<tr>
<td>bench_8061.smt2.cnf</td>
<td>1357</td>
<td>0.66</td>
</tr>
<tr>
<td>bench_13535.smt2.cnf</td>
<td>923</td>
<td>0.35</td>
</tr>
<tr>
<td>qg01-08.cnf</td>
<td>1993</td>
<td>0.26</td>
</tr>
<tr>
<td>SAT_instance_N=33.cnf</td>
<td>775</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Need to Accelerate the **Whole** Problem

---

**Minisat Statistical Profile**

15 random APP16 benchmarks running >90s

- **propagate**
- **analyze**
- **litRedundant**
- **pickBranchLit**
- **cancelUntil**
- **reduceDB**
Platforms for Acceleration
Implementation Alternatives

![Bar chart comparing performance metrics for Deep Learning and Genomics across different technologies: CPU, FPGA, GPU, ASIC. Metrics include Images/s-Watt and Mcells/s-Watt.](chart.png)
GPUs Provide:

- High-Bandwidth, Hierarchical **Memory** System
  - Can be configured to match application

- Programmable **Control** and **Operand Delivery**

- Simple places to bolt on **Domain-Specific Hardware**
  - As instructions or memory clients
Volta V100

21B xtors | TSMC 12nm FFN | 815mm²
5,120 CUDA cores
7.8 FP64 TFLOPS | 15.7 FP32 TFLOPS
125 Tensor TFLOPS
20MB SM RF | 16MB Cache
32GB HBM2 @ 900 GB/s
300 GB/s NVLink
Tensor Core

\[
D = \begin{pmatrix}
A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\
A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3}
\end{pmatrix}_{\text{FP16}}
\ \cdot\ 
\begin{pmatrix}
B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\
B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\
B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\
B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3}
\end{pmatrix}_{\text{FP16}}
\ +
\begin{pmatrix}
C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\
C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\
C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\
C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3}
\end{pmatrix}_{\text{FP16 or FP32}}
\]

\[
D = AB + C
\]
### Specialized Instructions Amortize Overhead

*Overhead is instruction fetch, decode, and operand fetch – 30pJ

**Energy numbers from 45nm process

<table>
<thead>
<tr>
<th>Operation</th>
<th>Ops</th>
<th>Energy**</th>
<th>Overhead*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vs op</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>%tot</td>
</tr>
<tr>
<td>HFMA</td>
<td>2</td>
<td>1.5pJ</td>
<td>20x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>95%</td>
</tr>
<tr>
<td>HDP4A</td>
<td>8</td>
<td>6.0pJ</td>
<td>5x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>83%</td>
</tr>
<tr>
<td>HMMA</td>
<td>128</td>
<td>130pJ</td>
<td>0.23x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19%</td>
</tr>
<tr>
<td>IMMA</td>
<td>1024</td>
<td>230pJ</td>
<td>0.13x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12%</td>
</tr>
</tbody>
</table>

*Overhead is instruction fetch, decode, and operand fetch – 30pJ

**Energy numbers from 45nm process
Program

(map force (pairs particles))

Mapping Directives

Mapper & Runtime

Synthesis

Data & Task Placement

Custom Compute Blocks (Instructions or Clients)

SMs

Configurable Memory

Efficient NoC

GPU
DSA Design is Programming
With a Hardware Cost Model

Algorithm

\[
\begin{align*}
&\text{tb} \leftarrow \text{GACT}(r, q) \\
&\text{input} : r[TS], q[TS] \\
&\text{output} : \text{tb}[TS,TS] \\
&\text{for } i = 0..TS-1 \text{ do} \\
&\hspace{1em} \text{for } j = 0..TS-1 \text{ do} \\
&\hspace{2em} \text{in } (i,j) \leftarrow \text{Max} \ (h(i,j-1) - O, \text{in } (i, j-1) - E) \\
&\hspace{2em} \text{del } (i,j) \leftarrow \text{Max} \ (h(i-1,j) - O, \text{del } (i-1,j) - E) \\
&\hspace{2em} h(i,j) \leftarrow \text{Max} \ (0, \text{in}(i,j), \text{del } (i,j), h(i-1, j-1) + W \ (r[i].q[j])) \\
&\hspace{2em} \text{tb } [i,j] \leftarrow \text{ComputeTb } (h(i,j), \text{in } (i,j), \text{del } (i,j)) \\
&\text{end} \\
&\text{end}
\end{align*}
\]

Mapping

\[
\begin{align*}
&\text{STRIPES } \leftarrow \text{TS / AS} \\
&\text{processor_array } p \ (\text{AS}) \\
&\text{memory_array } \text{tbm } (\text{AS})[\text{STRIPES, TS }] \\
&\text{Map } h \ (i,j) \rightarrow p \ (i \ % \ AS) \\
&\hspace{1em} \text{at } t = (i \ % \ AS) \cdot TS + j - i \ / \ AS \\
&\text{Map } \text{tb } [i,j] \rightarrow \text{tbm } (i \ % \ AS) \ [i \ / \ AS, j]
\end{align*}
\]
Implementation Alternatives

- GDDR6
- LPDDR4
- DPSTEP

Comparative chart showing performance metrics for different technologies in Deep Learning and Genomics.
Multi-Chip Modules (MCMs)
de-novo assembly of noisy long-reads 50x coverage
Conclusion
Summary

• Moore’s Law is over, but we must continue scaling perf/W

• Accelerators are the future
  – Specialization -> Efficiency
  – Parallelism -> Speedup
  – Co-Design: The algorithm has to change

• Memory Dominates:
  – Minimize global memory access
  – Minimize memory footprint – new algorithms, sparsity, compression
  – Lots of small, fast on-chip memories

• GPUs as accelerator platforms
  – GPUs – efficient memory, communication and control
  – Custom blocks – instructions or clients

• DSA design is programming – with a hardware cost model