The Future of Computing: Domain-Specific Accelerators

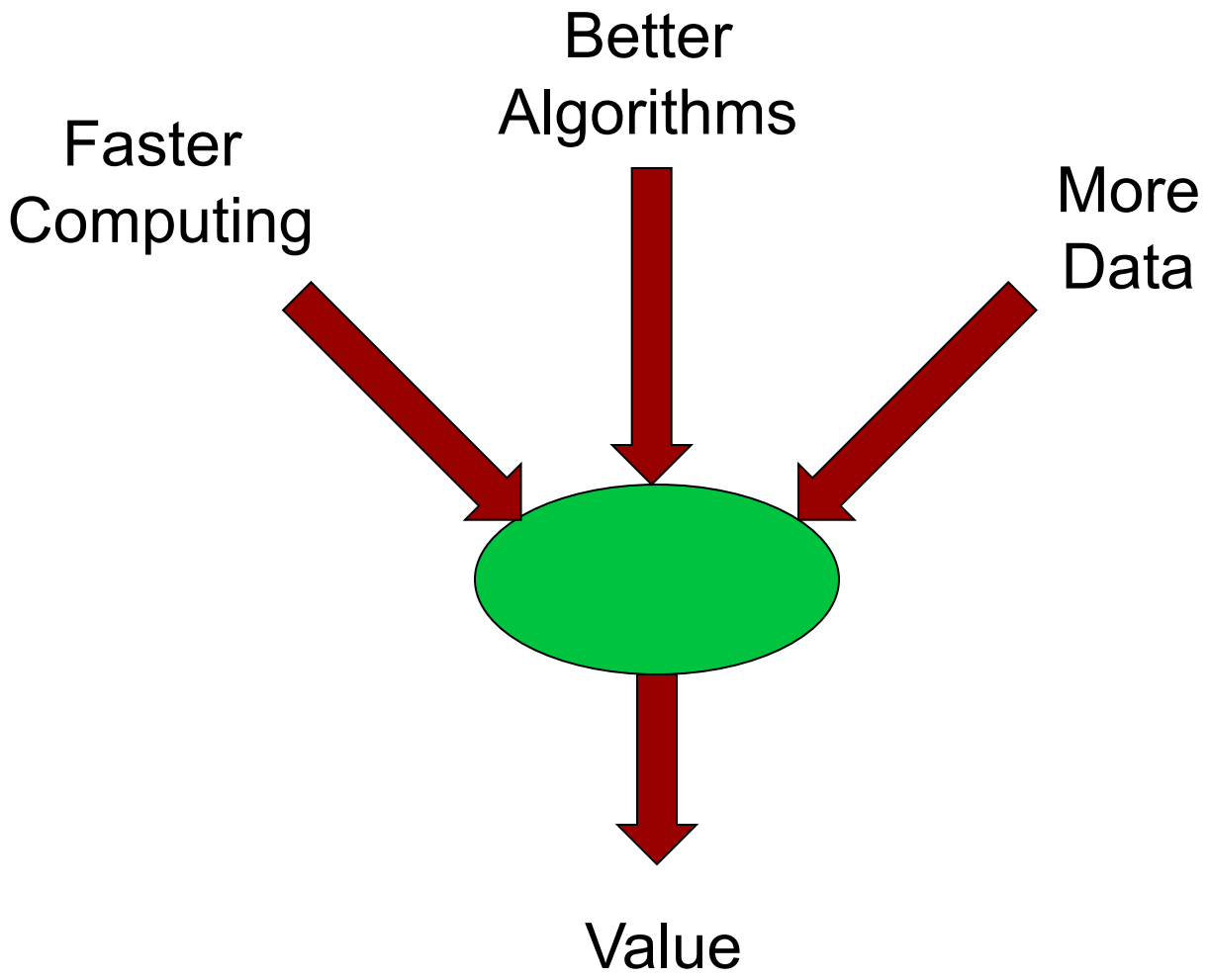
Micro

October 15, 2019

Bill Dally

Chief Scientist and SVP of Research, NVIDIA Corporation

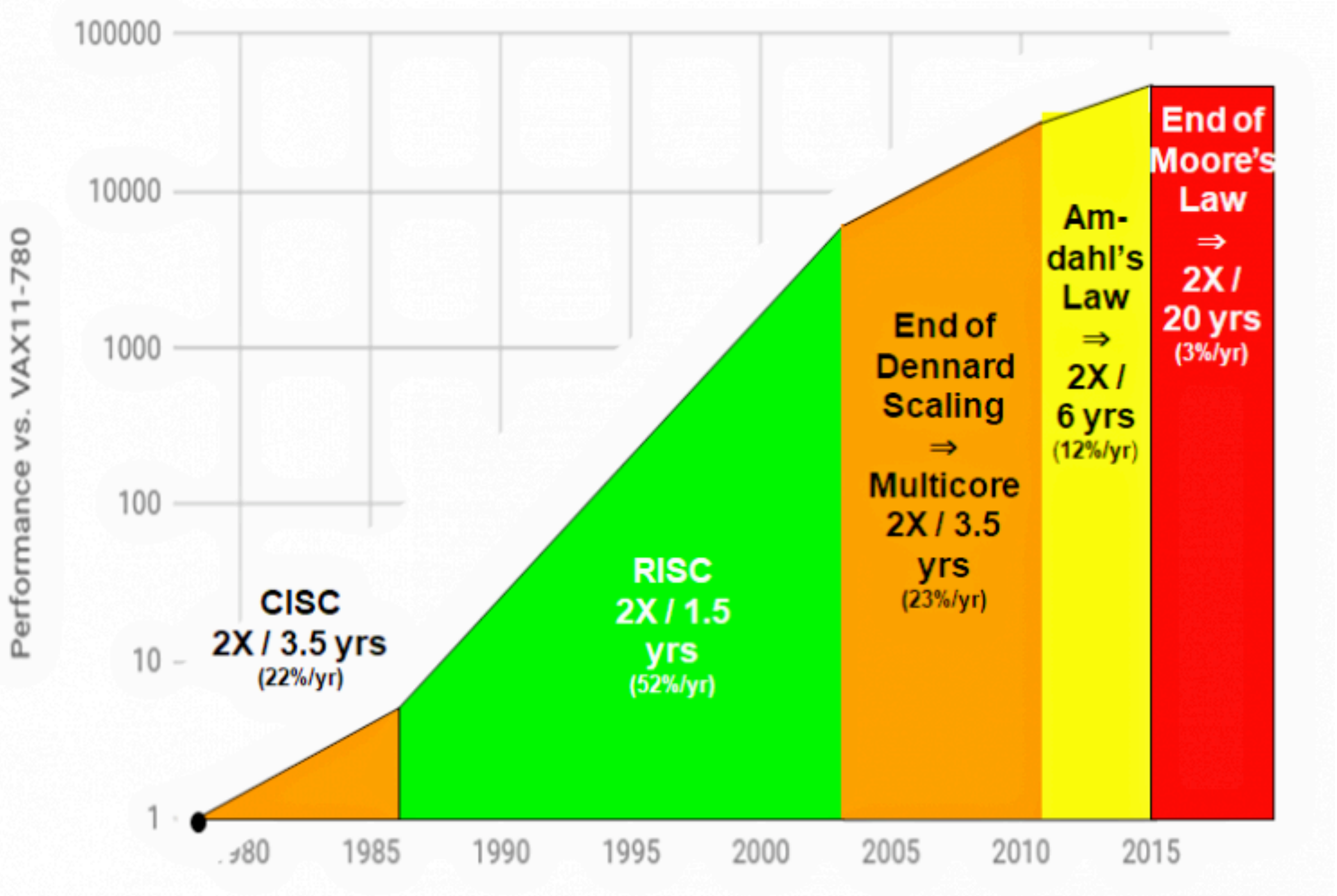
Professor (Research), Stanford University


Better
Algorithms
Faster
Computing
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



John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

Accelerators can continue scaling perf and perf/W

Fast Accelerators since 1985

- **Mossim Simulation Engine**: Dally, W.J. and Bryant, R.E., 1985. A hardware architecture for switch-level simulation. *IEEE Trans. CAD*, *4*(3), pp.239-250.
- **MARS Accelerator**: Agrawal, P. and Dally, W.J., 1990. A hardware logic simulation system. *IEEE Trans. CAD*, *9*(1), pp.19-29.
- **Reconfigurable Arithmetic Processor**: Fiske, S. and Dally, W.J., 1988. *The reconfigurable arithmetic processor* . ISCA 1988.
- **Imagine**: Kapasi, U.J., Rixner, S., Dally, W.J., Khailany, B., Ahn, J.H., Mattson, P. and Owens, J.D., 2003. Programmable stream processors. *Computer*, *36*(8), pp.54-62.
- **ELM**: Dally, W.J., Balfour, J., Black-Shaffer, D., Chen, J., Harting, R.C., Parikh, V., Park, J. and Sheffield, D., 2008. Efficient embedded computing. *Computer*, *41*(7).
- **EIE**: Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A. and Dally, W.J., 2016, June. EIE: efficient inference engine on compressed deep neural network, ISCA 2016
- **SCNN**:Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S.W. and Dally, W.J., 2017, June. Scnn: An accelerator for compressed-sparse convolutional neural networks, ISCA 2017
- **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

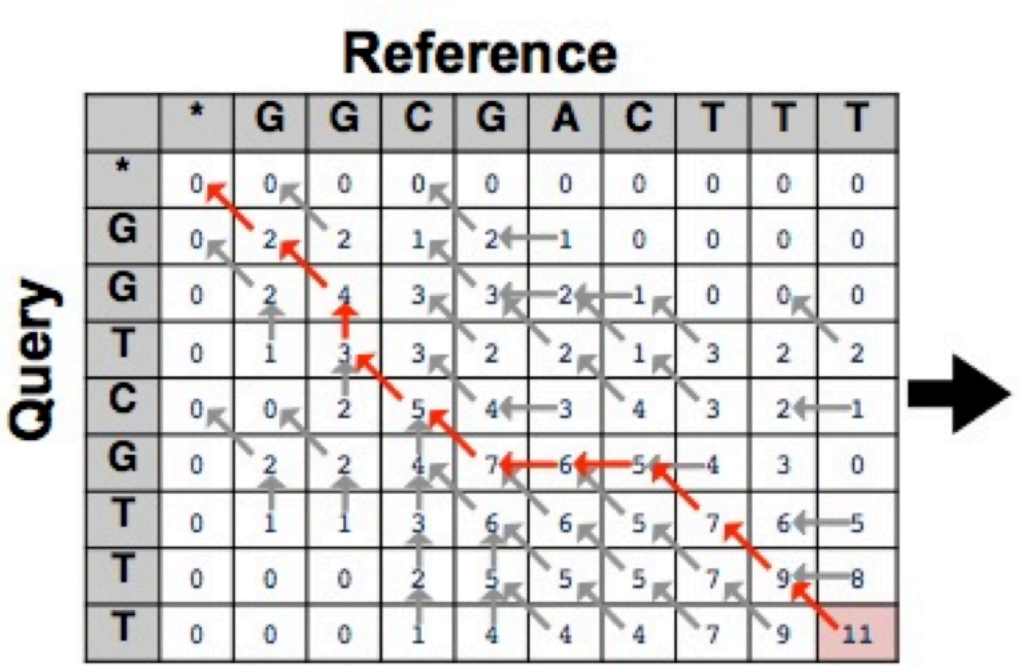
- GPUs
  - Rasterizer
  - Texture filter
  - Compositing
  - Compression/Decompression
  - Tensor computations
  - BVH Traversal
  - Ray-Triangle Intersection

Specialized Operations

Orders of Magnitude Efficiency

Moderate Speedup

Specialized Operations



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

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

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

Dynamic programming for gene sequence alignment (Smith-Waterman)

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

OOO CPU Instruction – 250pJ (99.99% overhead, ARM A-15)

Evangelos Vasilakis. 2015. *An Instruction Level Energy Characterization of Arm Processors.* Foundation of Research and Technology Hellas, Inst. of Computer Science, Tech. Rep. FORTH-ICS/TR-450 (2015)

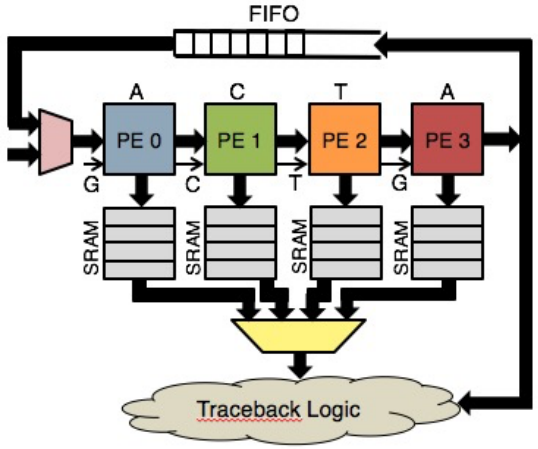
Specialization -> Efficiency

Efficiency -> Parallelization

Parallelization -> Speedup

Specialized Operations

$$
\begin{aligned}
I(i,j) &= \max\{H(i,j-1)-o,\ I(i,j-1)-e\} \\
D(i,j) &= max\{H(i-1,j)-o,\ D(i-1,j)-e\} \\
H(i,j) &= max\begin{cases} 0 \\ I(i,j) \\ D(i,j) \\ H(i-1,j-1)+W(r_i,q_j) \end{cases}
\end{aligned}
$$



Dynamic programming for gene sequence alignment (Smith-Waterman)

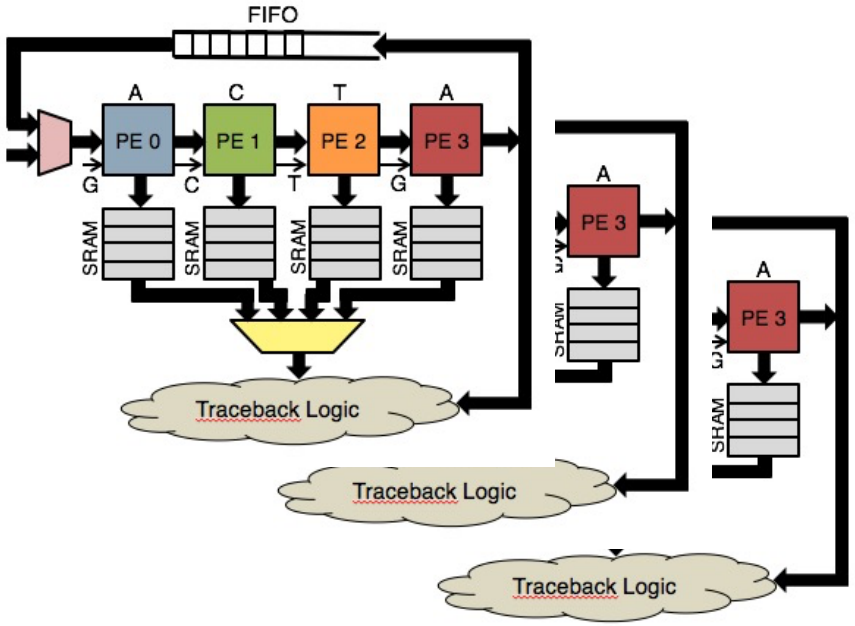
Specialization -> 37x speedup, 26,000x efficiency, 270,000x for logic

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

Speedup = 37 (Specialization) x 4,034 (Parallelism) = 150,000x total

Specialized Operations

$$
\begin{aligned}
I(i,j) &= \max\{H(i,j-1)-o,\ I(i,j-1)-e\} \\
D(i,j) &= max\{H(i-1,j)-o,\ D(i-1,j)-e\} \\
H(i,j) &= max\begin{cases} 0 \\ I(i,j) \\ D(i,j) \\ H(i-1,j-1)+W(r_i,q_j) \end{cases}
\end{aligned}
$$



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

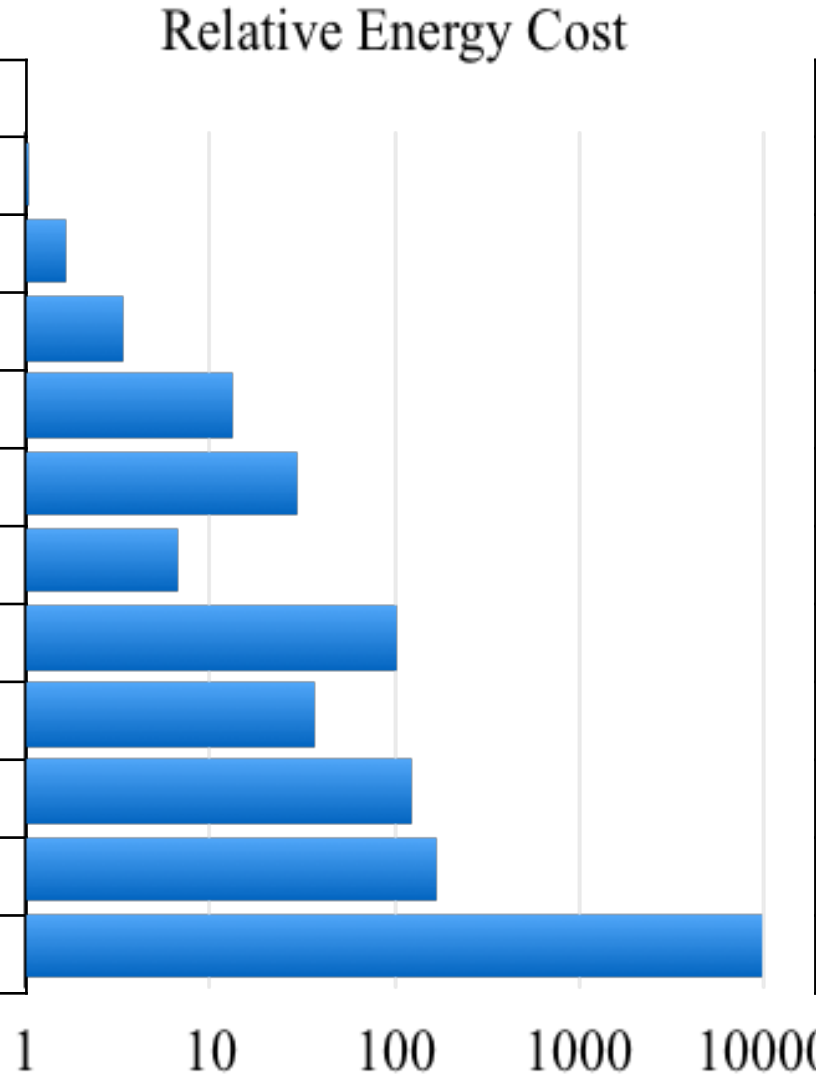
Arithmetic is Free
(particularly low-precision)

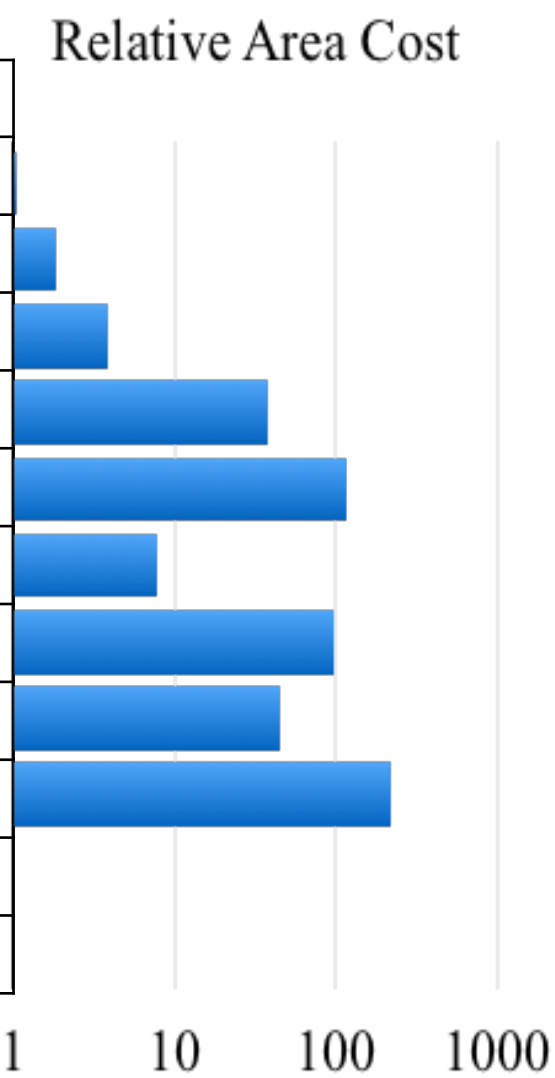
Memory is expensive

Communication is prohibitively expensive

Need to Understand Cost of Operations And Communication

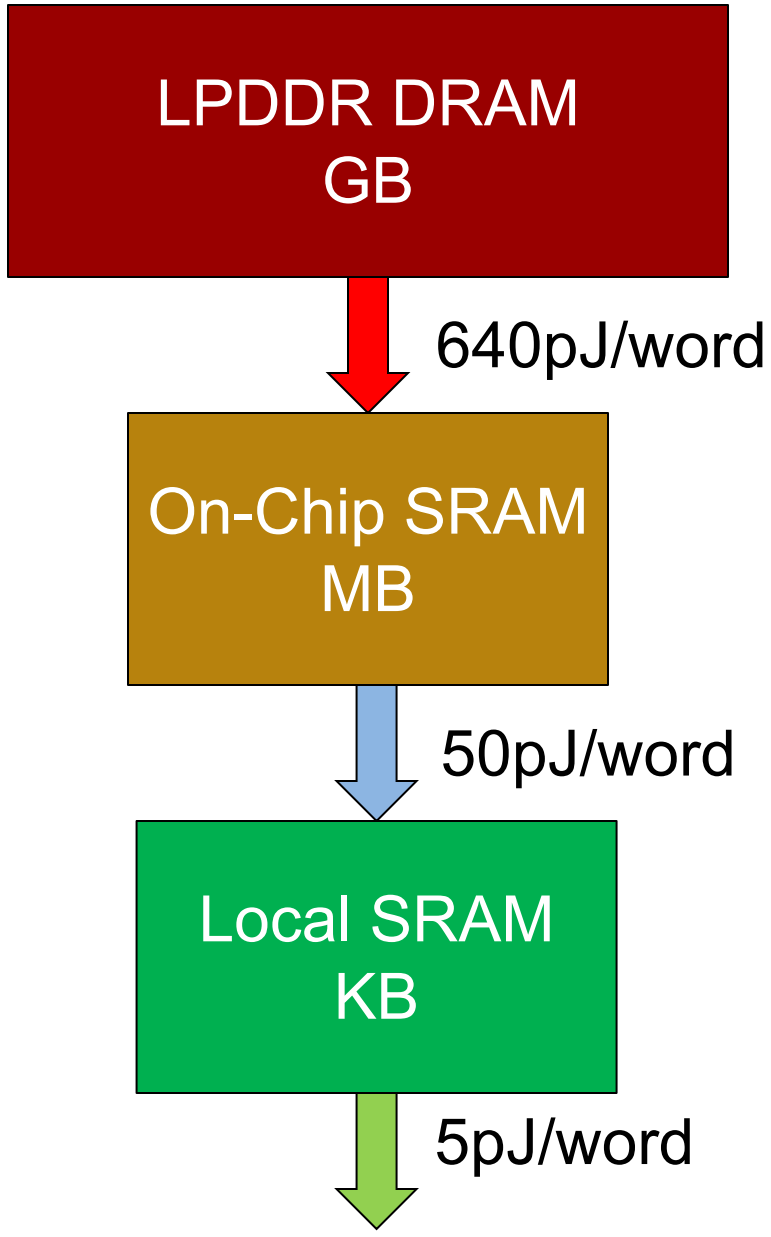
Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A



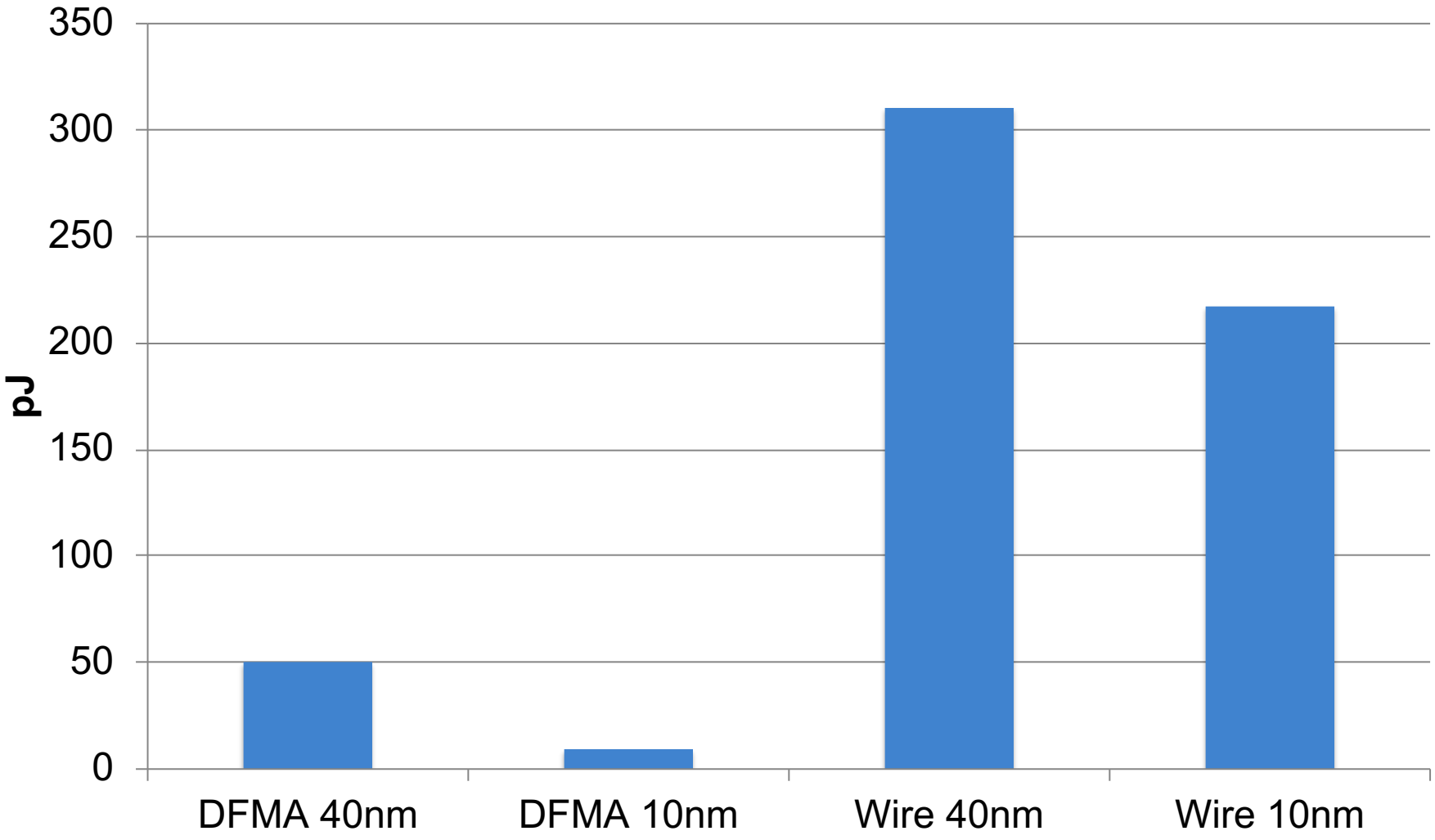


Energy numbers are from Mark Horowitz “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014
Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

Communication is Expensive, Be Small, Be Local



Scaling of Communication

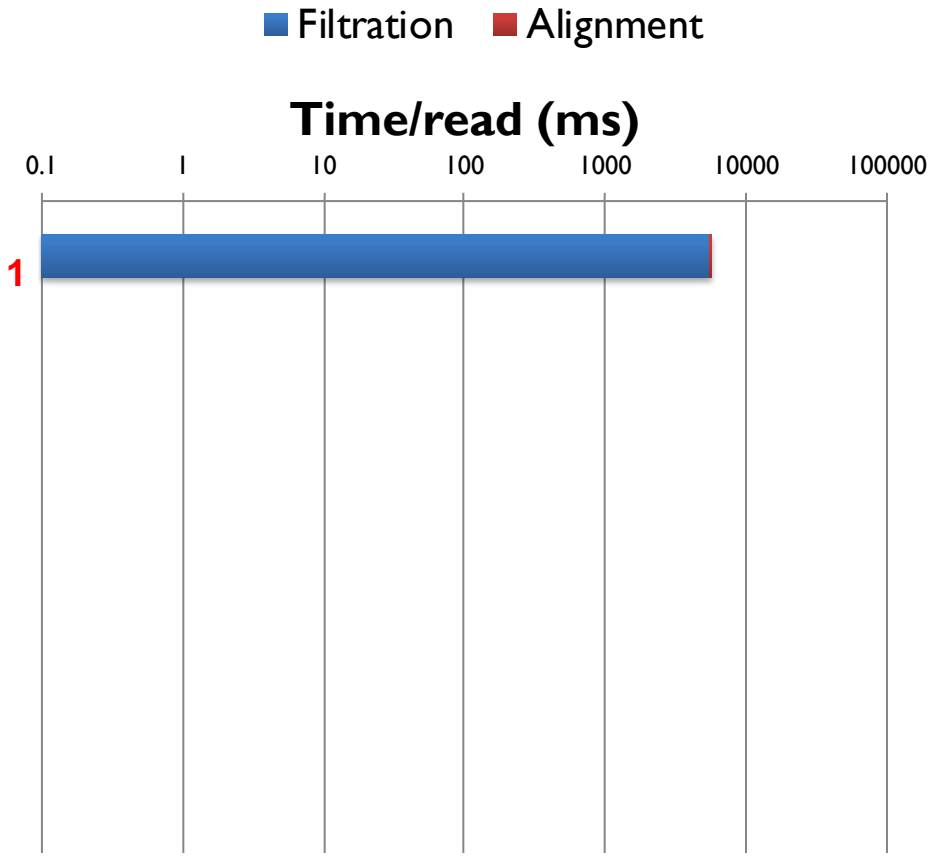


Keckler et al. Micro 2011.

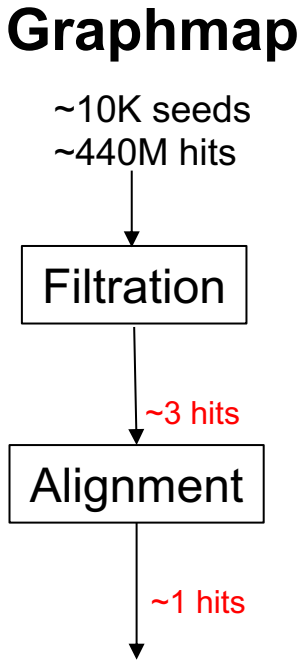
The Algorithm Often Has to Change

To Avoid Being Global Memory Limited

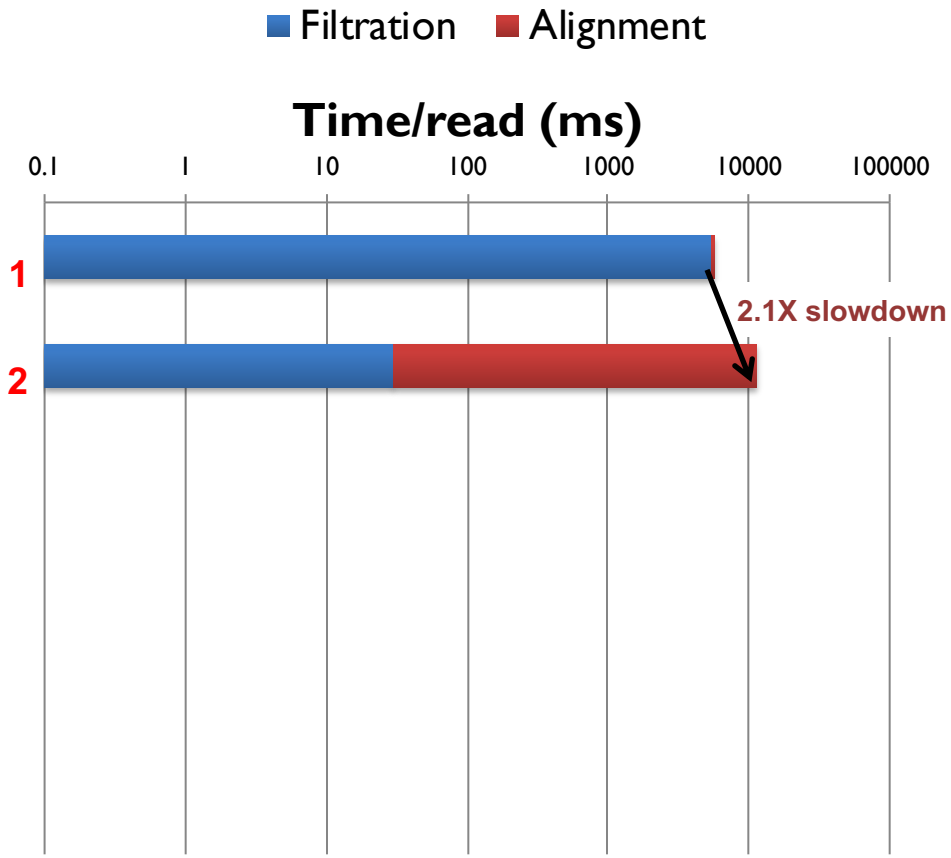
Algorithm-Architecture Co-Design for Darwin
Start with Graphmap



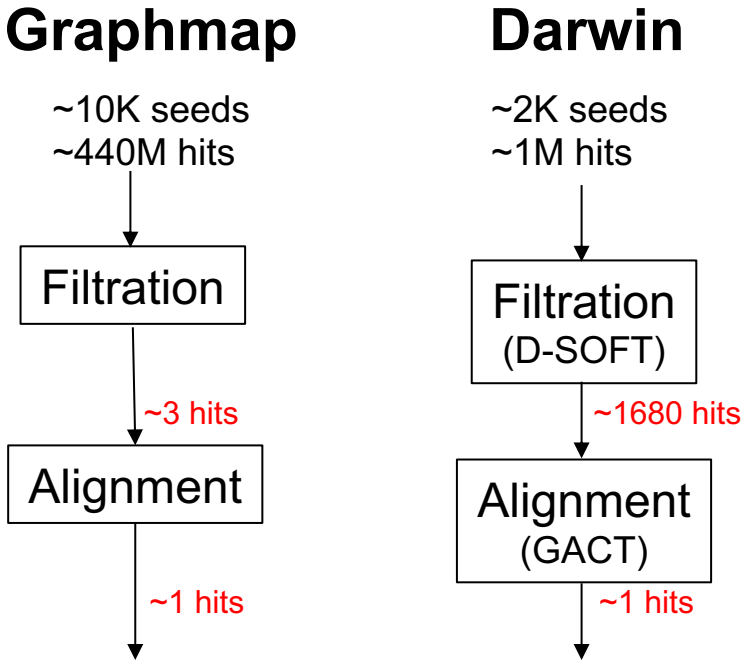
1. Graphmap (software)



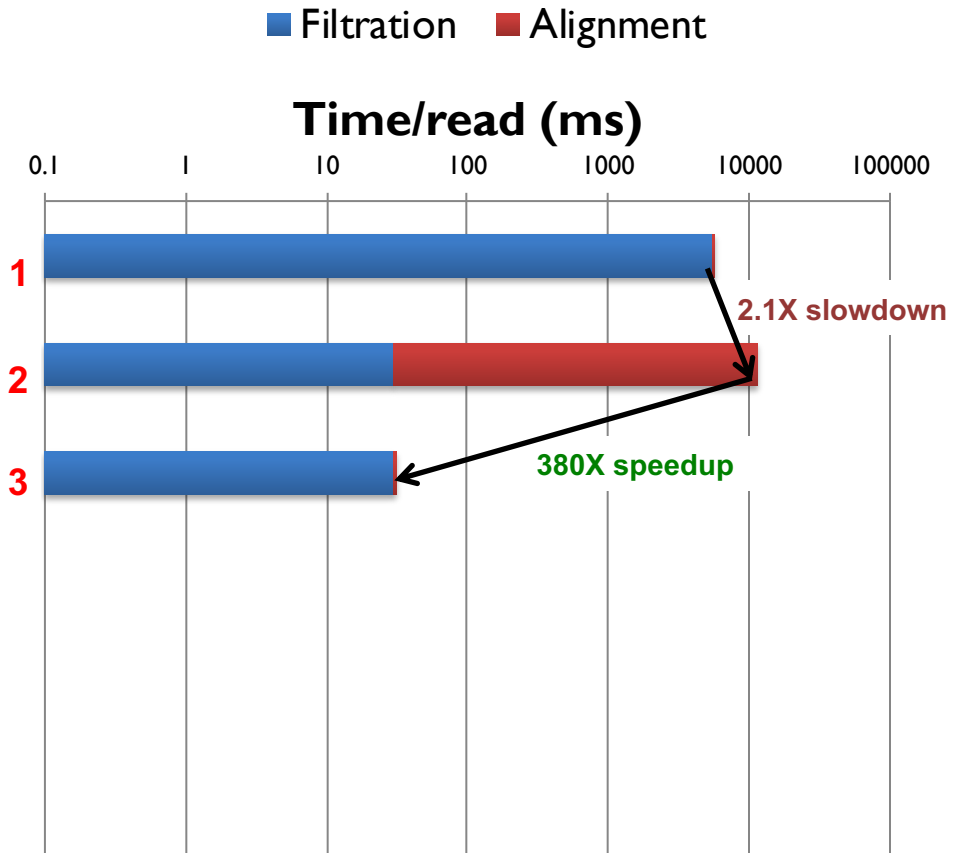
Algorithm-Architecture Co-Design for Darwin
Replace Graphmap with Hardware-Friendly Algorithms
Speed up Filtering by 100x, but 2.1x Slowdown Overall



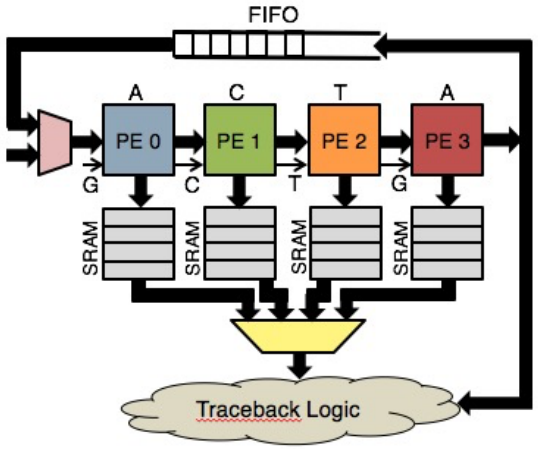
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)



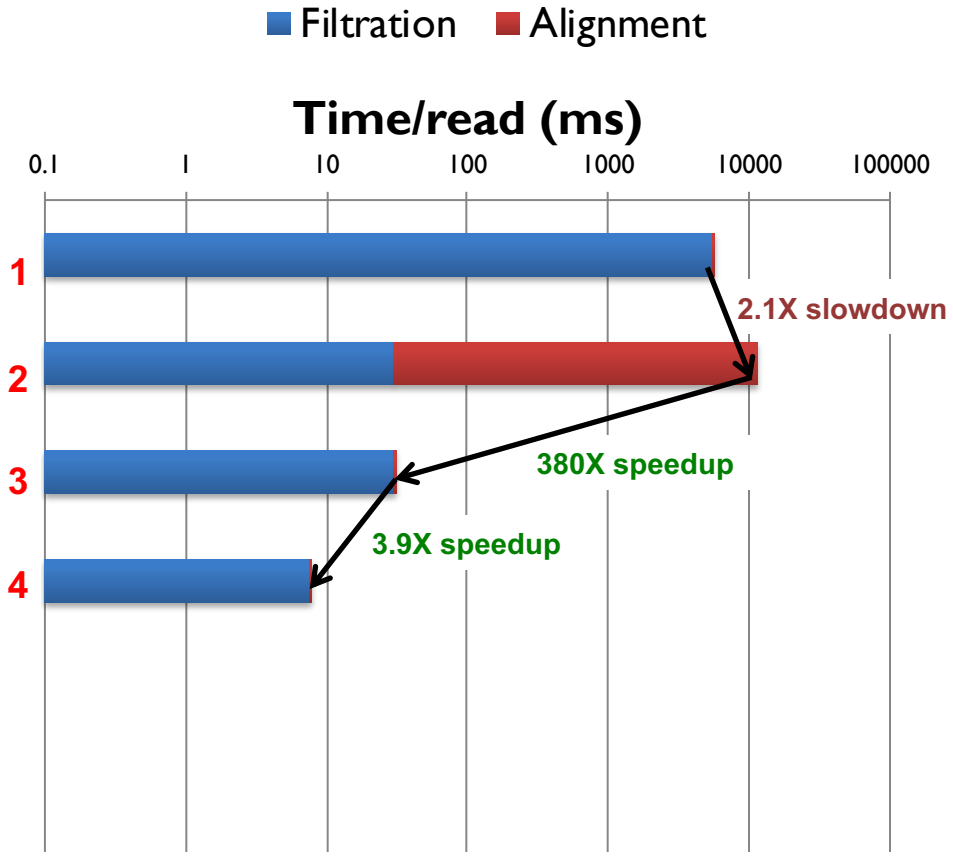
Algorithm-Hardware Co-Design for Darwin
Accelerate Alighment – 380x Speedup



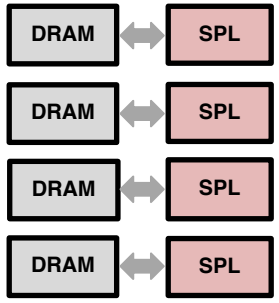
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration



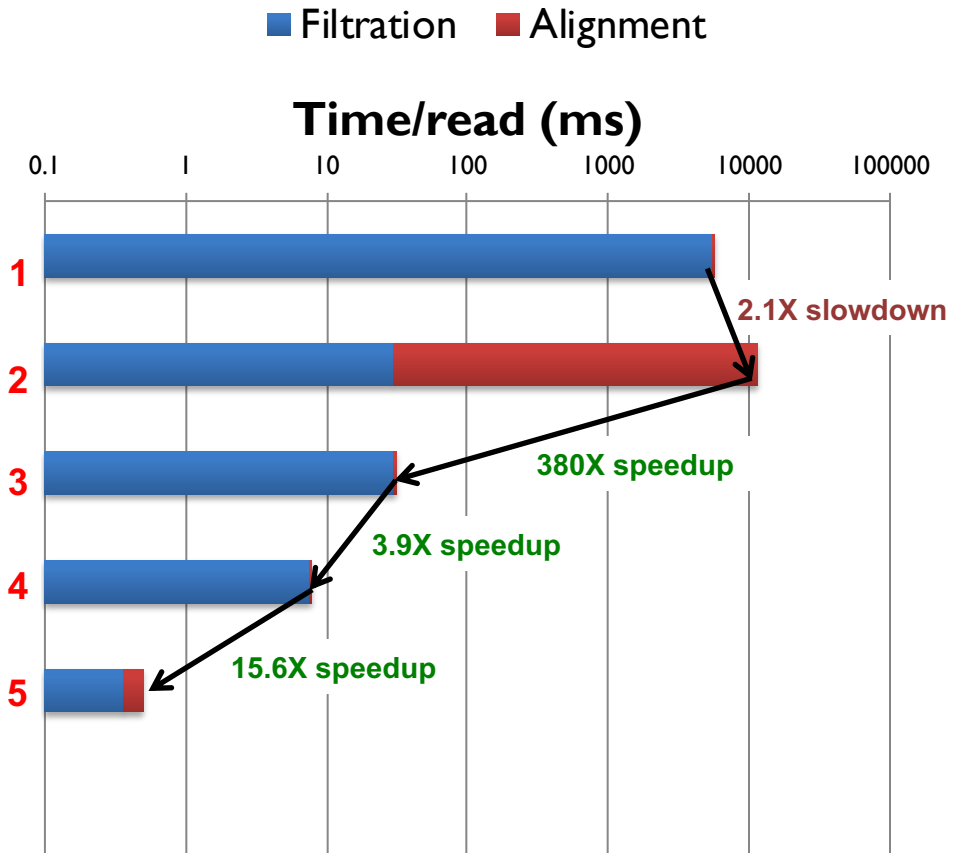
Algorithm-Hardware Co-Design for Darwin
4x Memory Parallelism – 3.9x Speeedup



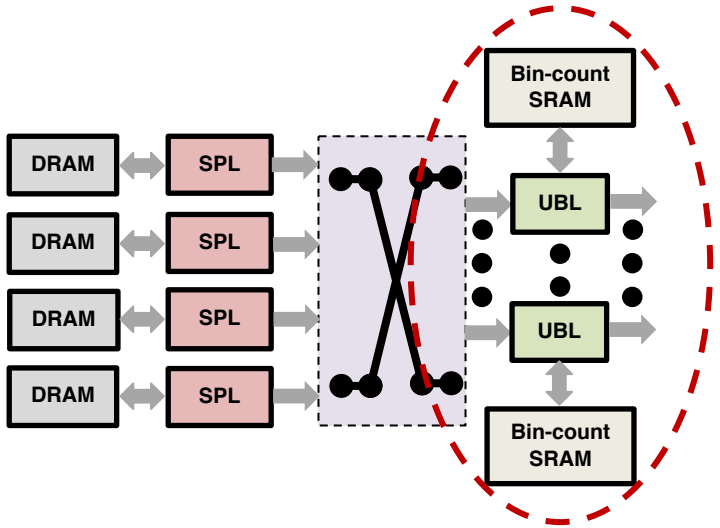
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration
4. Four DRAM channels for D-SOFT



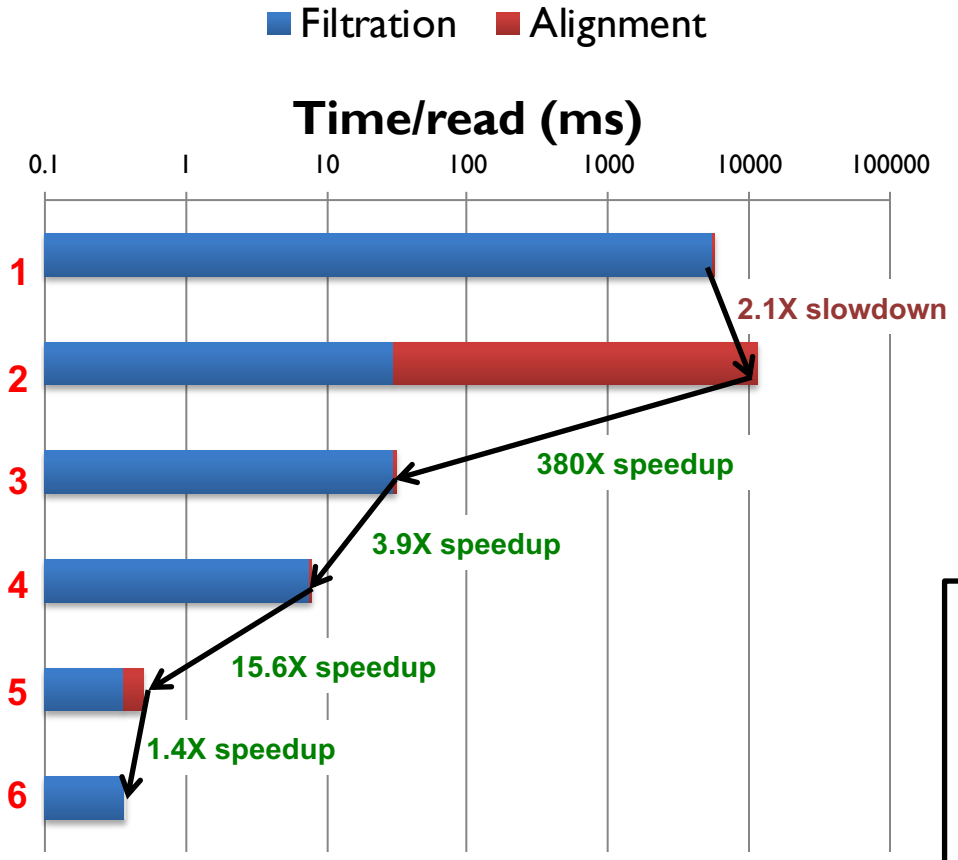
Algorithm-Hardware Co-Design for Darwin
Specialized Memory for D-Soft Bin Updates – 15.6x Speedup



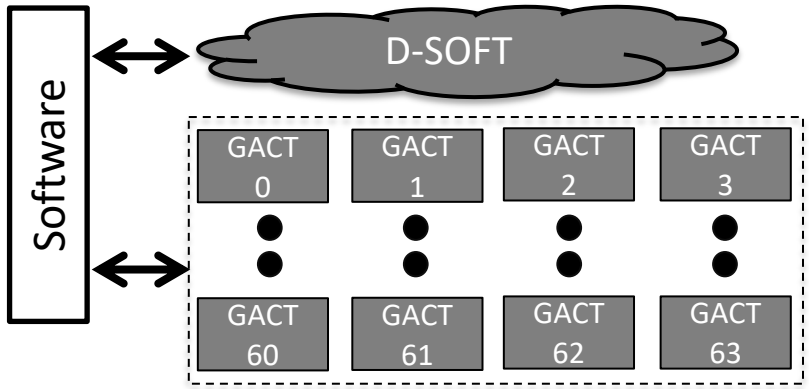
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration
4. Four DRAM channels for D-SOFT
5. Move bin updates in D-SOFT to SRAM (ASIC)



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

Memory Dominates

	Unit	Area (mm^2)	(%)	Power (W)	(%)
GACT	Logic	17.6	20.5%	1.04	23.6%
	Memory	68.0	79.5%	3.36	76.4%
D-SOFT	Logic	6.2	1.8%	0.41	4.4%
	Memory	320.3	98.2%	8.80	95.6%
EIE	Logic	2.8	6.9%	0.23	40.3%
	Memory	38.0	93.1%	0.34	59.7%

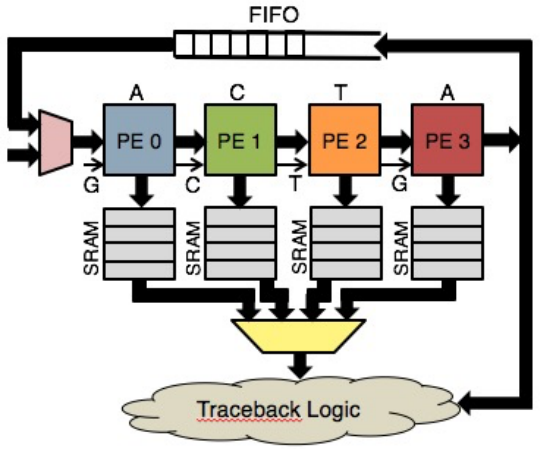
Algorithms must be memory optimized

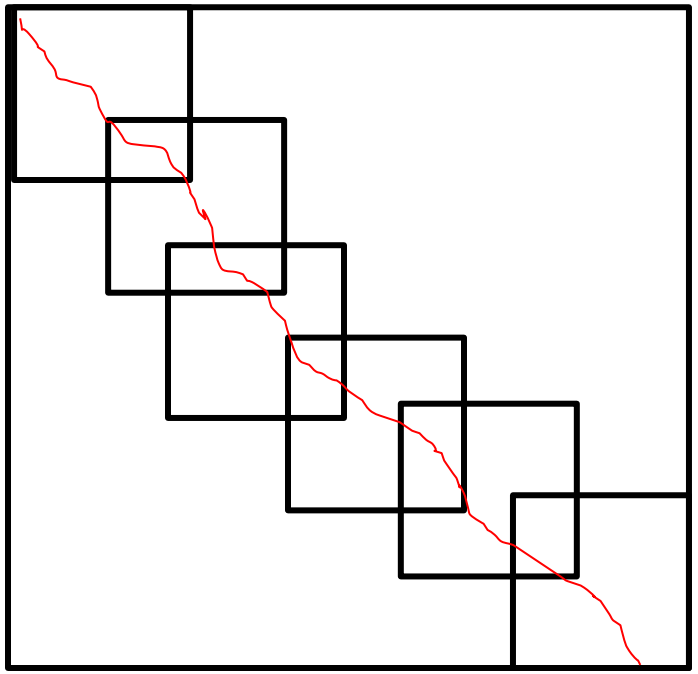
Minimize global memory accesses

Keep local memory footprint small

GACT Alignment

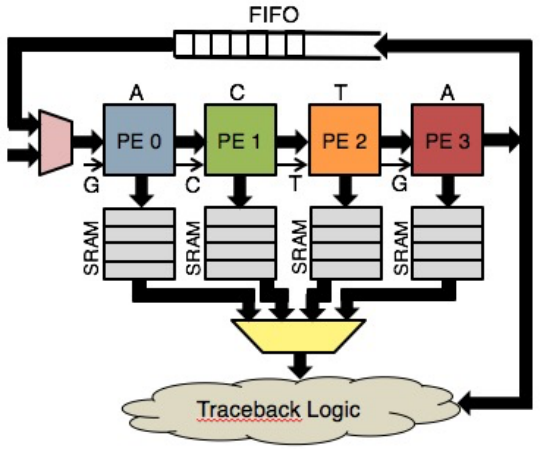
- 15M Reads, 10k bases each, ~2k hits each
  - ~300T Alignments to be done
  - Additional parallelism within each alignment
- But long reads have large (10M) memory footprint
- Solution: GACT (Tiling)

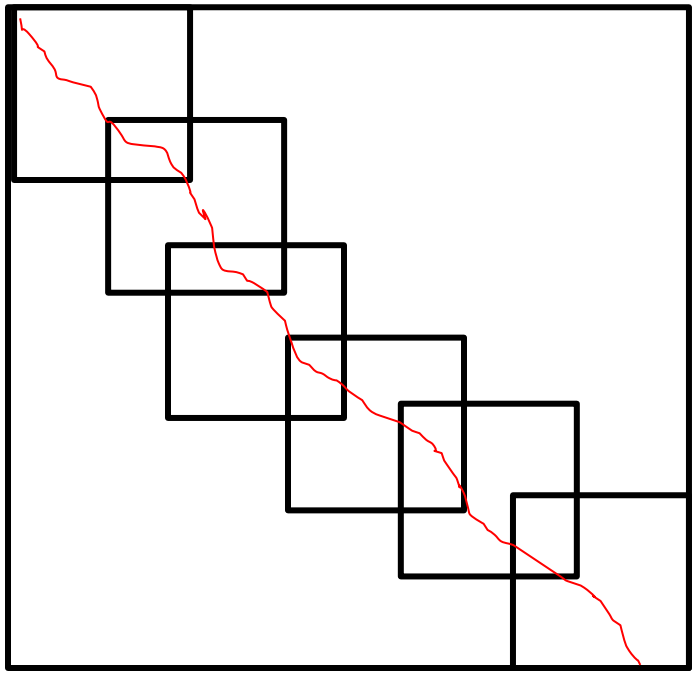




GACT Alignment

- 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)



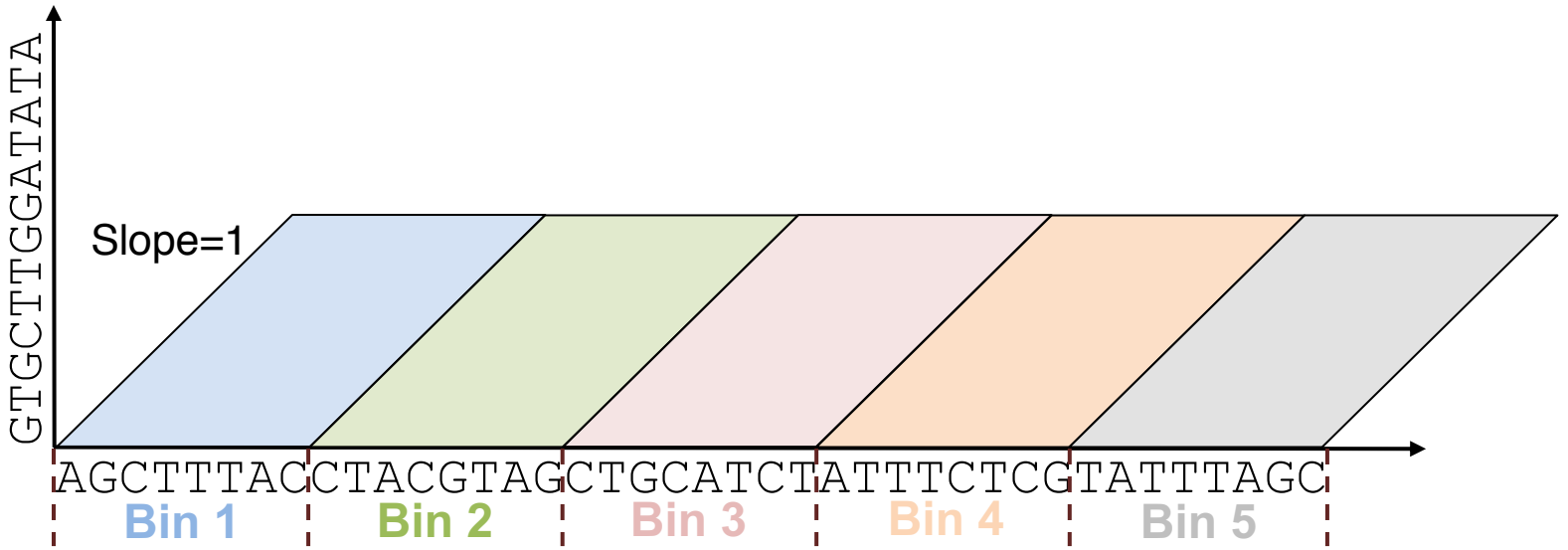


Darwin GACT hardware
4k PEs - 64 PEs per Array x 64 Arrays
~50 operations per cycle per PE
200k operations per cycle
Specialized memory
150,000x speedup vs CPU

On-Chip Memory
Cost per Bit is 10-100x Commodity DRAM

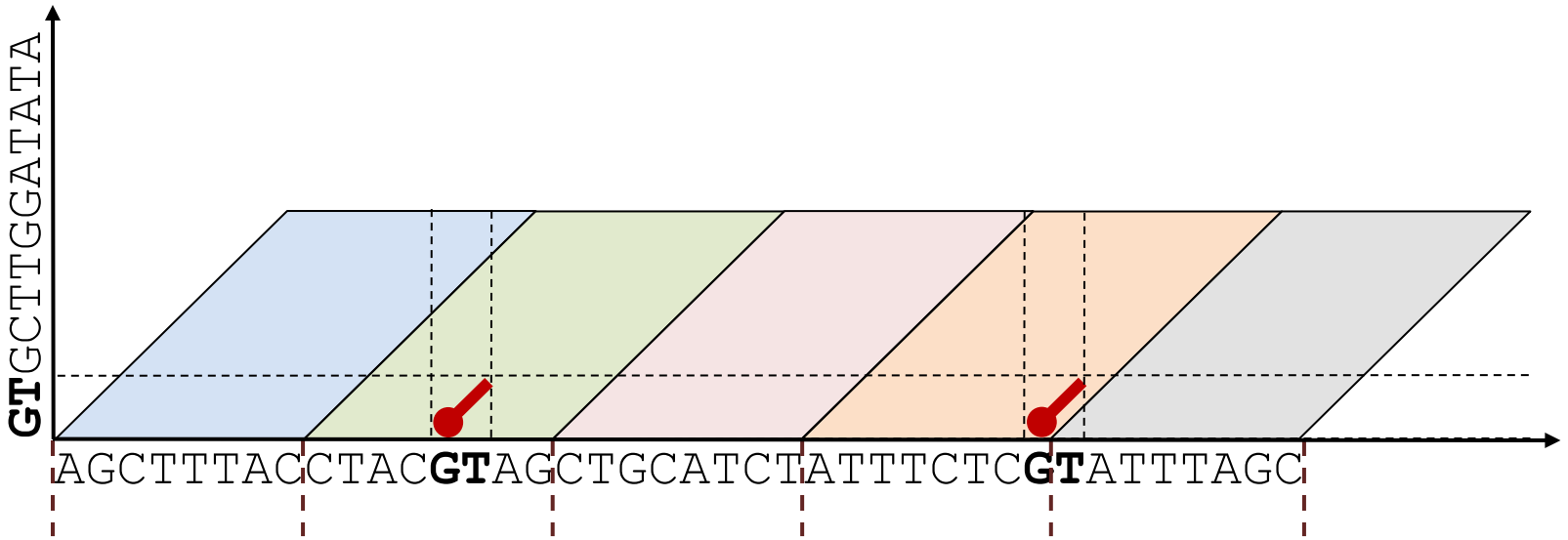
And It's Often Less Expensive

D-SOFT: Algorithm Overview



Bin count (bases)	Last hit offset
0	-inf
0	-inf
0	-inf
0	-inf
0	-inf

D-SOFT: Algorithm Overview



Pointer Table

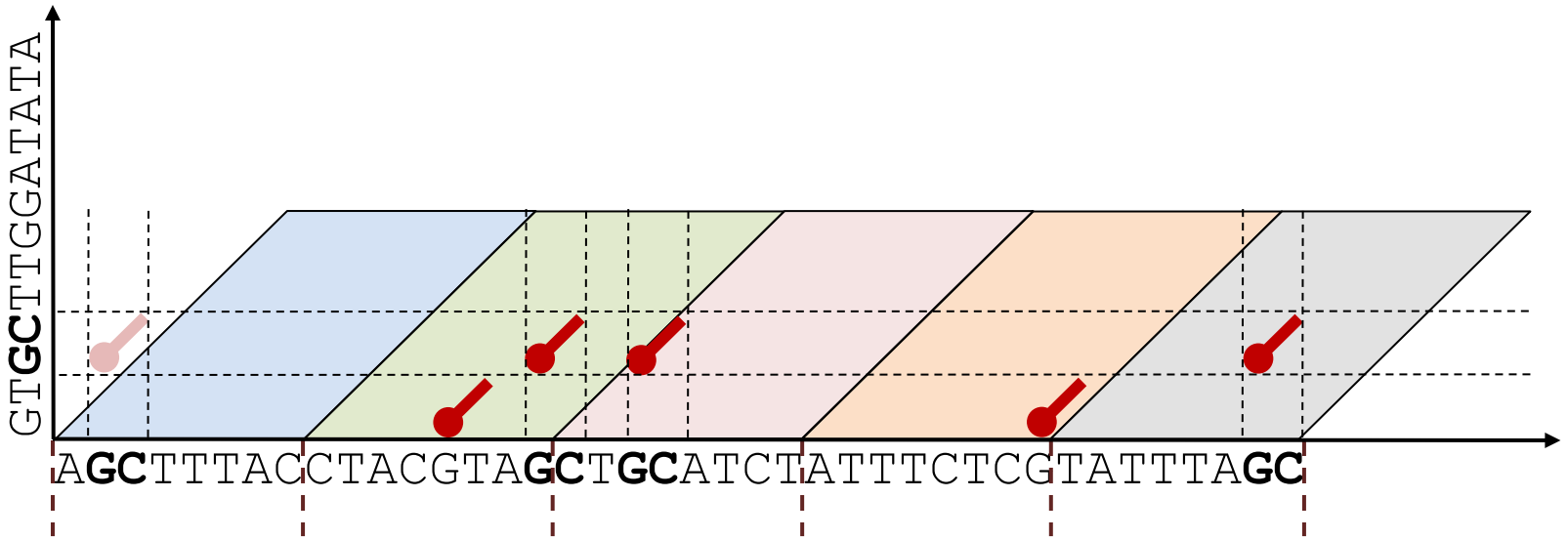
.	.
.	.
.	.
GG	21
GT	23
TA	29
.	.
.	.

Position Table

.	.
.	.
20	38
21	12
22	31
23	5
.	.
.	.
.	.

Bin count (bases)	Last hit offset
0	-inf
2	0
0	-inf
2	0
0	-inf

D-SOFT: Algorithm Overview



Pointer Table

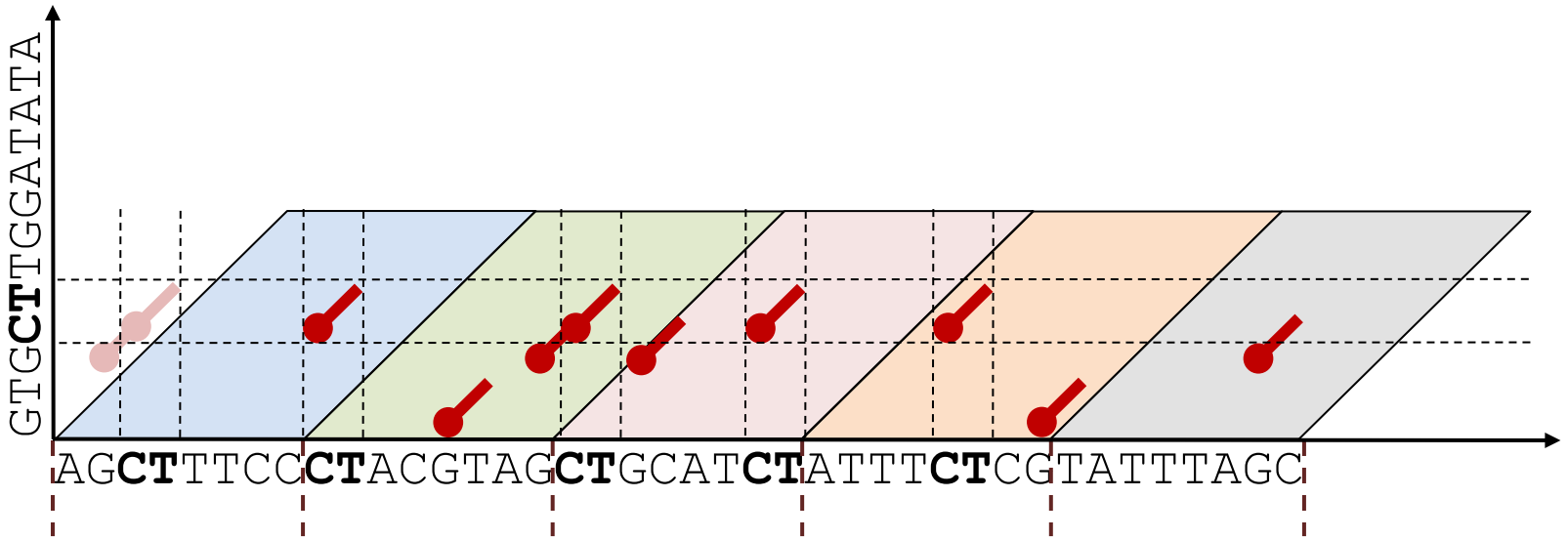
.	.
.	.
.	.
GA	17
GC	21
GG	21
.	.
.	.

Position Table

.	.
.	.
17	1
18	15
19	18
20	38
21	.
.	.
.	.

Bin count (bases)	Last hit offset
0	-inf
4	2
2	2
2	0
2	2

D-SOFT: Algorithm Overview



Pointer Table

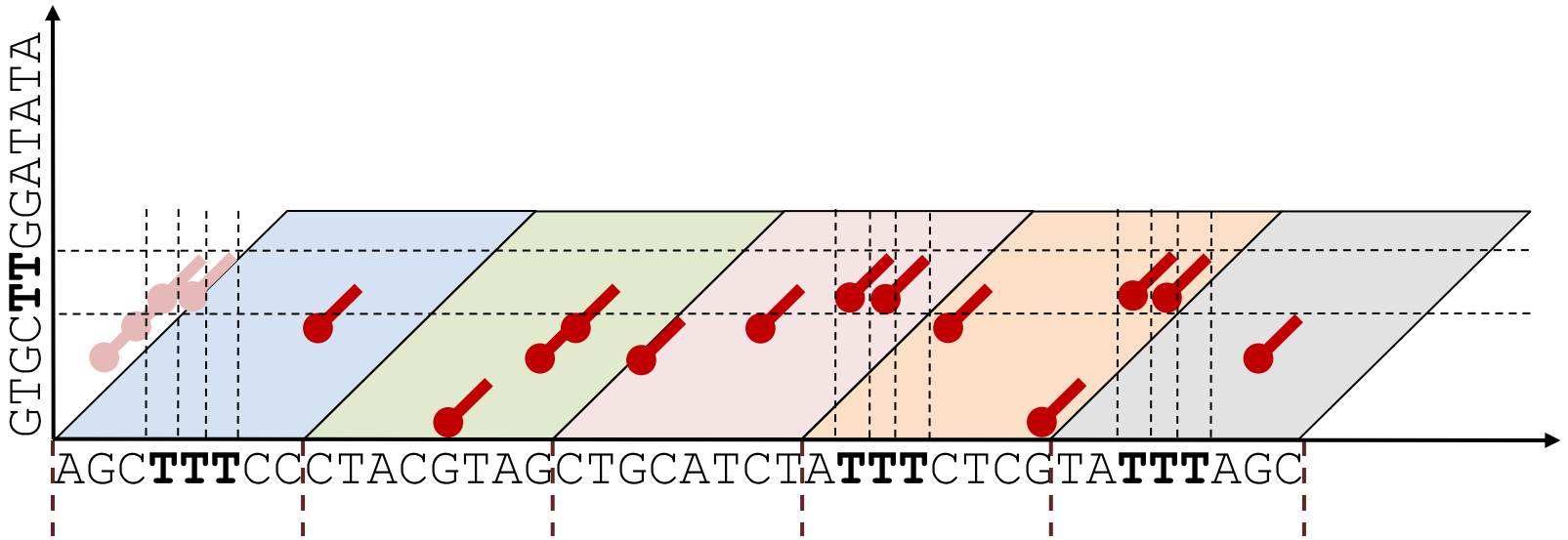
.	.
.	.
CG	12
CT	17
GA	17
.	.
.	.
.	.

Position Table

.	.
12	2
13	8
14	16
15	22
16	28
17	1
.	.
.	.

Bin count (bases)	Last hit offset
2	3
5	3
3	3
4	3
2	2

D-SOFT: Algorithm Overview



Pointer Table

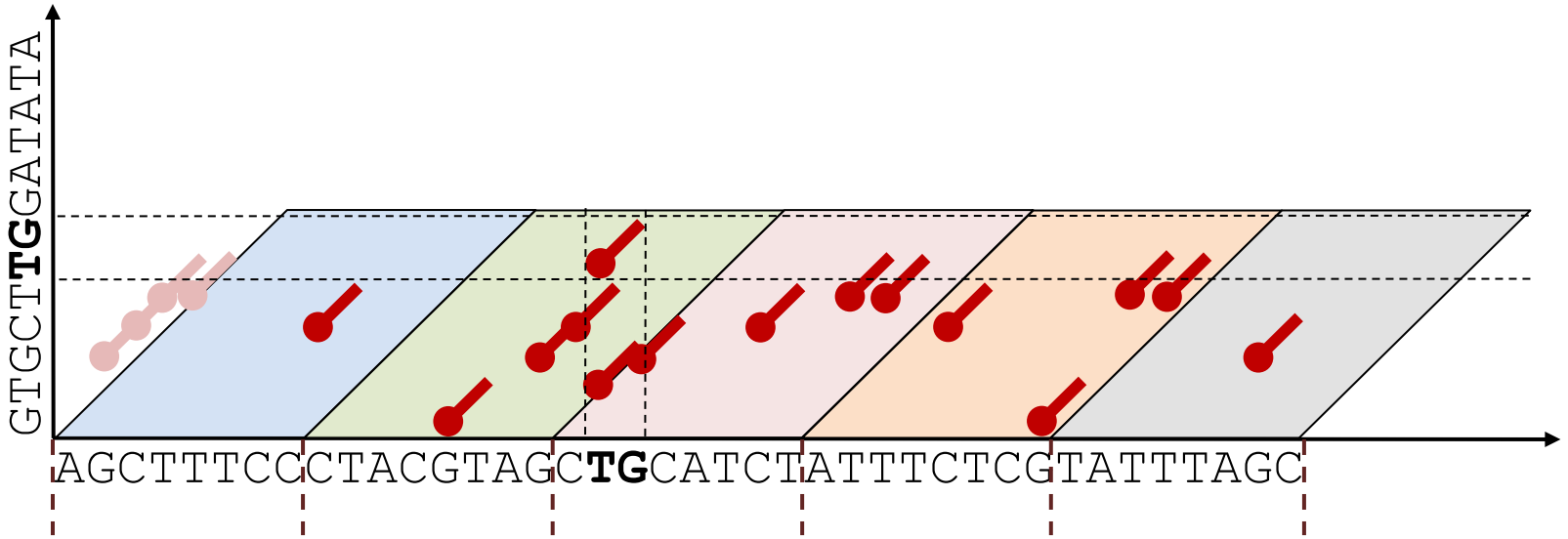
.	.
.	.
.	.
.	.
.	.
TC	32
TG	33
TT	39

Position Table

.	.
.	.
.	.
33	3
34	4
35	25
36	26
37	34
38	35

Bin count (bases)	Last hit offset
2	3
5	3
4	4
5	4
2	2

D-SOFT: Algorithm Overview



Pointer Table

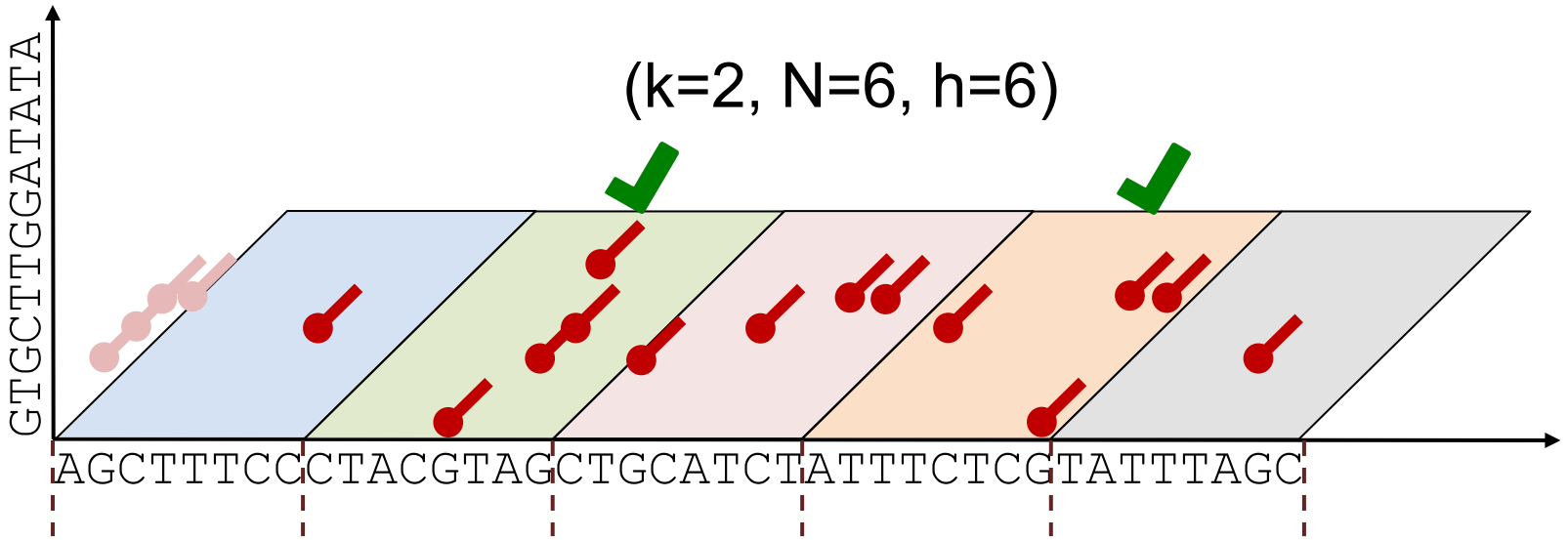
.	.
.	.
.	.
.	.
.	.
TC	32
TG	33
TT	39

Position Table

.	.
.	.
.	.
.	.
32	17
33	3
34	4
.	.
.	.

Bin count (bases)	Last hit offset
2	3
7	5
4	4
5	4
2	2

D-SOFT: Algorithm Overview



Parameters:

k: seed size

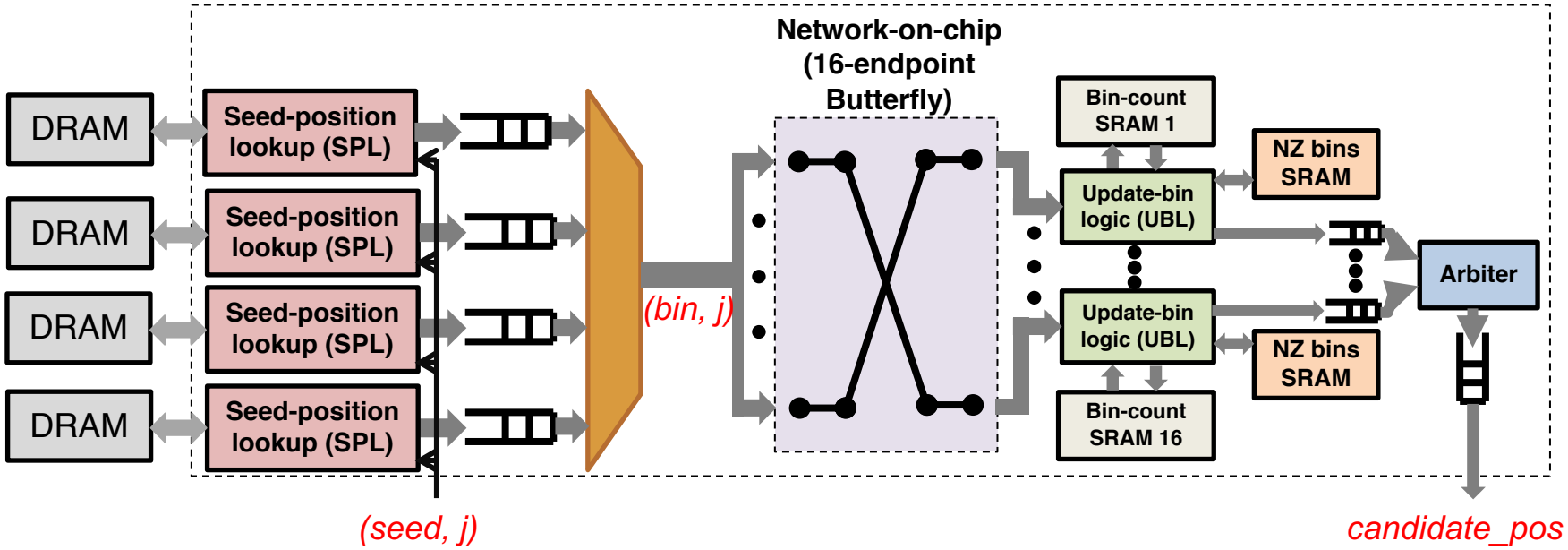
N: number of seeds

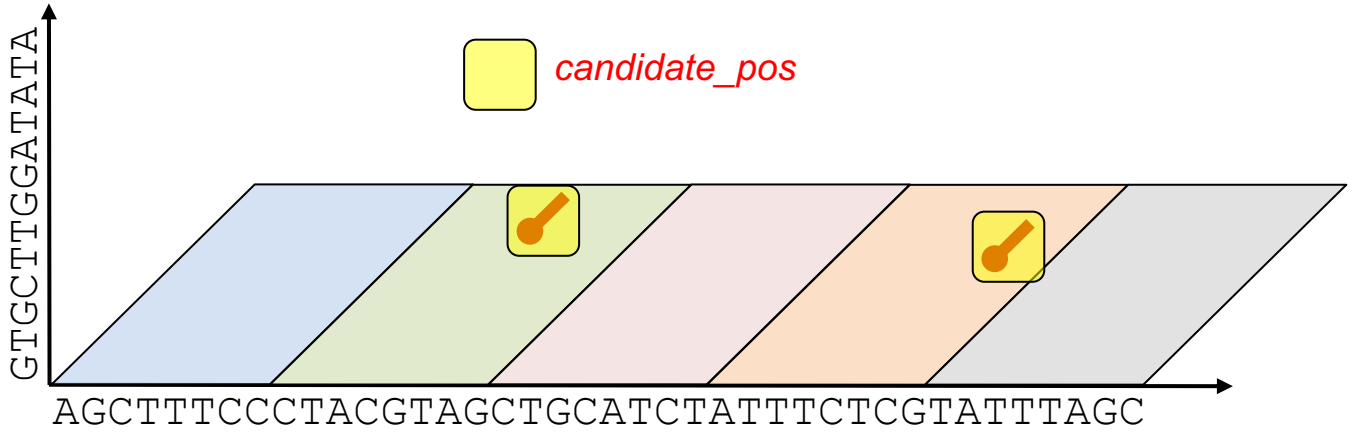
h: threshold on non-overlapping bases

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

Bin count (bases)	Last hit offset
2	3
7	5
4	4
5	4
2	2

D-SOFT: Hardware-acceleration





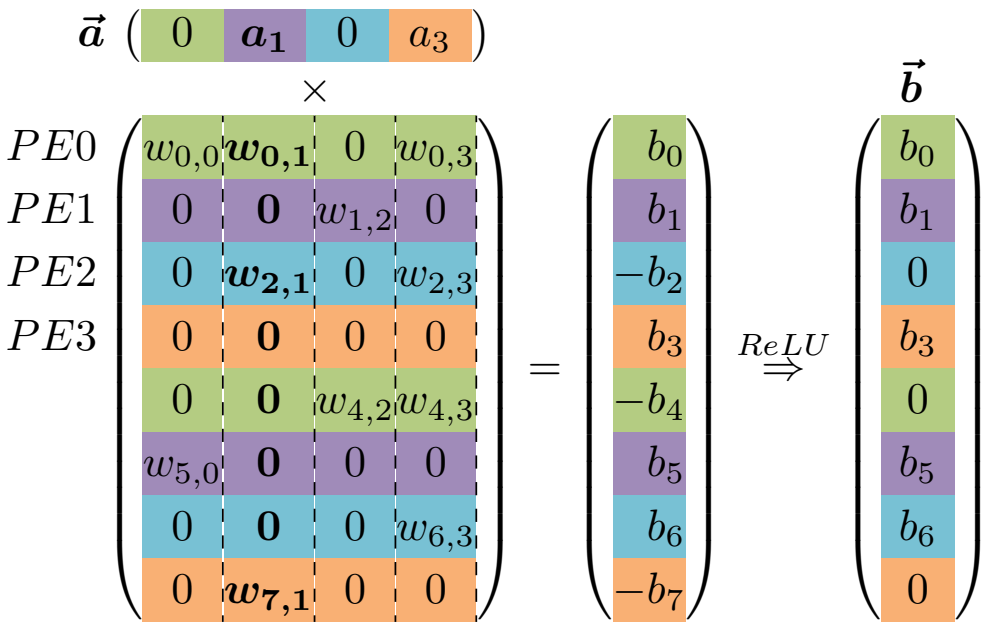
Cost has a Time Component

$$C = T(B_1N_1 + B_2N_2 + \ldots + P)$$

	T	B_1	N_1	B_2	N_2	C
Darwin Filter	1	100	64M	1	128G	134G
All DRAM	15.6			1	128G	1,997G

Hardware Enables Irregular, Compressed Data Structures (reduces memory footprint)

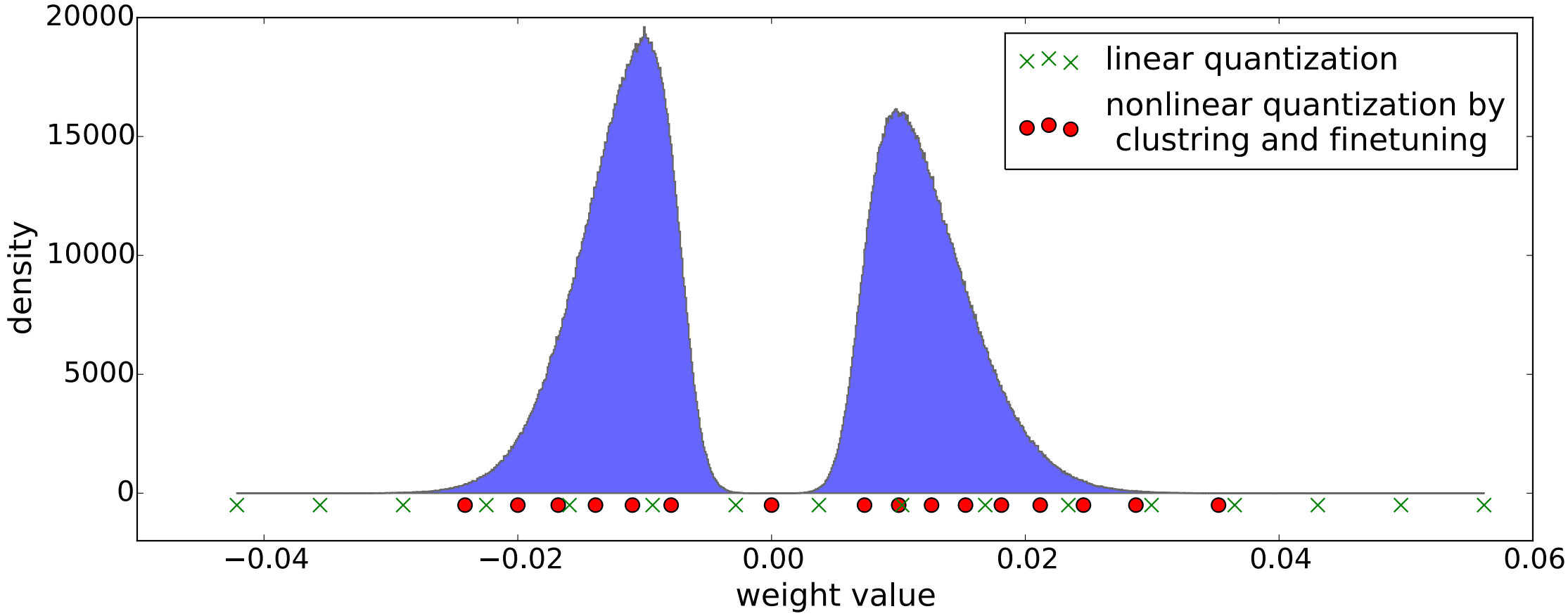
Dynamic Sparse Activations, Static Sparse Weights



Virtual Weight	$W_{0,0}$	$W_{0,1}$	$W_{4,2}$	$W_{0,3}$	$W_{4,3}$
Relative Index	0	1	2	0	0
Column Pointer	0	1	2	3	

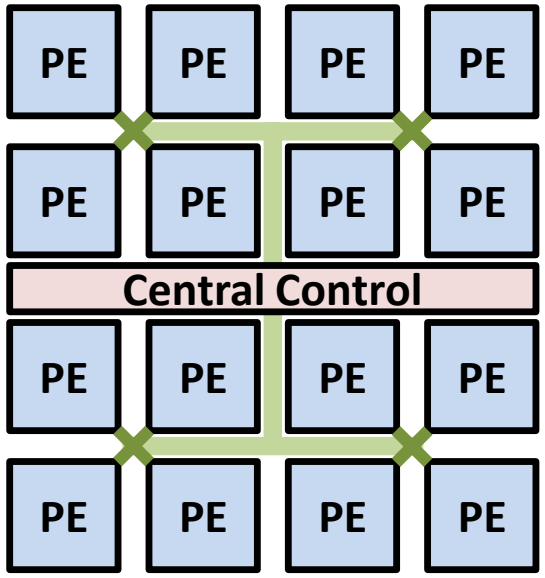
Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A. and Dally, W.J., 2016, June. EIE: efficient inference engine on compressed deep neural network, ISCA 2016

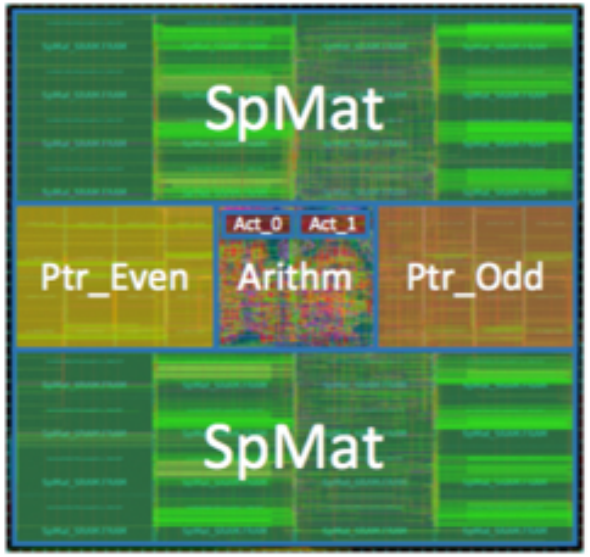
Trained Quantization

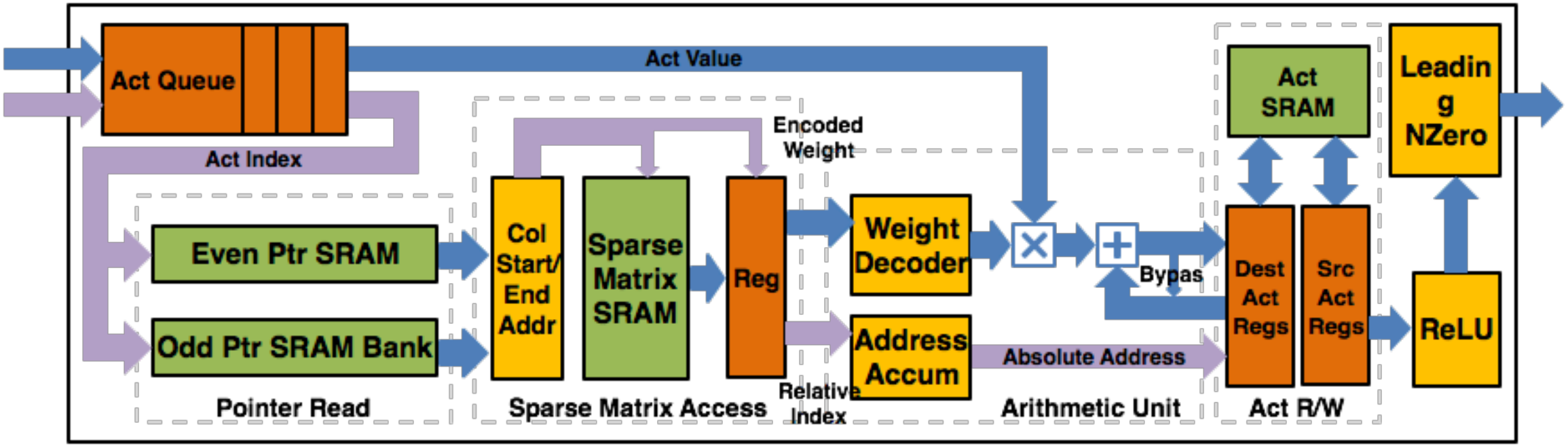


Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, arXiv 2015

EIE Hardware

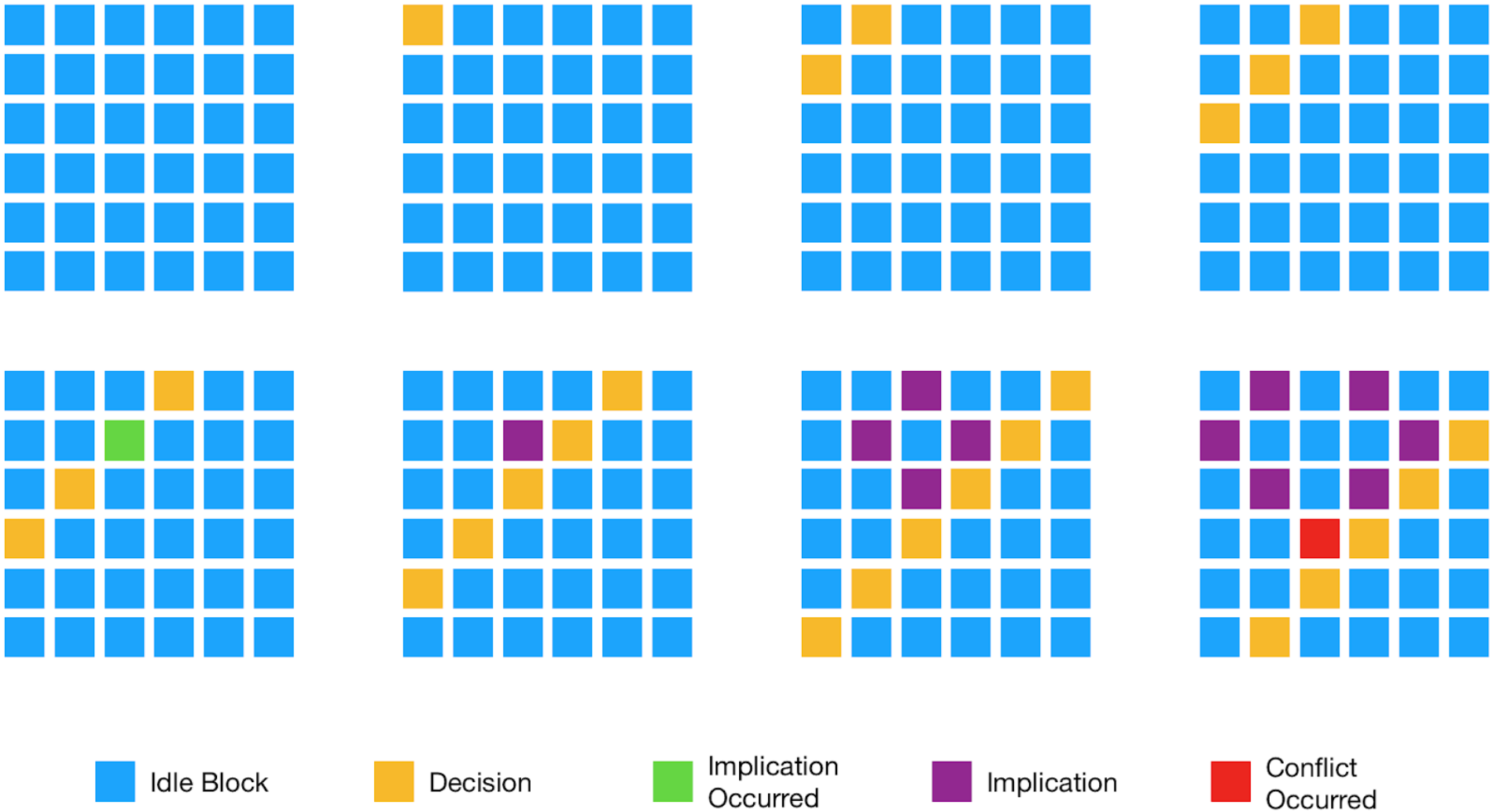






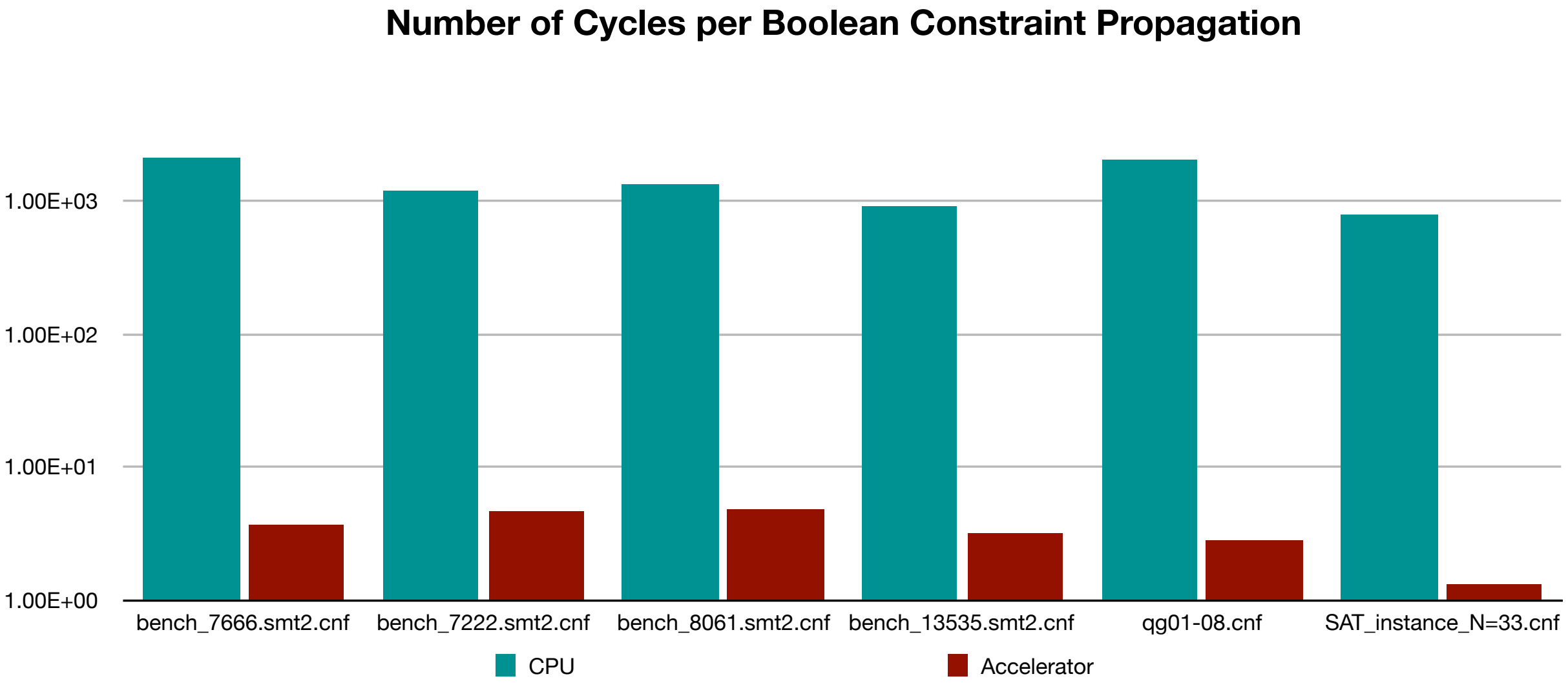
Simple Parallelism Often Beats a “Better” Algorithm

Broadcast Literals to Associative Memory of Clauses More Comparisons but Lower Latency

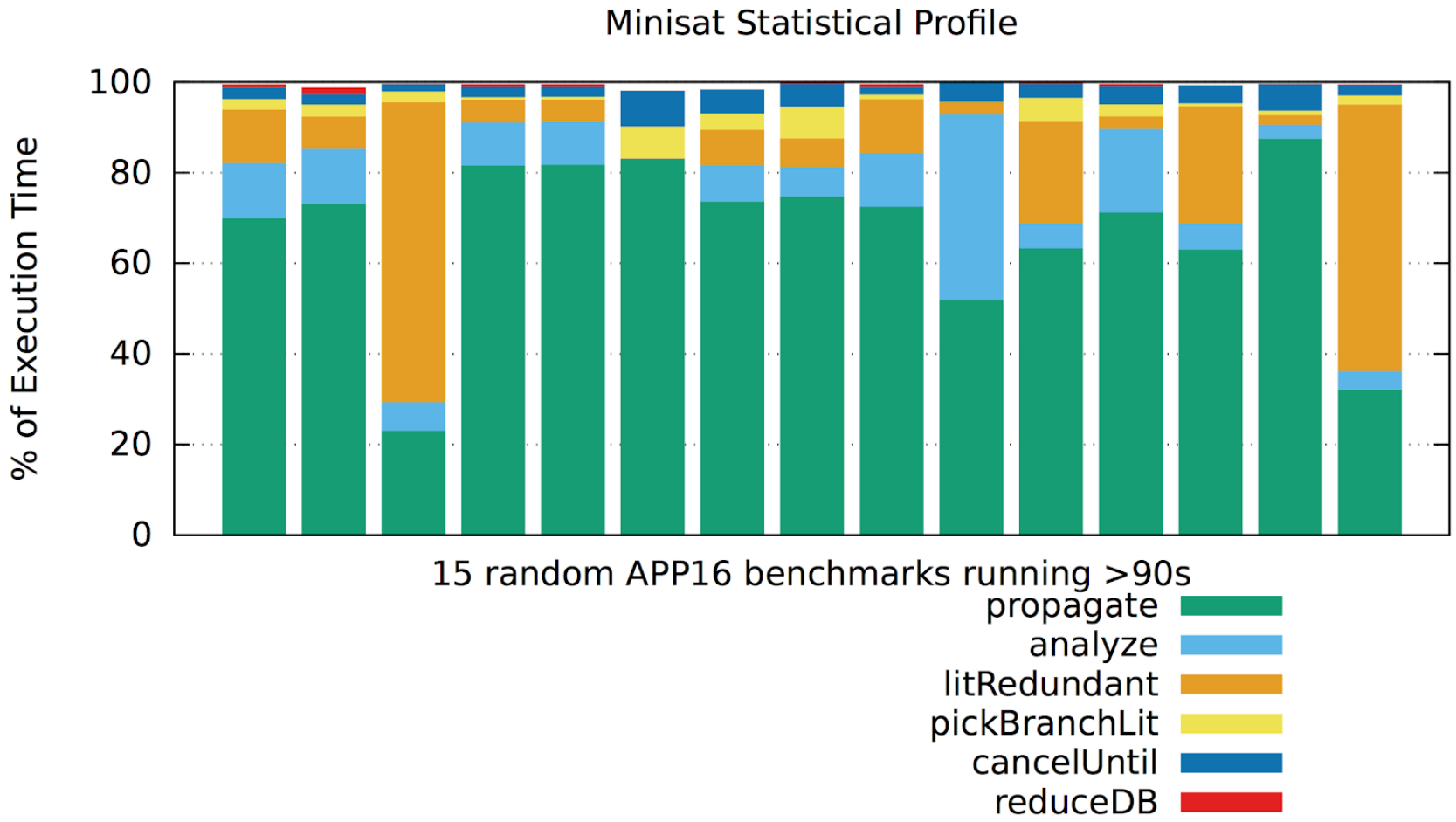


Need to Accelerate the Whole Problem

Implication 300-500x Faster than CPU

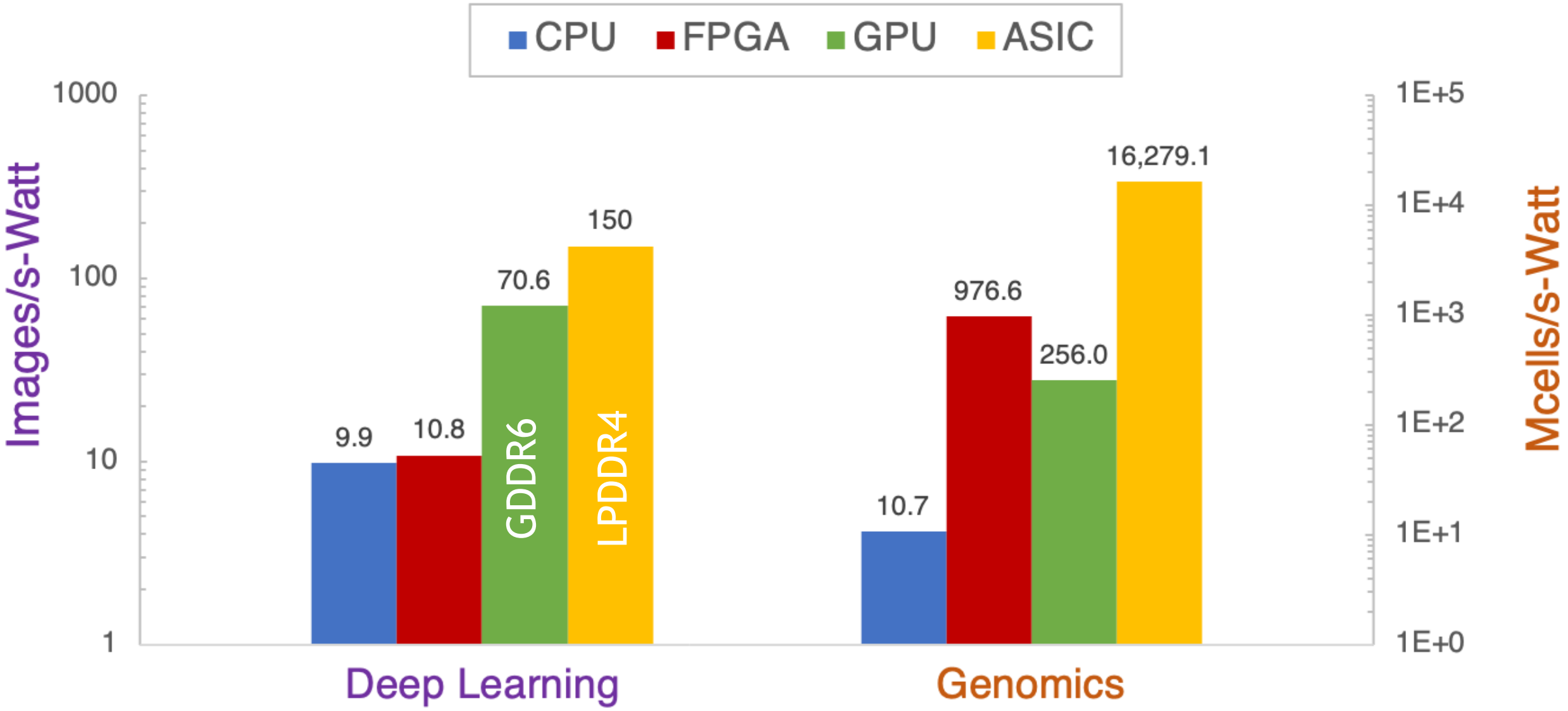


Need to Accelerate the **Whole** Problem



Platforms for Acceleration

Implementation Alternatives

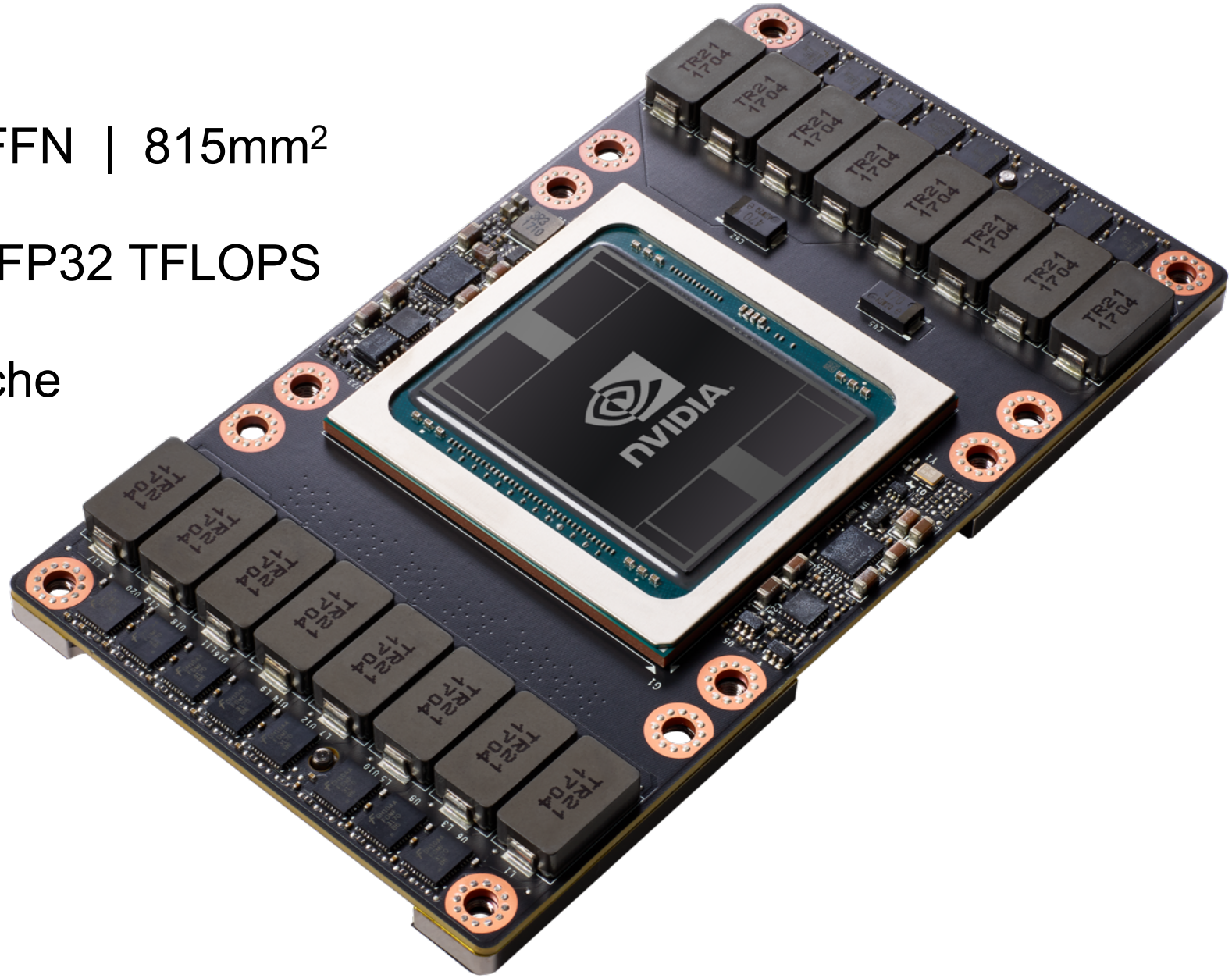


GPUs Provide:

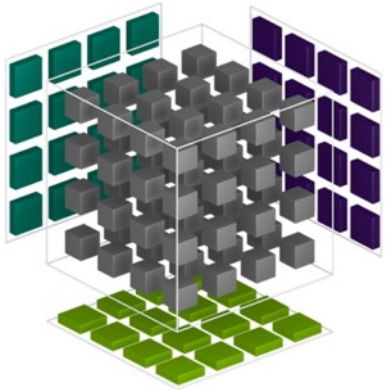
- High-Bandwidth, Hierarchical **Memory** System
  - Can be configured to match application
- Programmable **Control** and **Operand Delivery**
- Simple places to bolt on **Domain-Specific Hardware**
  - As instructions or memory clients

Volta V100

21B xtors | TSMC 12nm FFN | 815mm^2
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



$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

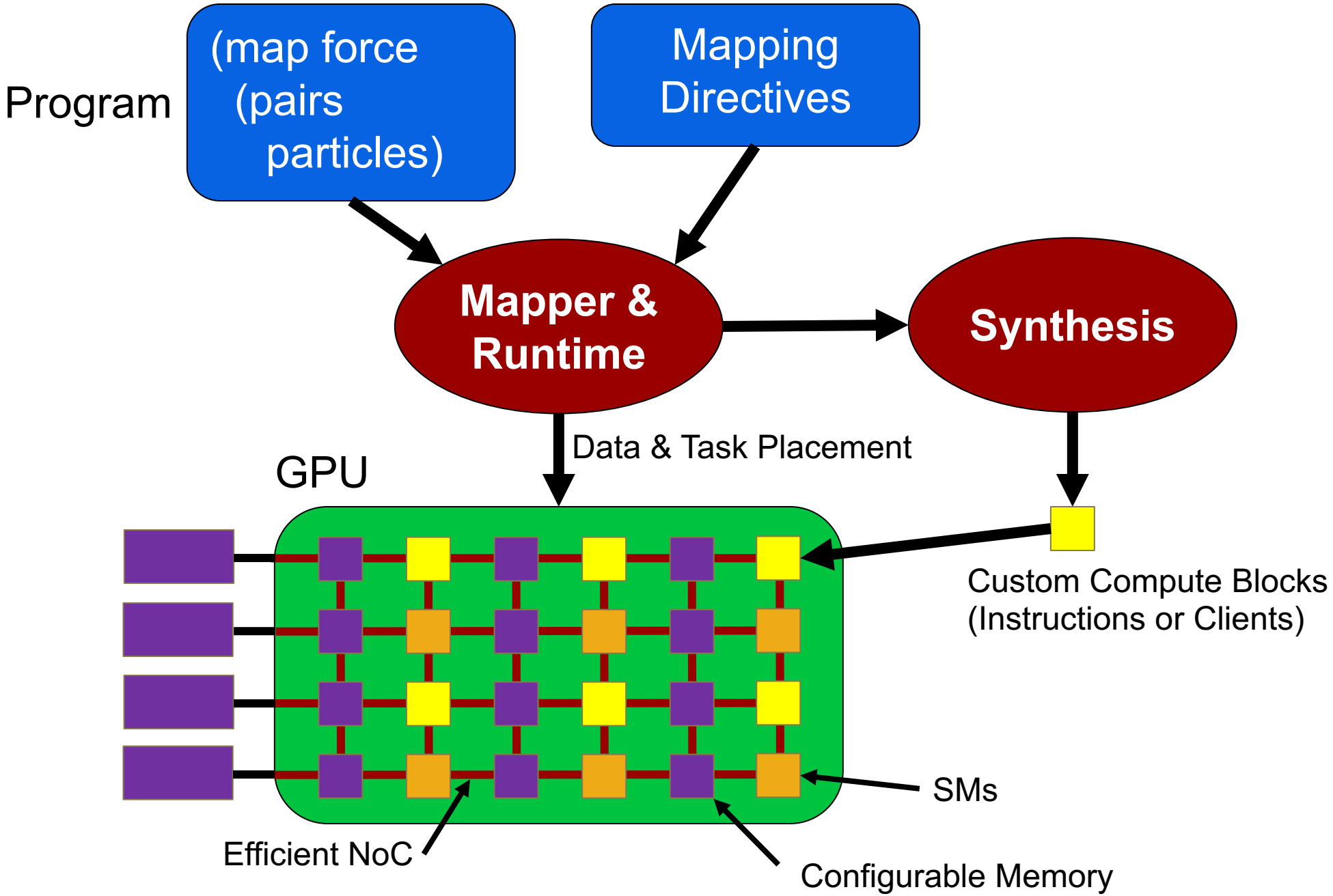
FP16 FP16 FP16 or FP32

$$D = AB + C$$

Specialized Instructions Amortize Overhead

Operation	Ops	Energy**	Overhead*	
			Vs op	%tot
HFMA	2	1.5pJ	20x	95%
HDP4A	8	6.0pJ	5x	83%
HMMA	128	130pJ	0.23x	19%
IMMA	1024	230pJ	0.13x	12%

*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
GPU
Custom Compute Blocks
(Instructions or Clients)
SMs
Efficient NoC
Configurable Memory

DSA Design is Programming

With a Hardware Cost Model

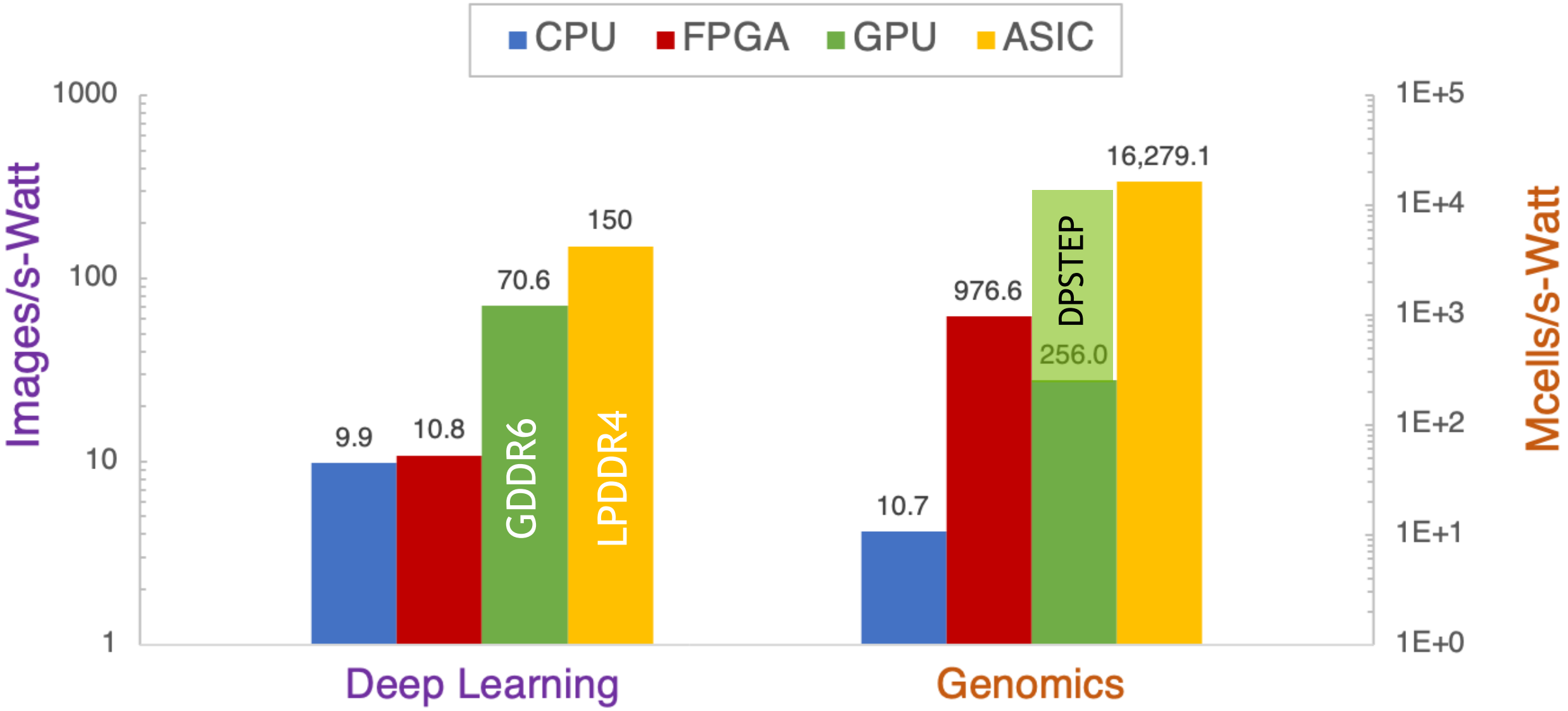
Algorithm

```
tb ← GACT(r, q)
input  : r[TS], q[TS]
output : tb[TS,TS]
for i = 0..TS-1 do
    for j = 0..TS-1 do
        in (i,j) ← Max (h (i,j-1) - O, in (i, j-1) - E)
        del (i,j) ← Max (h(i-1,j) - O, del (i-i,j) - E)
        h (i,j) ← Max (0, in(i,j), del (i,j), h (i-1, j-1) + W
          (r[i],q[j]))
        tb [i,j] ← ComputeTb (h (i,j), in (i,j), del (i,j))
    end
end
```

Mapping

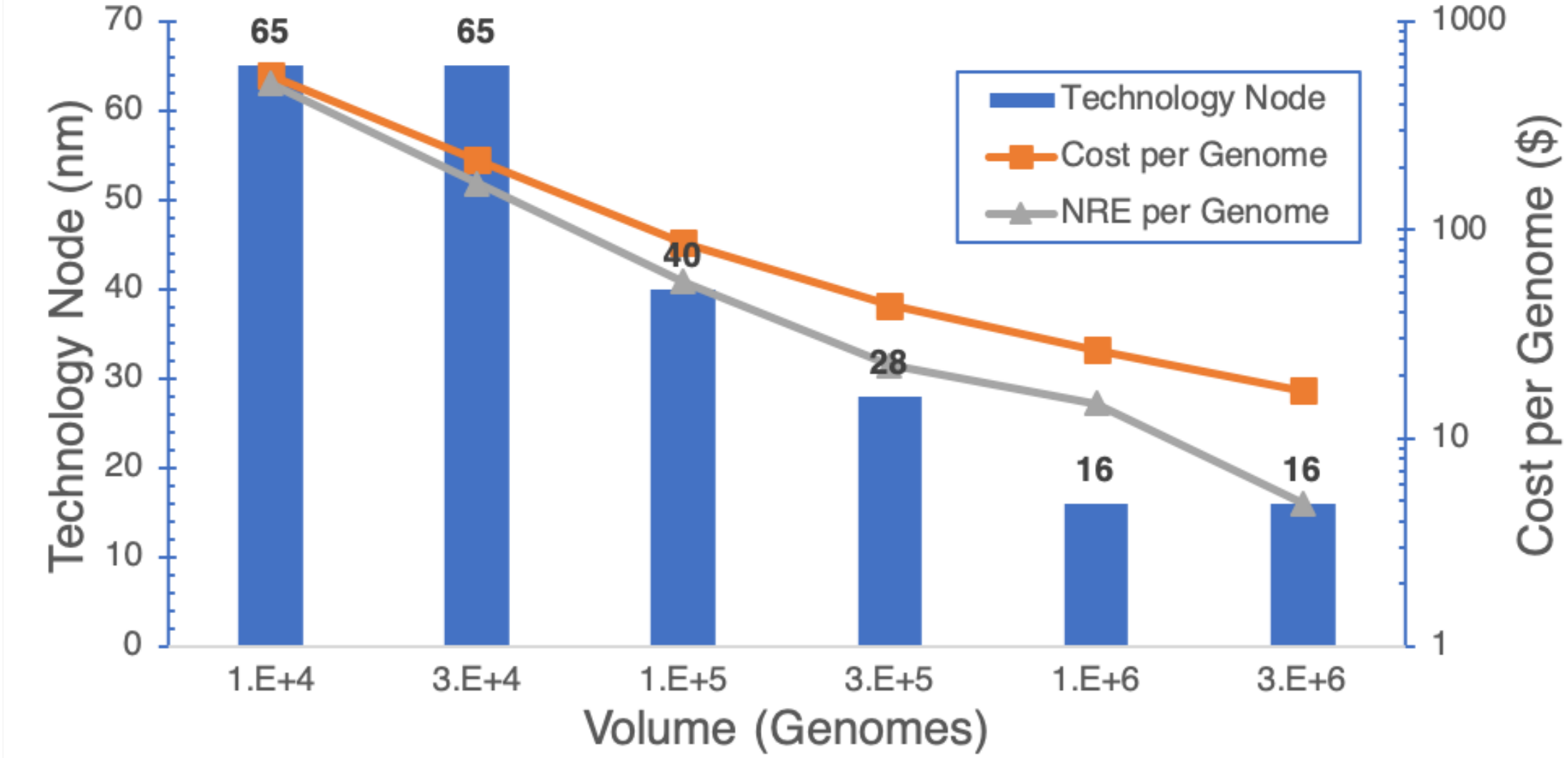
```
STRIPES ← TS / AS
processor_array p (AS)
memory_array tbm (AS)[STRIPES, TS ]
Map h (i,j) → p (i % AS)
    at t = (i % AS) · TS + j - i / AS
Map tb [i,j] → tbm (i % AS) [i / AS, j]
```

Implementation Alternatives



Multi-Chip Modules (MCMs)

Total Cost of Ownership



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