The Future of Computing: Domain-Specific Architecture

Chesapeake Large Scale Analytics Conference

October 5, 2021

Bill Dally

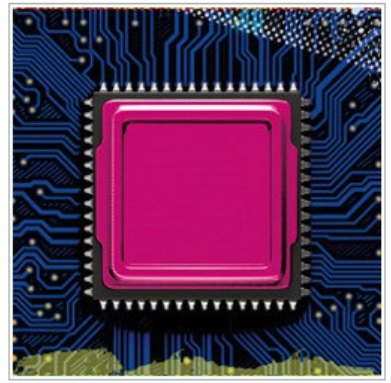
Chief Scientist and SVP of Research, NVIDIA Corporation

Adjunct Professor of EE and CS, Stanford University

CONTRIBUTED ARTICLES

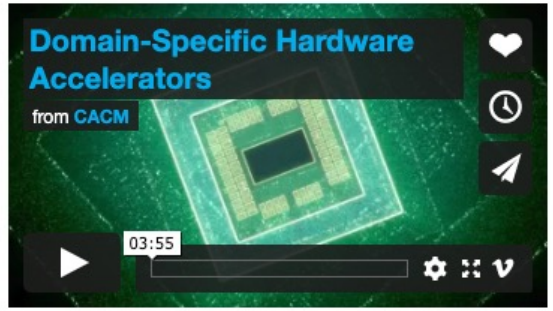
Domain-Specific Hardware Accelerators

By William J. Dally, Yatish Turakhia, Song Han
Communications of the ACM, July 2020, Vol. 63 No. 7, Pages 48-57
10.1145/3361682
Comments



Credit: Matt Herring, Bet_Noire / Getty Images

From the simple embedded processor in your washing machine to powerful processors in data center servers, most computing today takes place on general-purpose programmable processors or CPUs. CPUs are attractive because they are easy to program and because large code bases exist for them. The programmability of CPUs stems from their execution of sequences of simple instructions, such as ADD or BRANCH; however, the energy required to fetch and interpret an instruction is 10x to 4000x more than that required to perform a simple operation such as ADD. This high overhead was acceptable when processor performance and efficiency were scaling according to Moore's Law.[32] One could simply wait and an existing application would run faster and more efficiently. Our economy has become dependent on these increases in computing performance and efficiency to enable new features and new applications. Today, Moore's Law has largely ended,[12] and we must look to alternative architectures with lower overhead, such as domain-specific accelerators, to continue scaling of performance and efficiency. There are several ways to realize domain-specific accelerators as discussed in the sidebar on accelerator options.



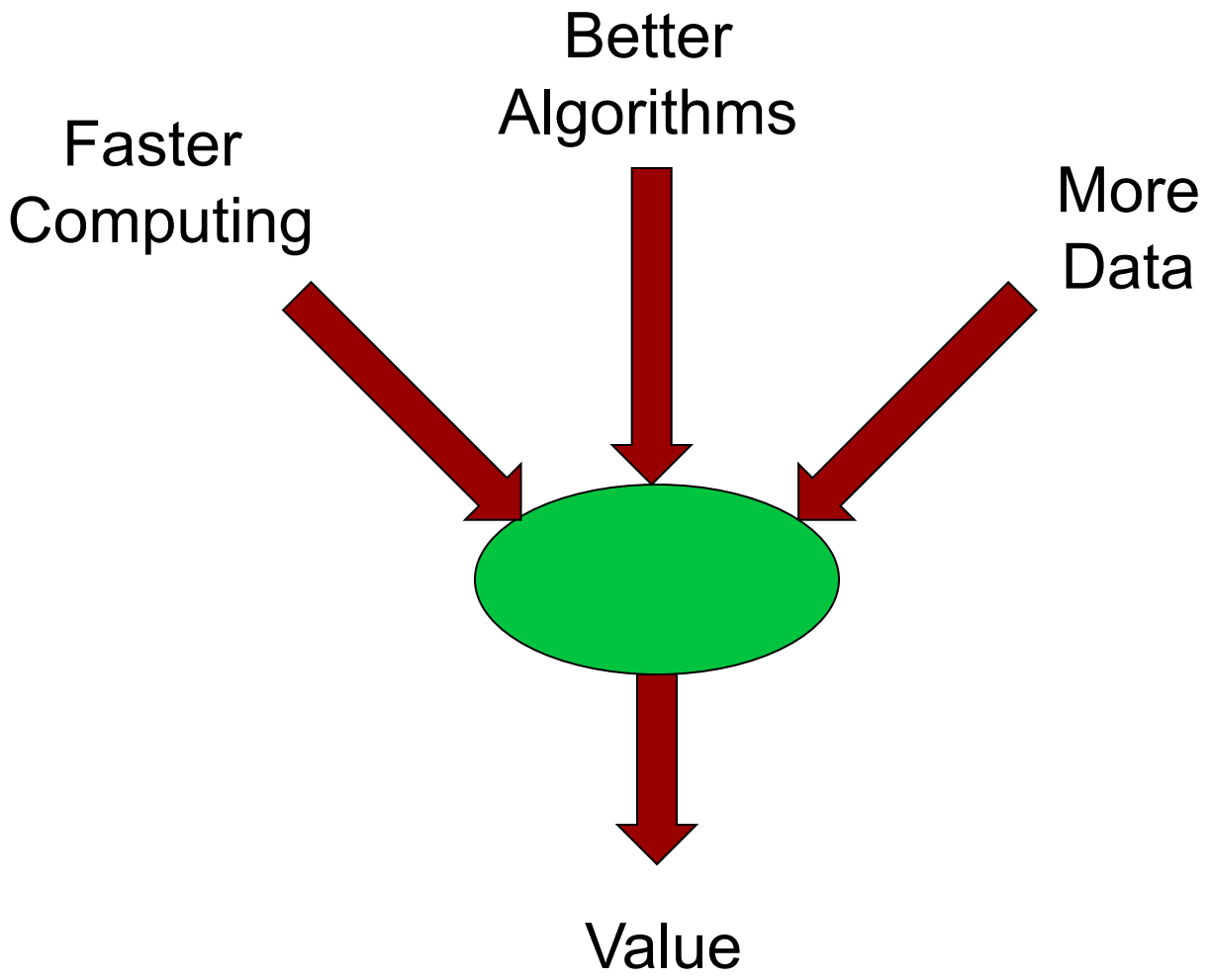
Back to Top

Key Insights

- Most speedup comes from parallelism

ARTICLE CONTENTS:
Introduction
Key Insights
Sources of Acceleration
Codesign is Needed
Memory Dominates Accelerators
Balancing Specialization and Generality
Total Cost of Ownership (TCO)
Accelerator Design
Conclusion
References
Authors
Footnotes
Sidebar: Acceleration Options

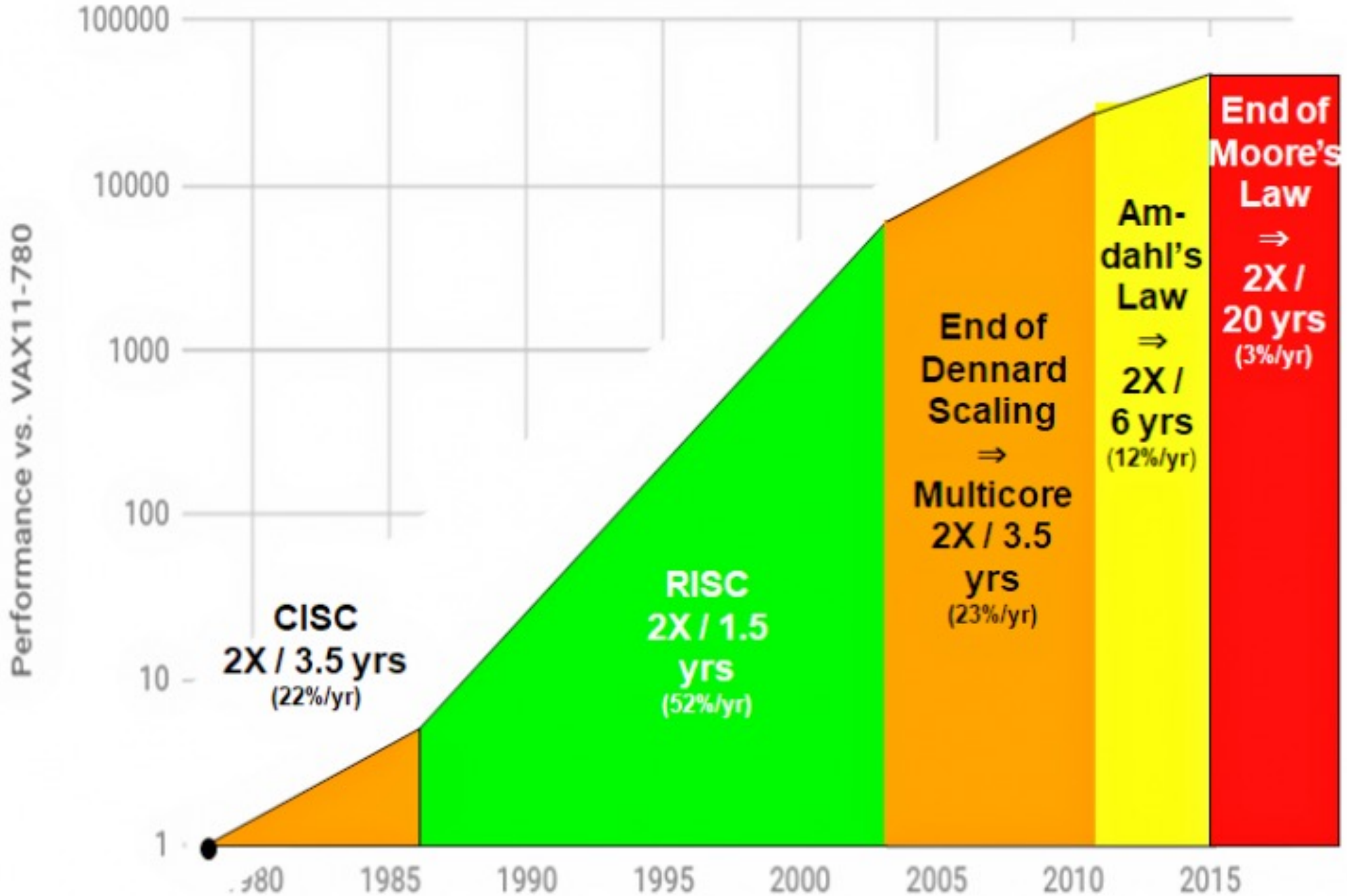
MORE NEWS & OPINIONS
Japan Post Delivery Robot Debuts in Tokyo
The Japan Times
AI Authorship?


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:

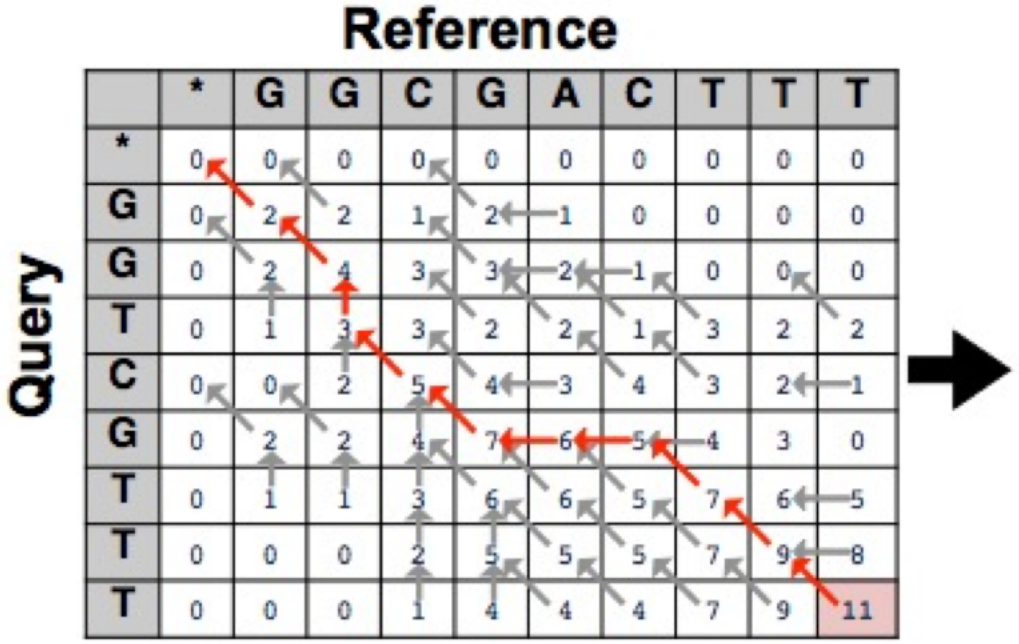
- Special **Data Types** and **Operations**
  - Do in 1 cycle what normally takes 10s or 100s – **10-1000x efficiency gain**
- Massive **Parallelism** – >1,000x, not 16x – with **Locality**
  - This gives performance, not efficiency
- Optimized **Memory**
  - High bandwidth (**and low energy**) for specific data structures and operations
- Reduced or Amortized **Overhead**
  - **10,000x efficiency gain** for simple operations
- Algorithm-Architecture **Co-Design**

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 (270,000x efficiency)

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)

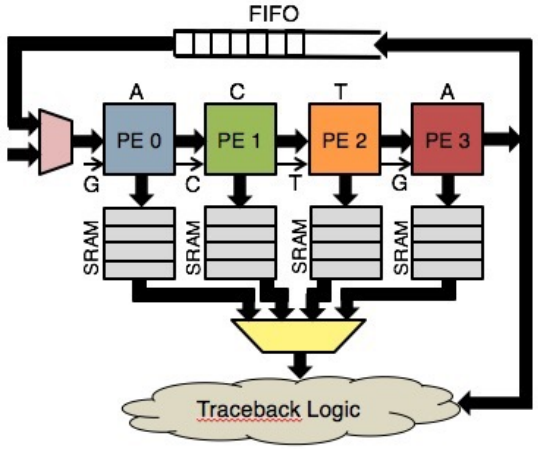
Evangelos Vasilakis. 2015. *An Instruction Level Energy Characterization of Arm Processors.* Foundation of Research and Technology Hellas, Inst. of Computer Science, Tech. Rep. FORTH-ICS/TR-450 (2015)

Specialization -> Efficiency

Efficiency -> Parallelization

Parallelization -> Speedup

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

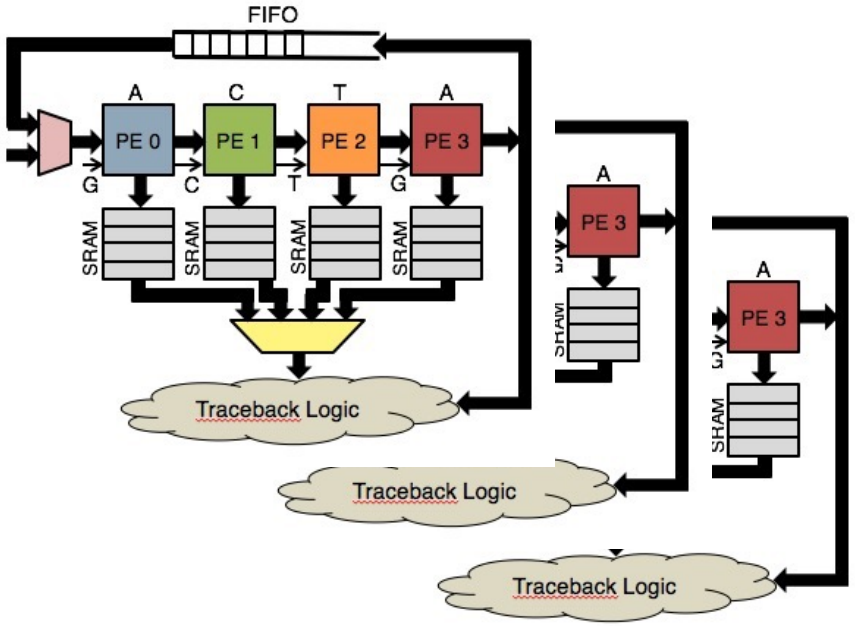


Specialization 37x speedup 26,000x efficiency

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

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

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



Specialization 37x speedup 26,000x efficiency

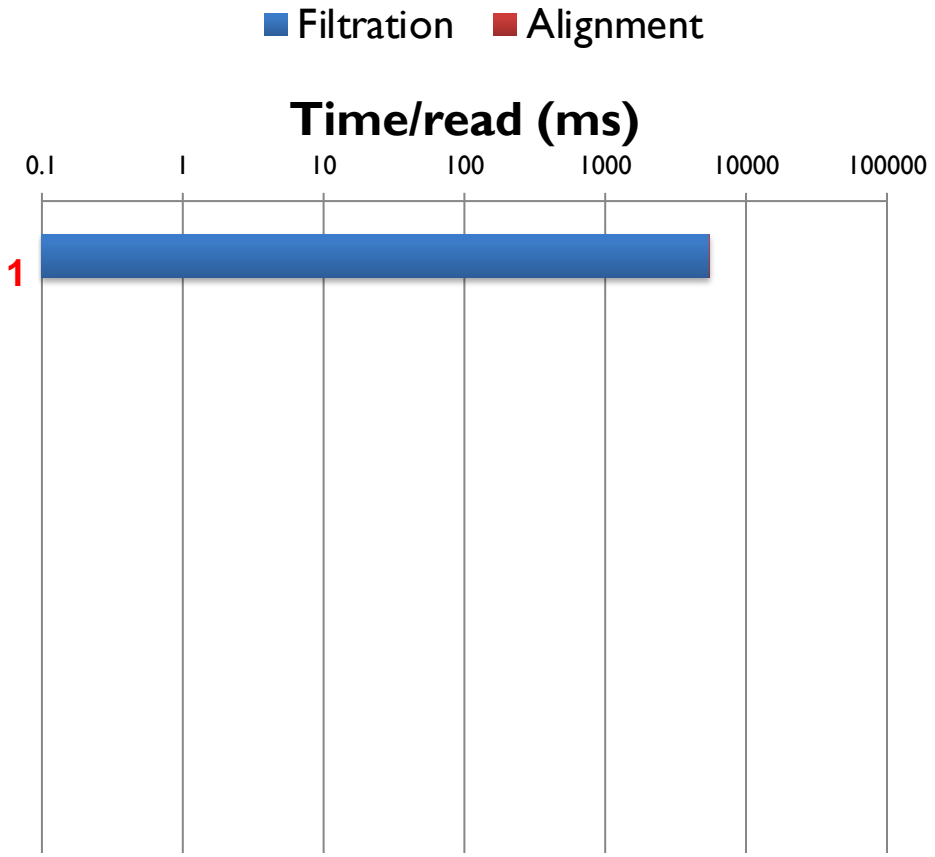
Efficiency Parallelism 64 PE arrays x 64 PEs per array, 4,096x total

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

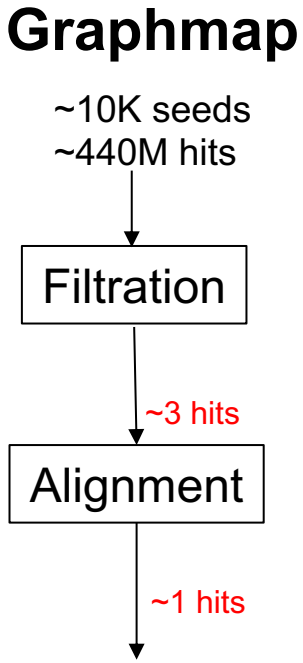
The Algorithm Often Has to Change

To Avoid Being Global Memory Limited

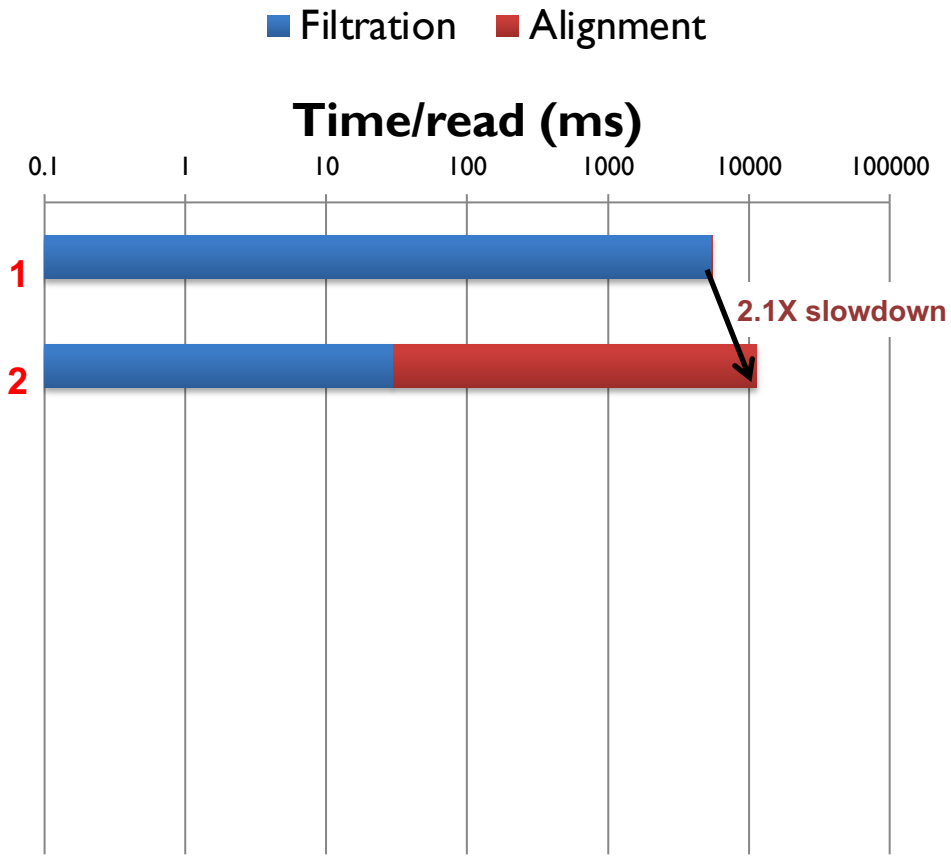
Algorithm-Architecture Co-Design for Darwin
Start with Graphmap



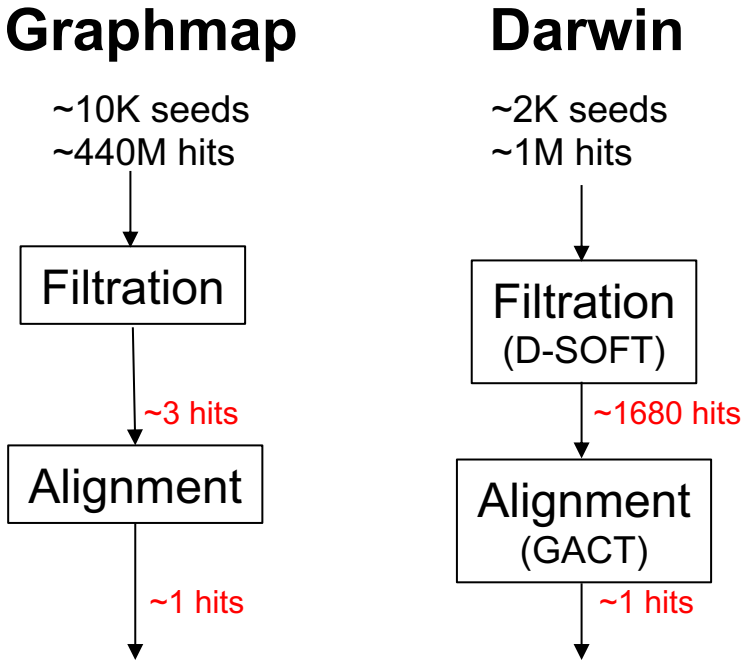
1. Graphmap (software)



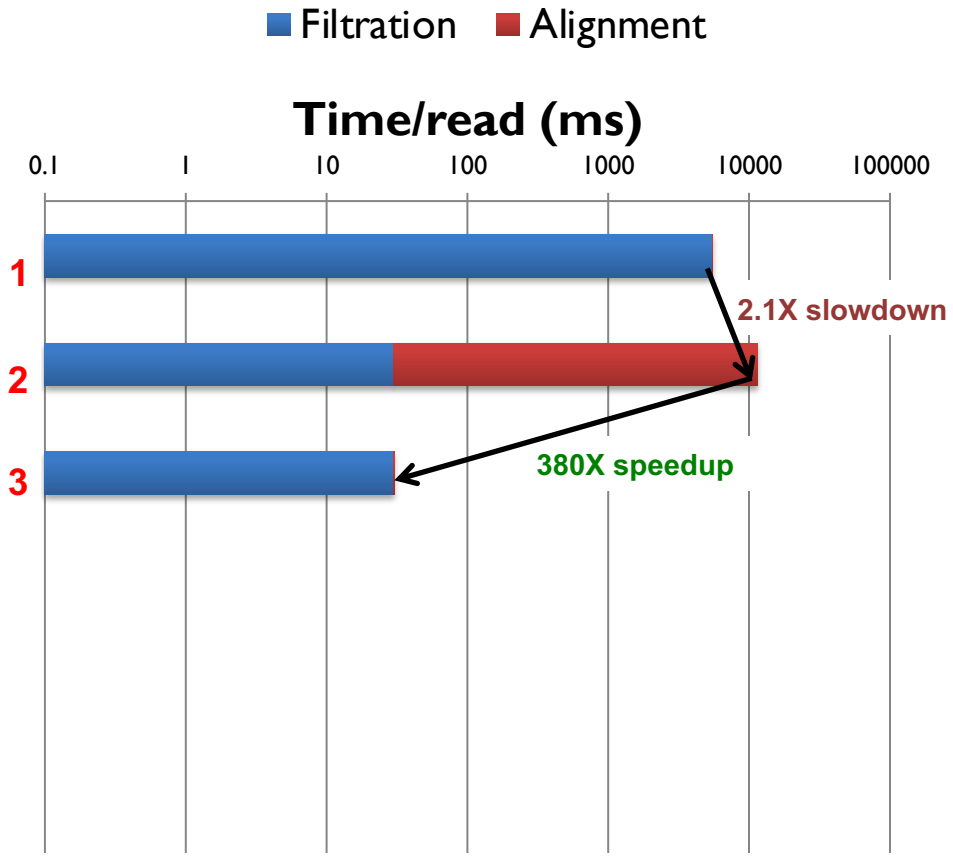
Algorithm-Architecture Co-Design for Darwin
Replace Graphmap with Hardware-Friendly Algorithms
Speed up Filtering by 100x, but 2.1x Slowdown Overall



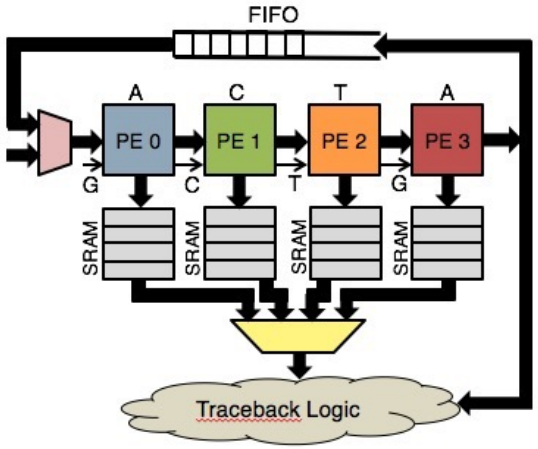
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)



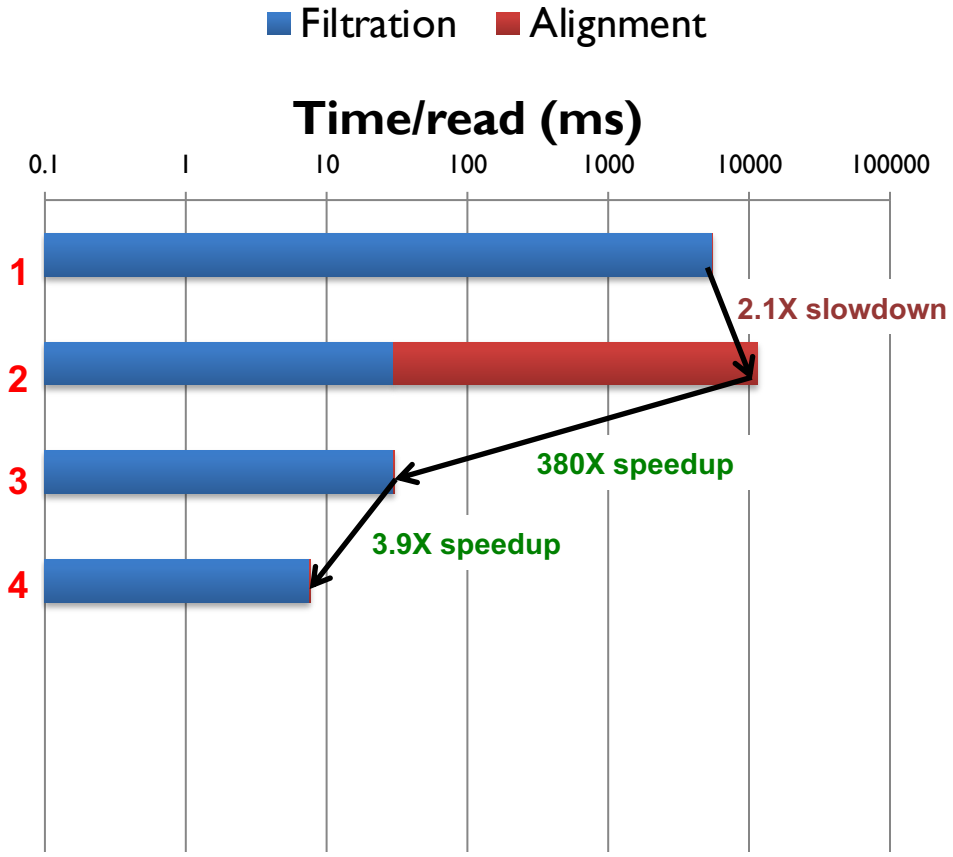
Algorithm-Hardware Co-Design for Darwin
Accelerate Alighment – 380x Speedup



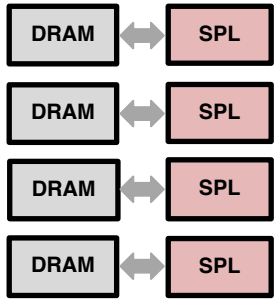
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration



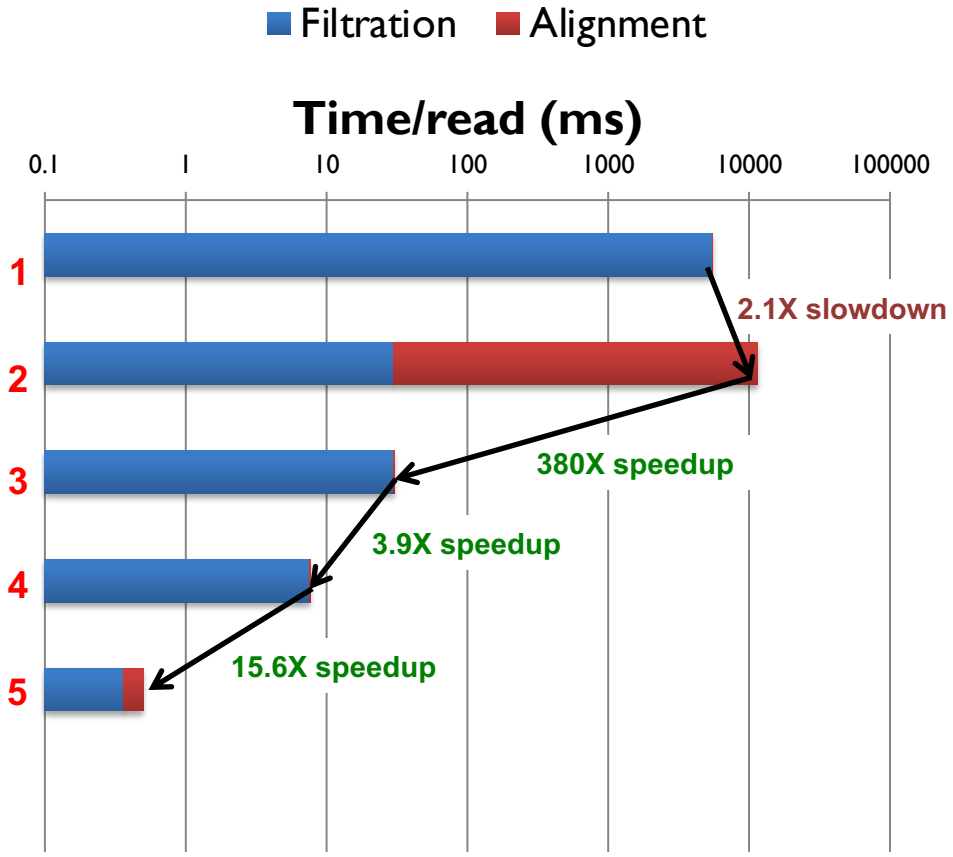
Algorithm-Hardware Co-Design for Darwin
4x Memory Parallelism – 3.9x Speeedup



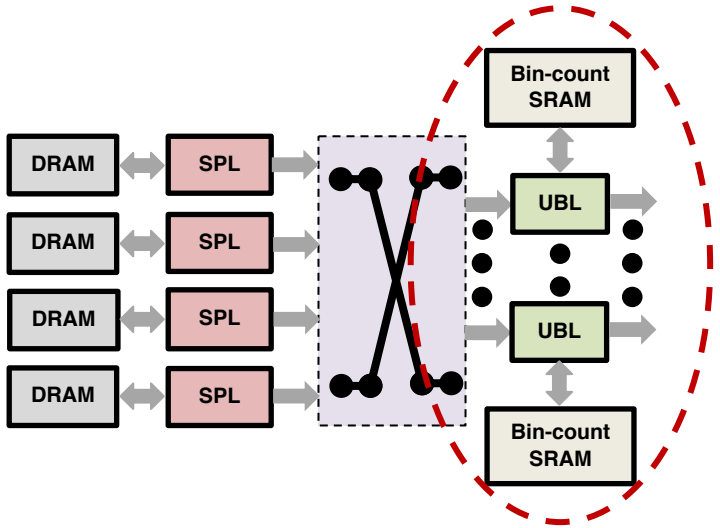
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration
4. Four DRAM channels for D-SOFT



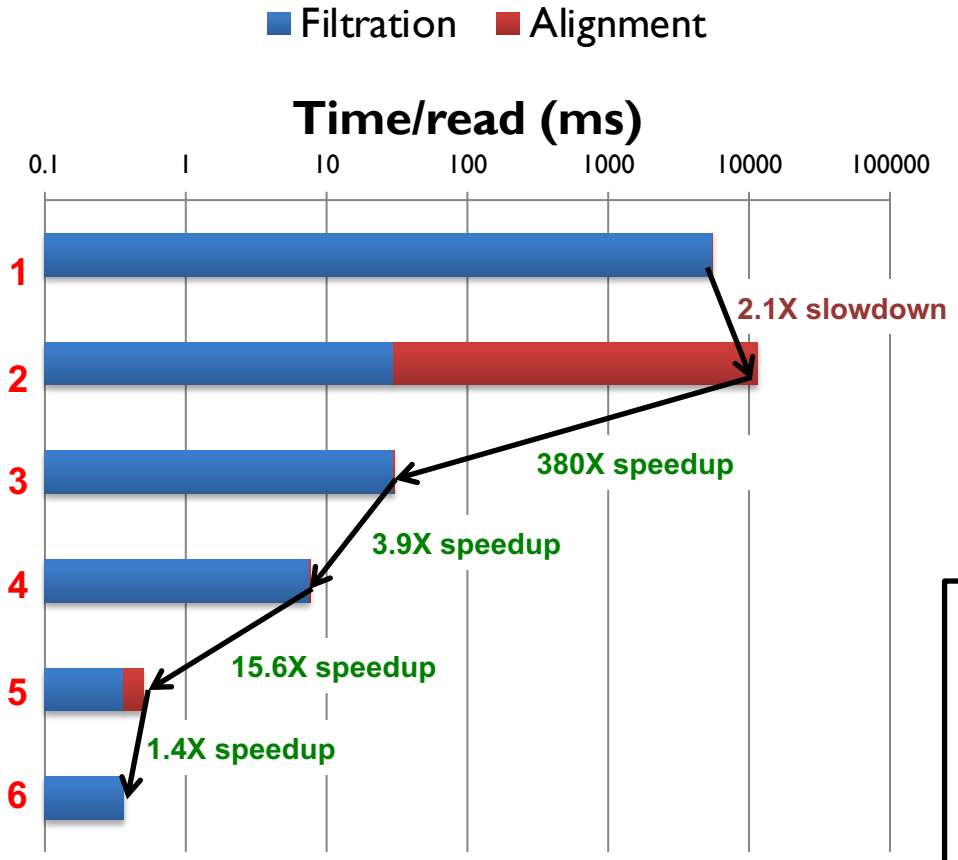
Algorithm-Hardware Co-Design for Darwin
Specialized Memory for D-Soft Bin Updates – 15.6x Speedup



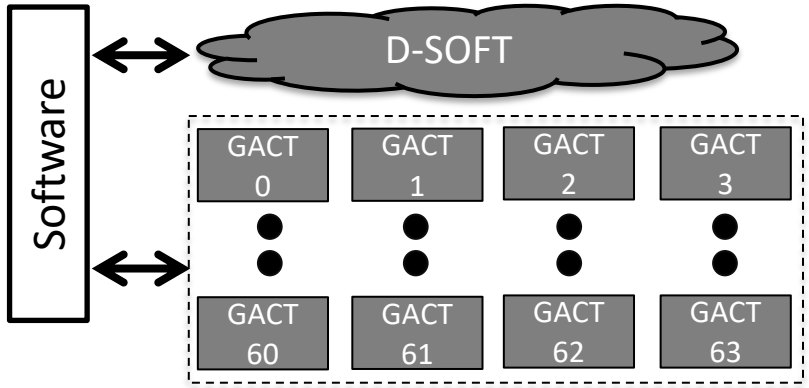
1. Graphmap (software)
2. Replace by D-SOFT and GACT (software)
3. GACT hardware-acceleration
4. Four DRAM channels for D-SOFT
5. Move bin updates in D-SOFT to SRAM (ASIC)



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



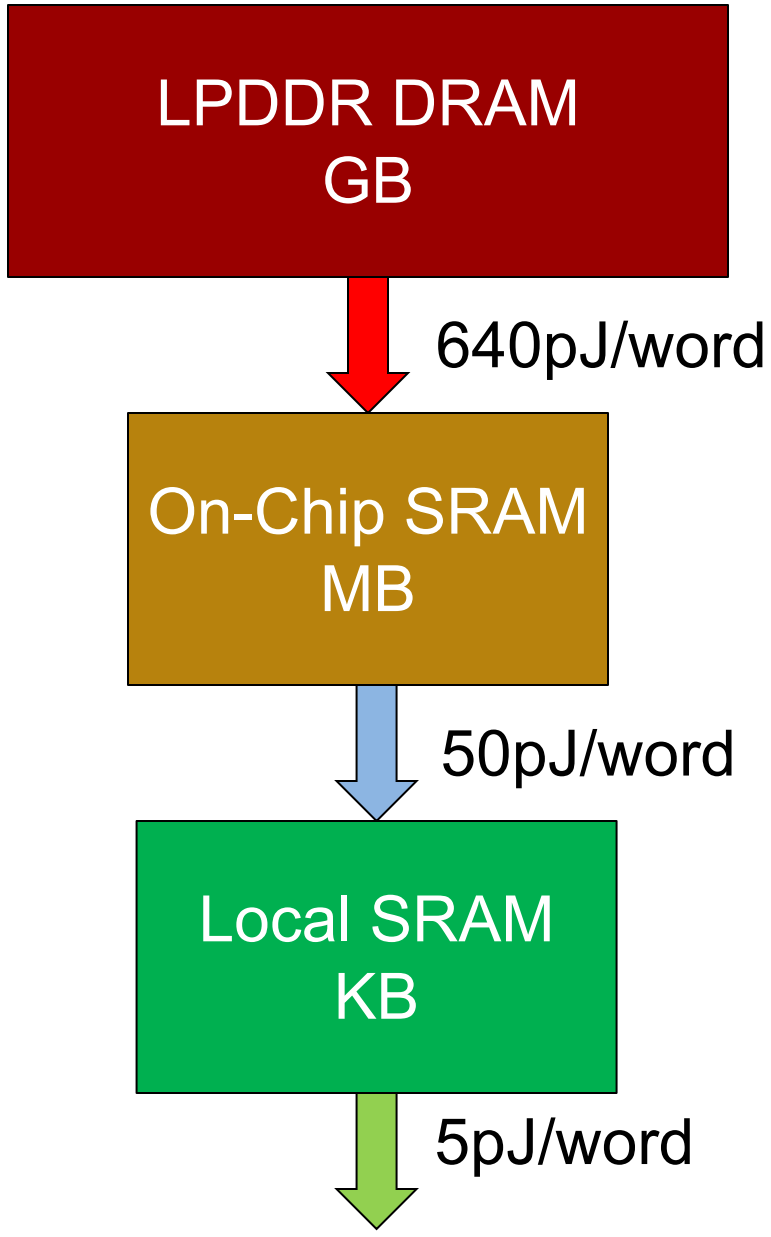
Memory Dominates

Memory dominates power and area

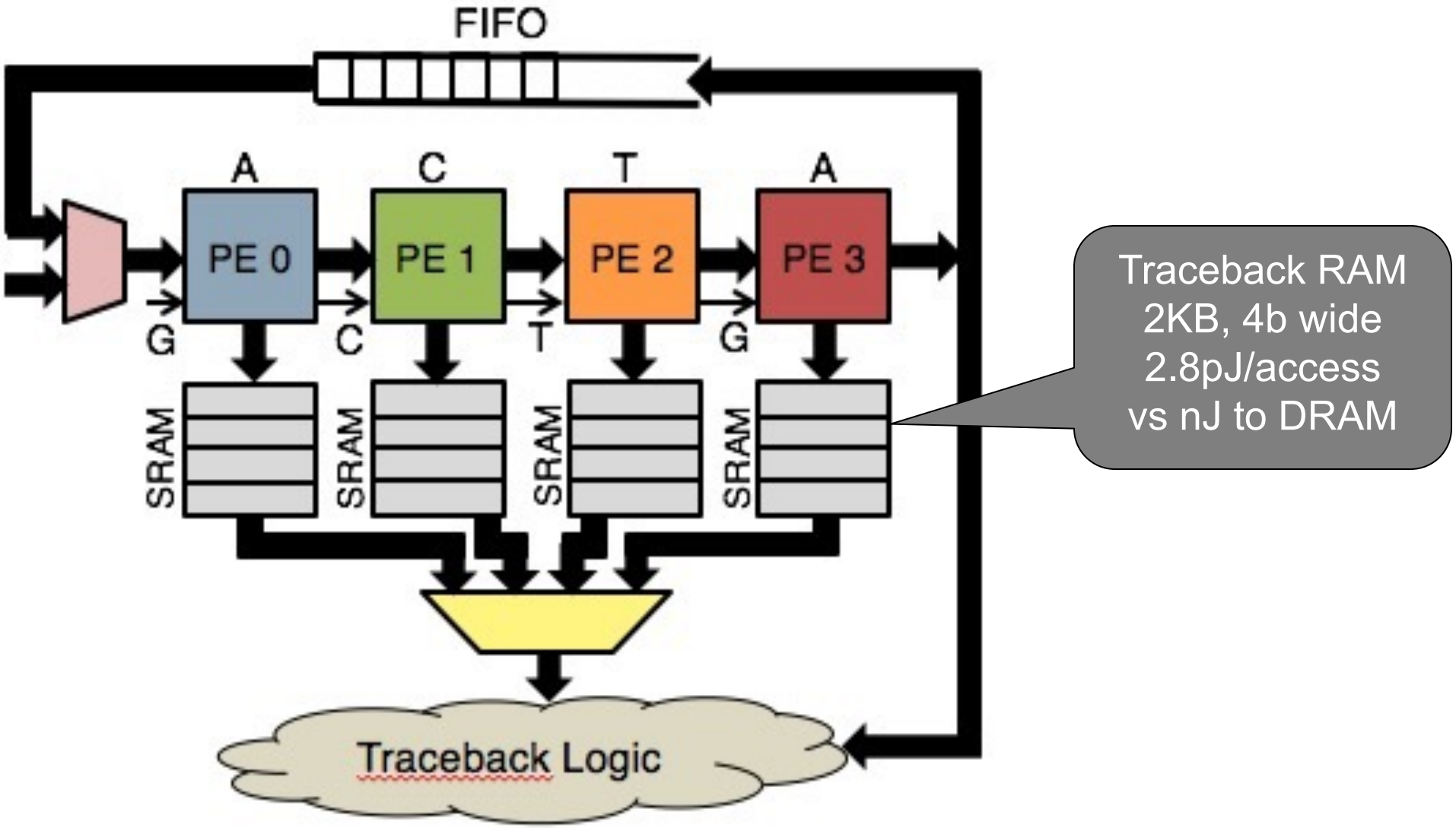
Memory Dominates

	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%

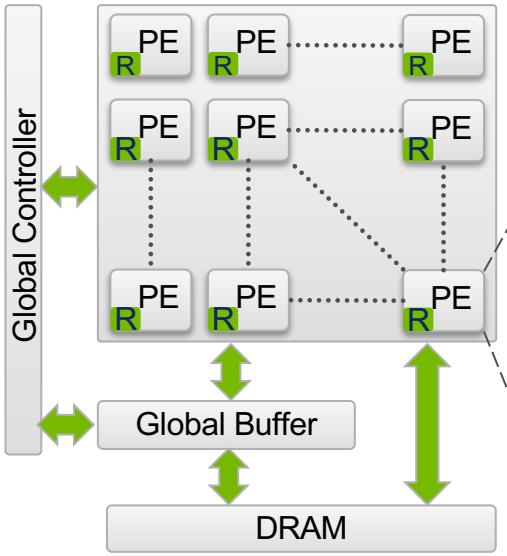
Communication is Expensive, Be Small, Be Local



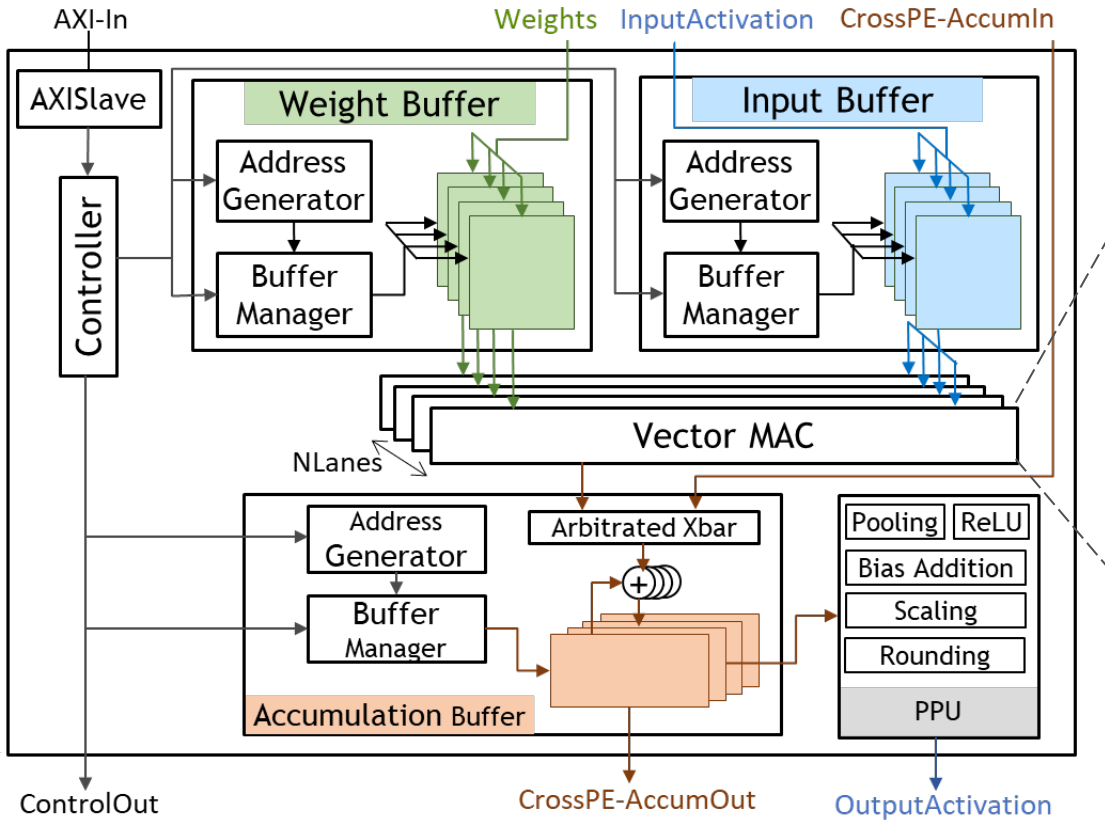
Small, Local Memories



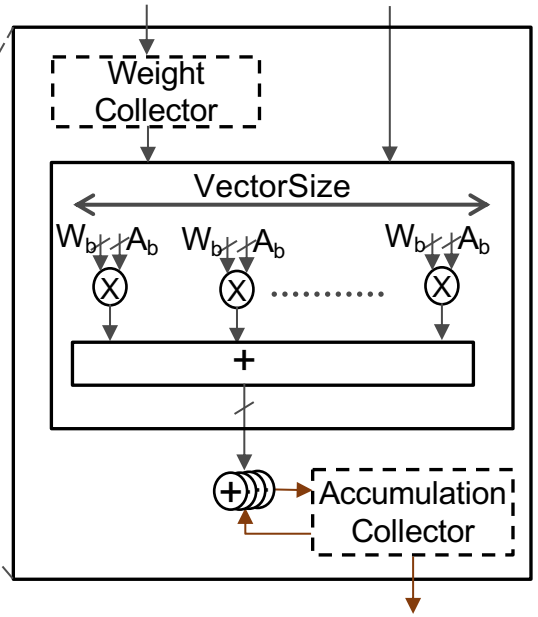
MAGNET



MAGNet System



Processing Element (PE)



Vector MAC unit

Venkatesan, et al. "MAGNet: A Modular Accelerator Generator for Neural Networks." *ICCAD*. 2019.

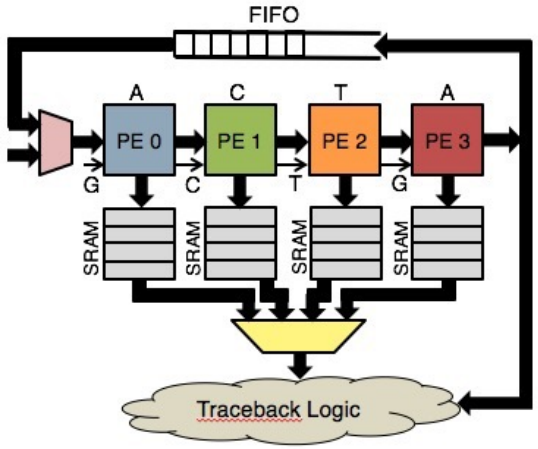
Algorithms must be memory optimized

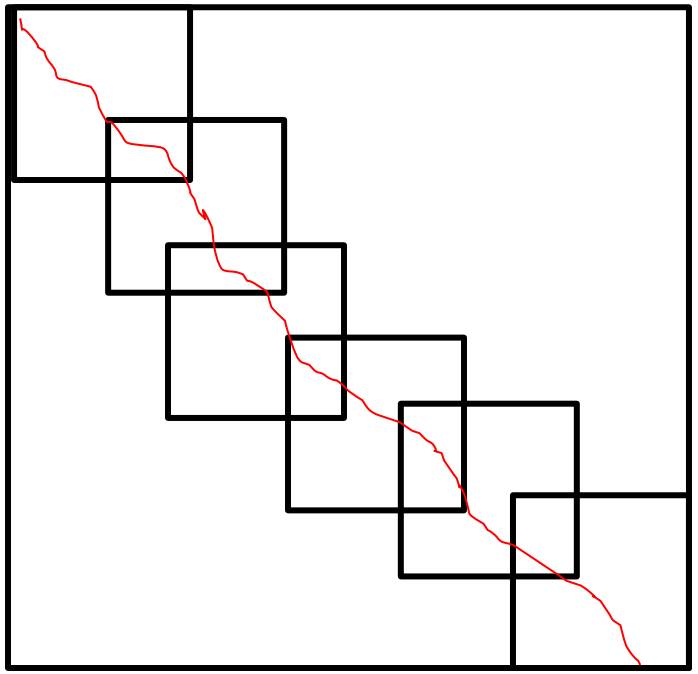
Minimize global memory accesses

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)





Complex Memory Ops

Not just Load/Store

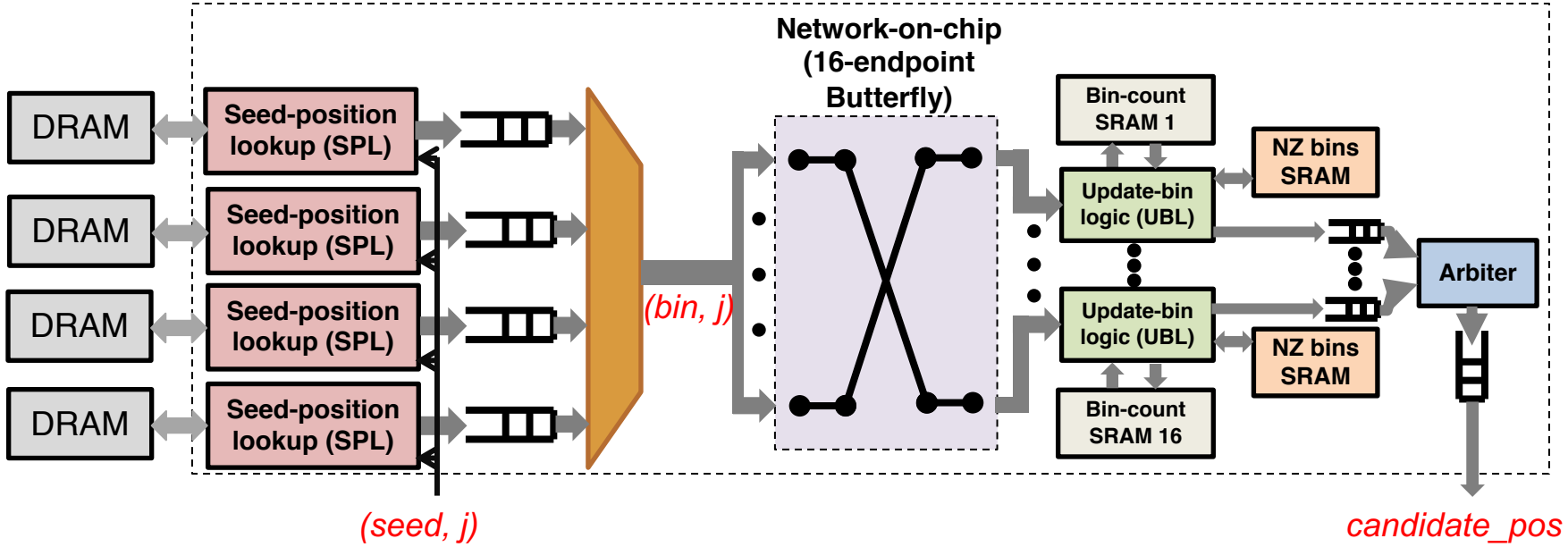
Hash, Atomic Functions, Side Effects

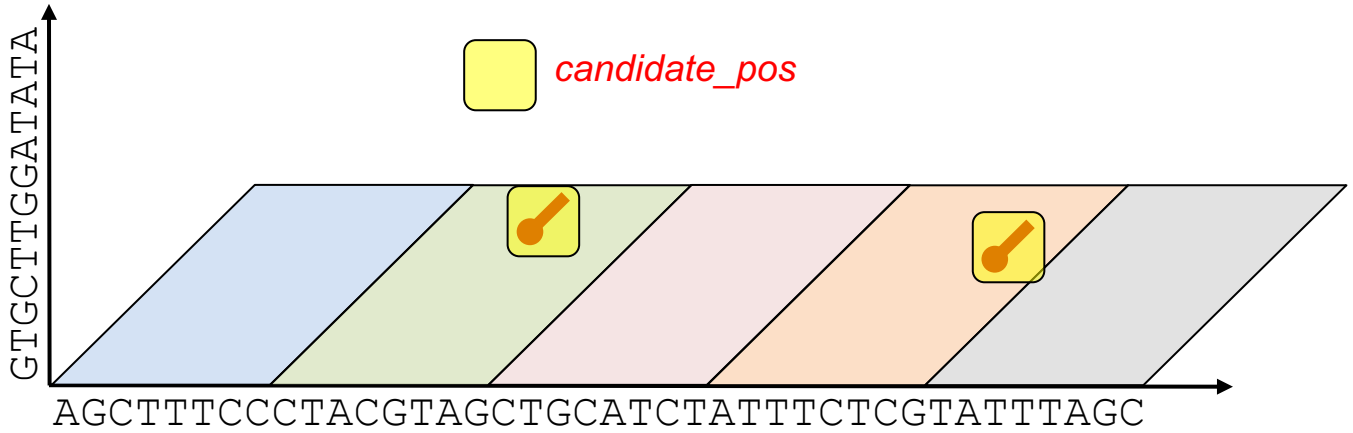
Make the Most Use of One Communication

Message-Driven Processing
One Communication, Many Operations

One traversal of network

- Access hash table
- Increment bin (RMW)
- If it was zero, append to NZ bins
- If over threshold, append to output queue

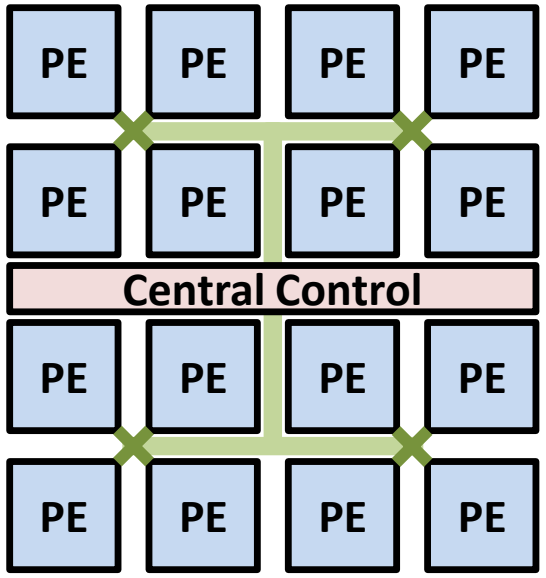




Sparsity and Compression

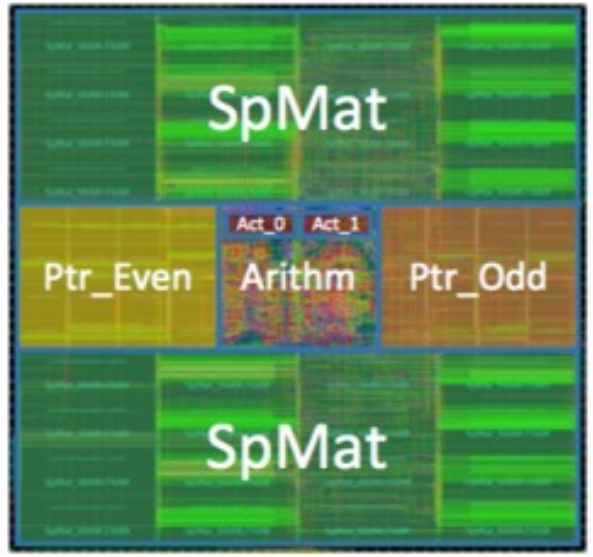
multiply

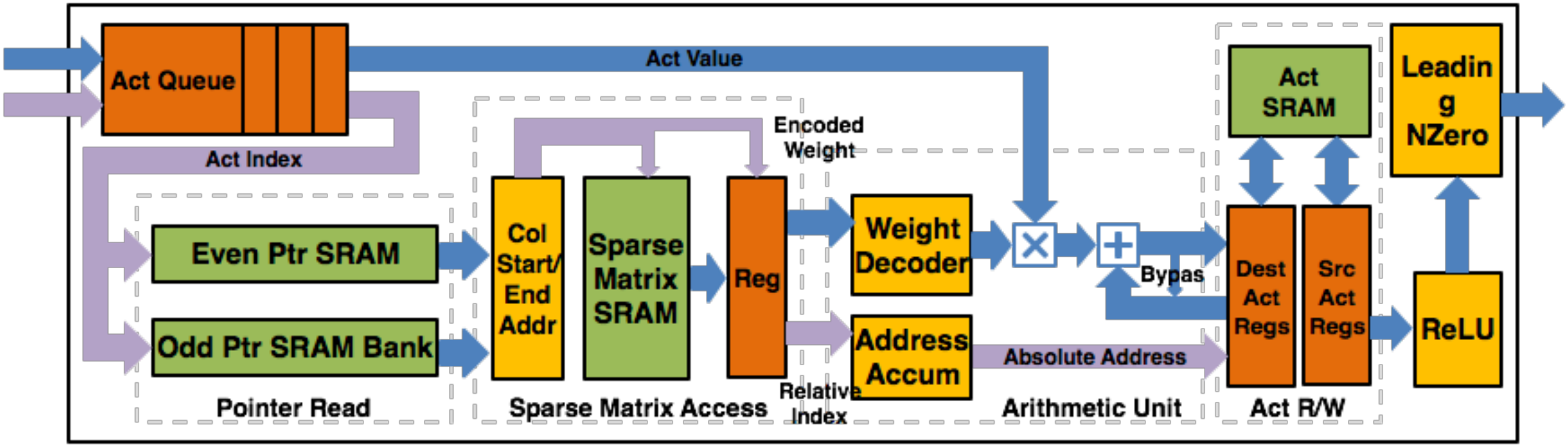
Memory Bandwidth and Capacity



EIE HARDWARE

- Traverse CSC Sparse matrix
- Decode scalar quantization
- Little overhead





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

Platforms for Acceleration

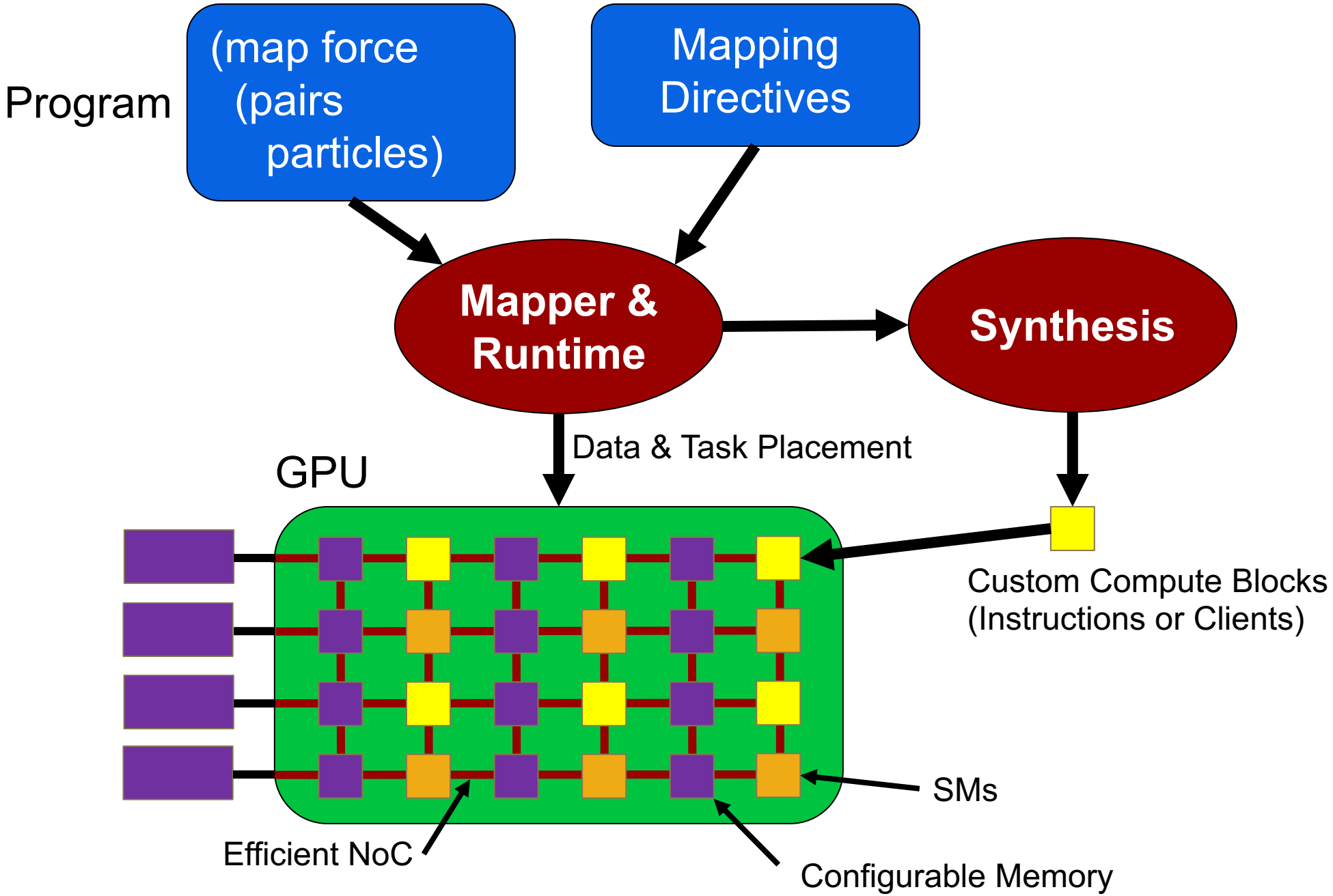
GPUs Provide:

- 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

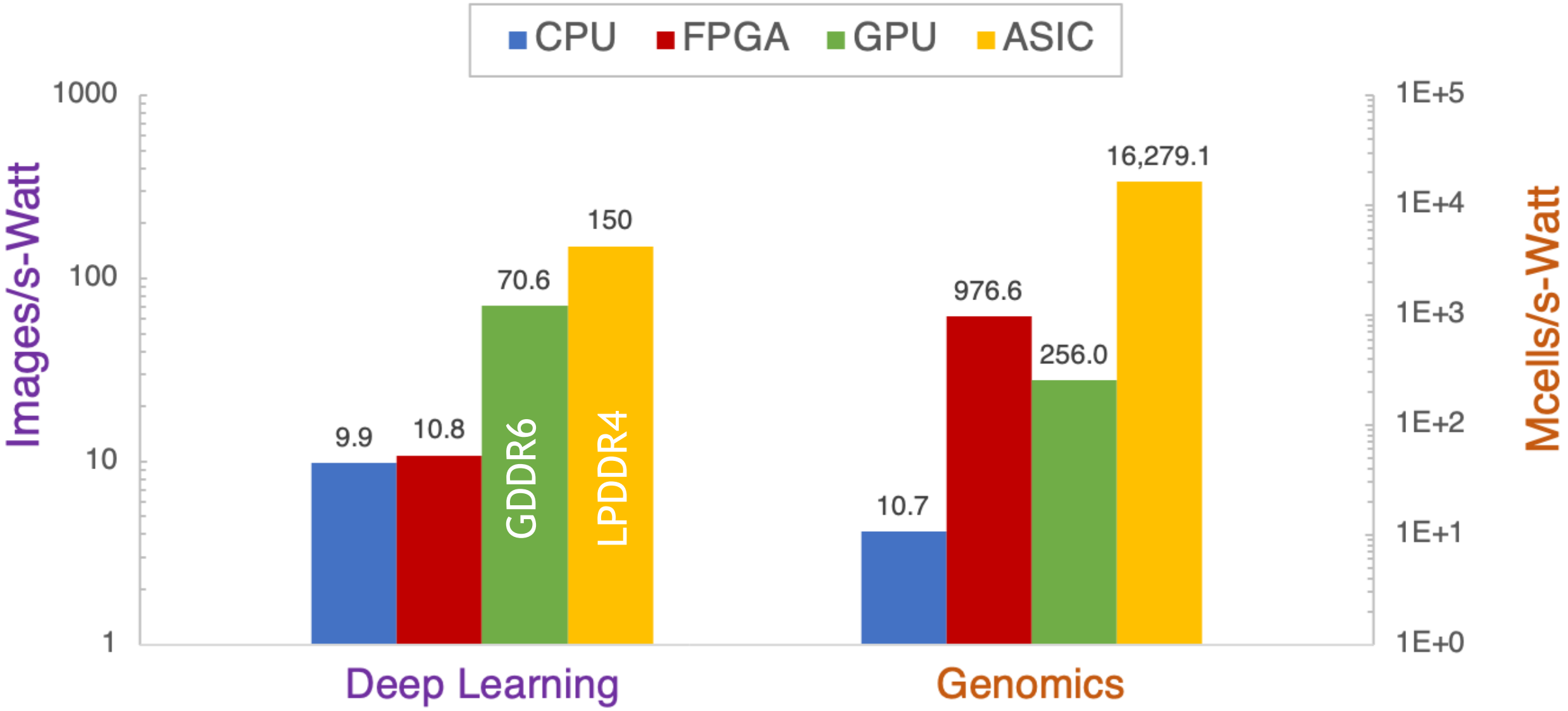
Specialized Instructions Amortize Overhead

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%

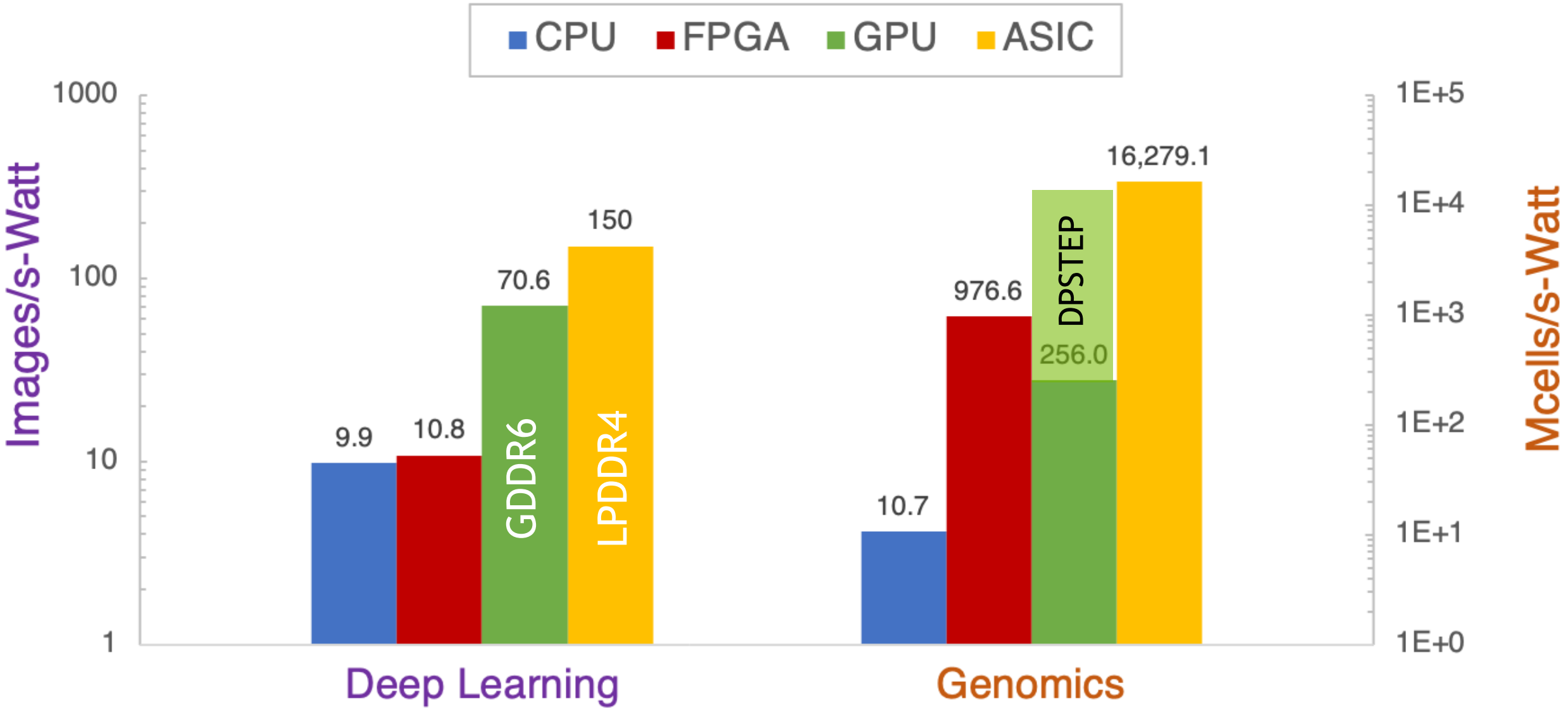
*Overhead is instruction fetch, decode, and operand fetch – 30pJ
**Energy numbers from 45nm process


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

Implementation Alternatives



Implementation Alternatives



MULTI-CHIP MODULES (MCMS)

Accelerator Design as Programming

With Hardware Costs

DSA Design is Programming

With a Hardware Cost Model

Algorithm

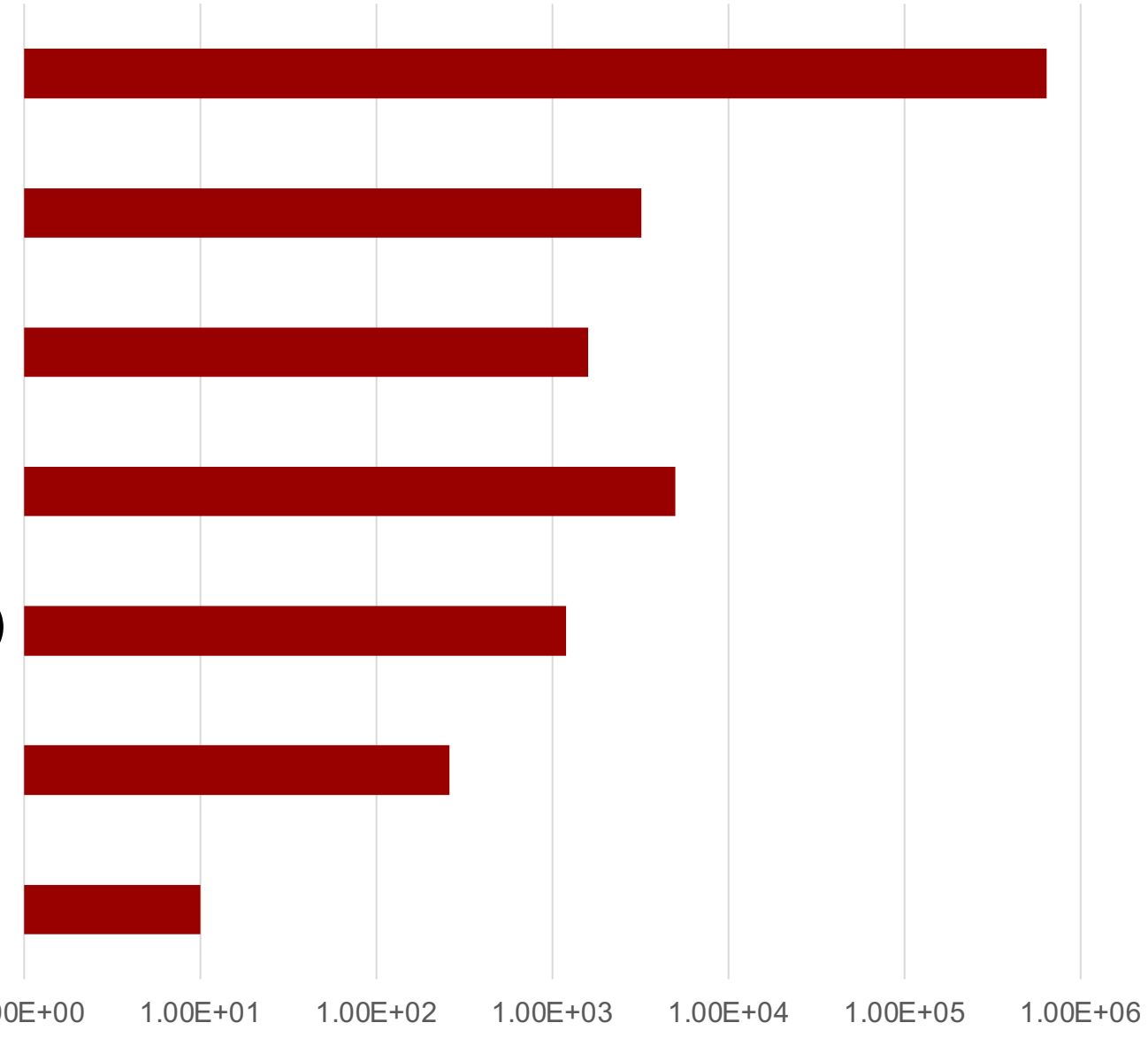
```
tb ← GACT(r, q)
input  : r[TS], q[TS]
output : tb[TS,TS]
for i = 0..TS-1 do
    for j = 0..TS-1 do
        in (i,j) ← Max (h (i,j-1) - O, in (i, j-1) - E)
        del (i,j) ← Max (h(i-1,j) - O, del (i-i,j) - E)
        h (i,j) ← Max (0, in(i,j), del (i,j), h (i-1, j-1) + W
          (r[i],q[j]))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

Hardware Costs

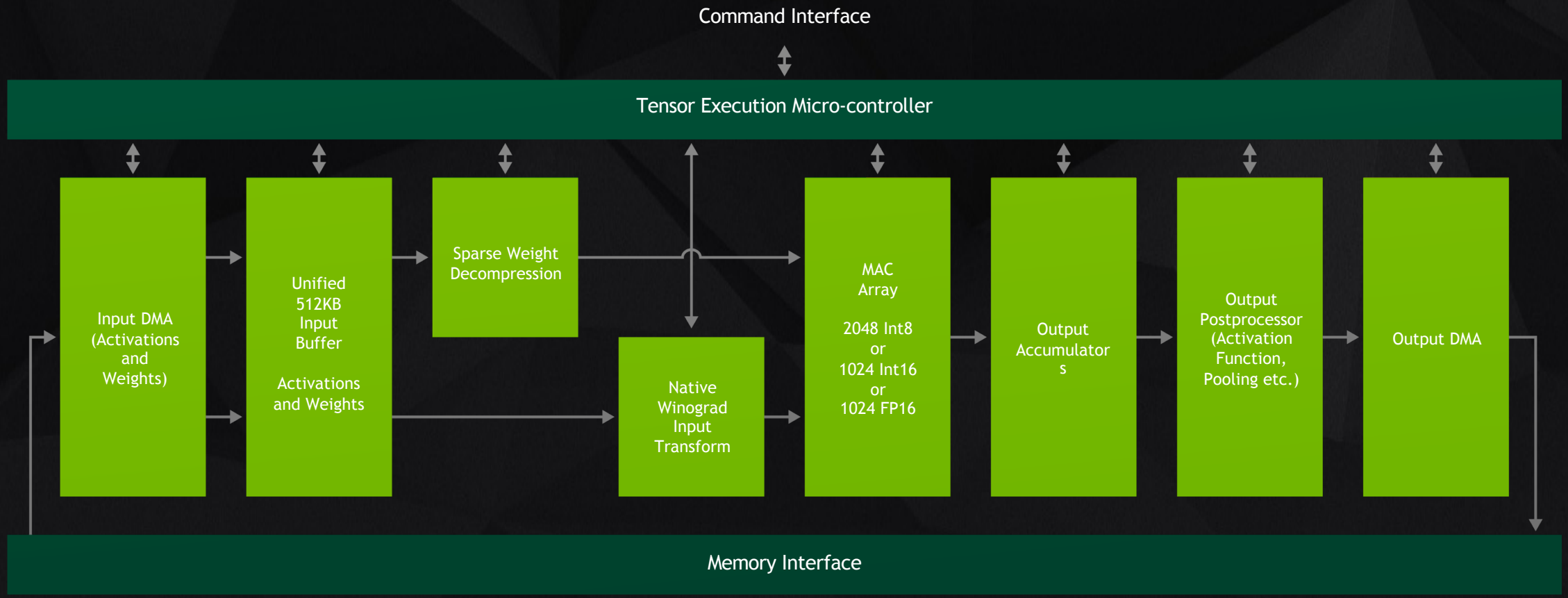
Global Memory	640pJ/word
On Chip Comm	3.2pJ/word-mm
Local Memory	1.6pJ/word
Arithmetic	5pJ (FP64 FMA)
	1.2pJ (FP32 FMA)
	260fJ (int16 mul)
	10fJ (int8 add)



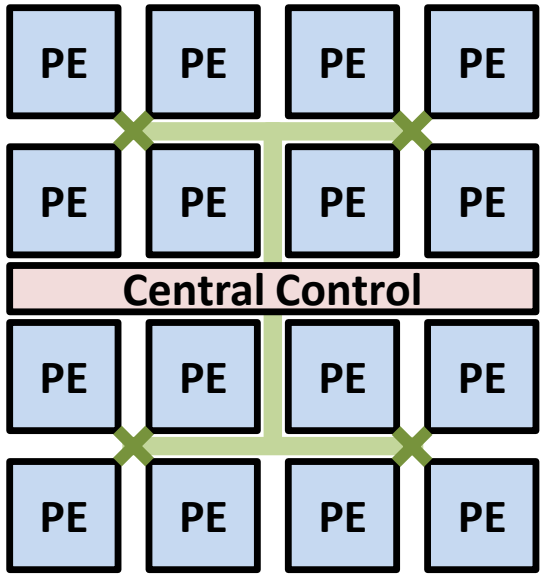
*Energy numbers for 14nm
**A word is 32 bits

Accelerating Deep Learning

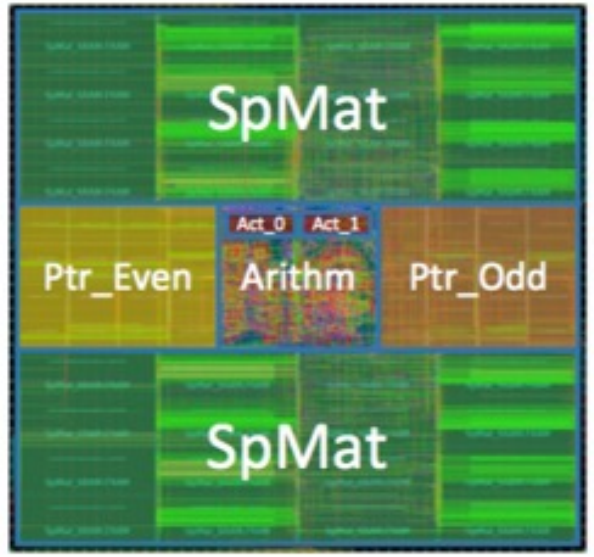
NVIDIA DLA

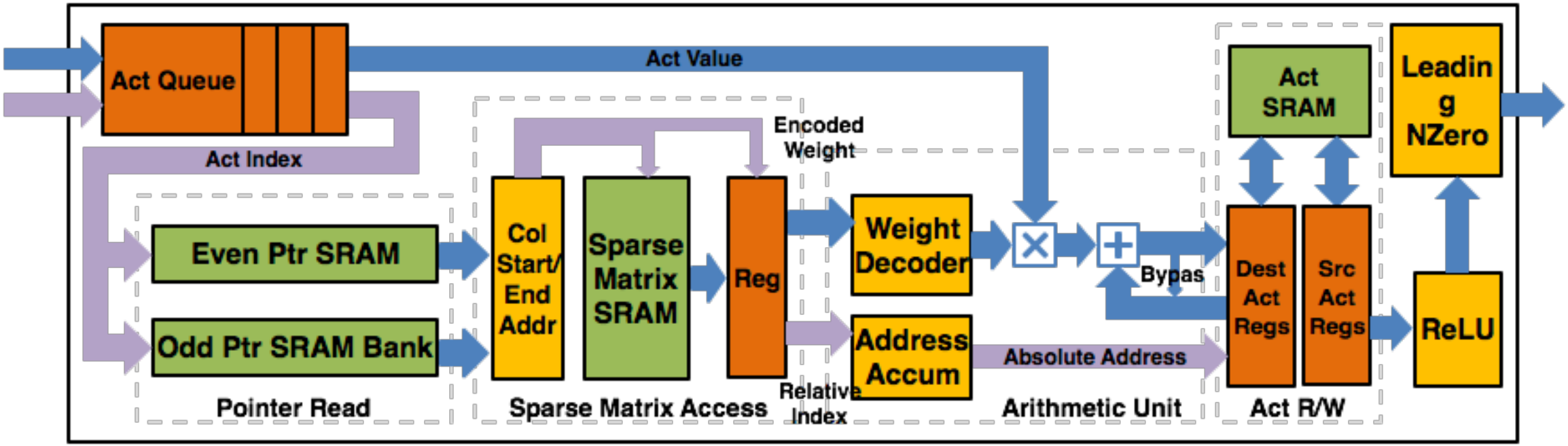

Command Interface
Tensor Execution Micro-controller
Input DMA (Activations and Weights)
Unified 512KB Input Buffer
Activations and Weights
Sparse Weight Decompression
Native Winograd Input Transform
MAC Array
2048 Int8 or 1024 Int16 or 1024 FP16
Output Accumulators
Output Postprocessor (Activation Function, Pooling etc.)
Output DMA
Memory Interface

EIE Hardware



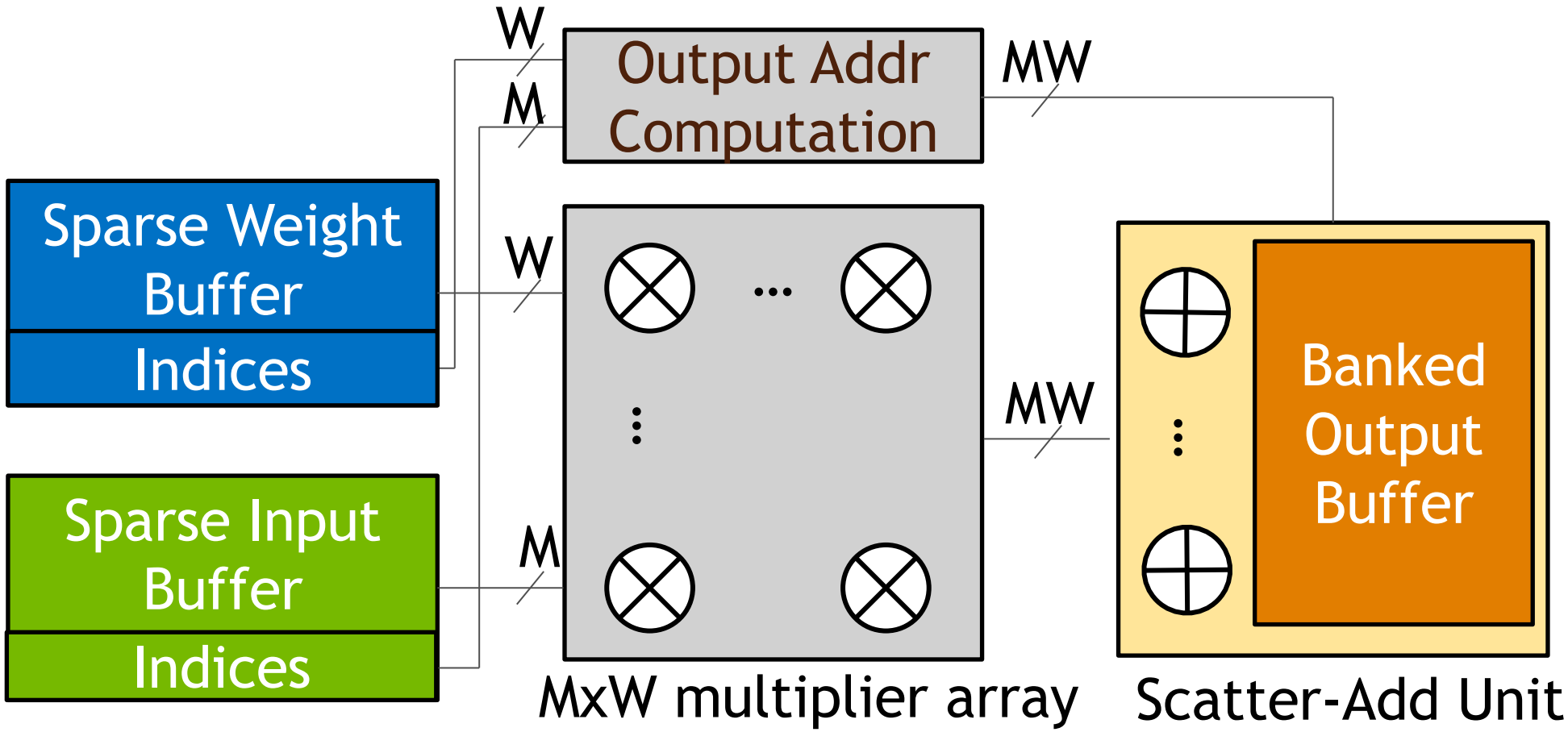
- Traverse CSC Sparse matrix
- Decode scalar quantization
- Little overhead





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

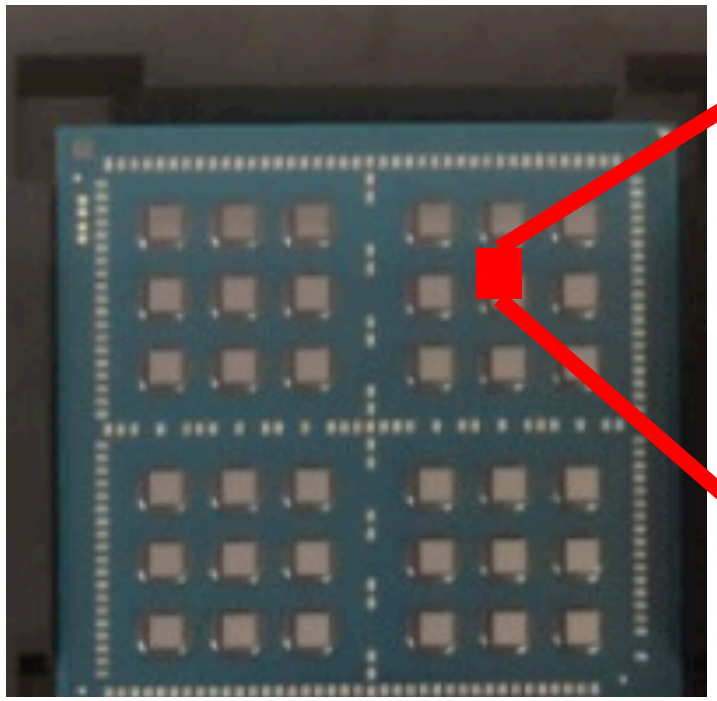
Sparse Convolution Engine



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

RC18: A 36-die MCM Architecture

Connected via Ground-Referencing Signaling (GRS)



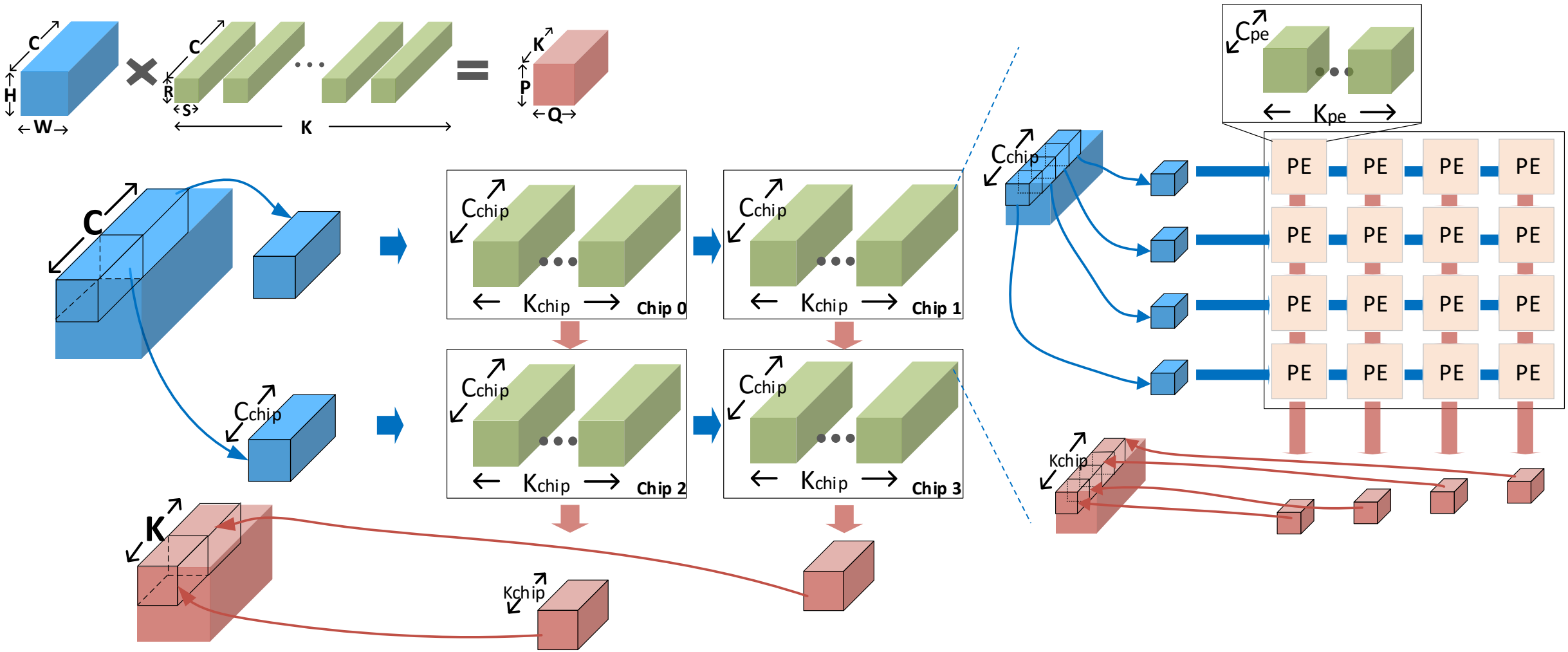
36-die

- GRS: 4 data bumps + 1 clock bump, **25Gbps/pin, 1.6pJ/bit*.**
- **8 GRS links per die** connected in mesh (NESW TX/RX).
- **100GB/s** per chiplet.
- **105fJ/op**

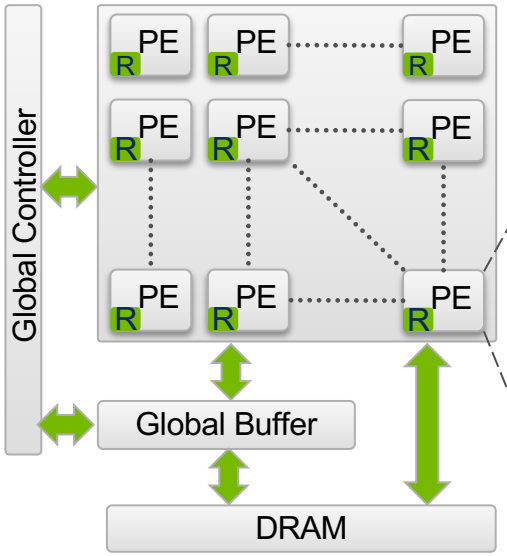
*1.2pJ/bit at 8b width, 1pJ/bit at 16b width

RC18: Partitioning of Weights and Activations

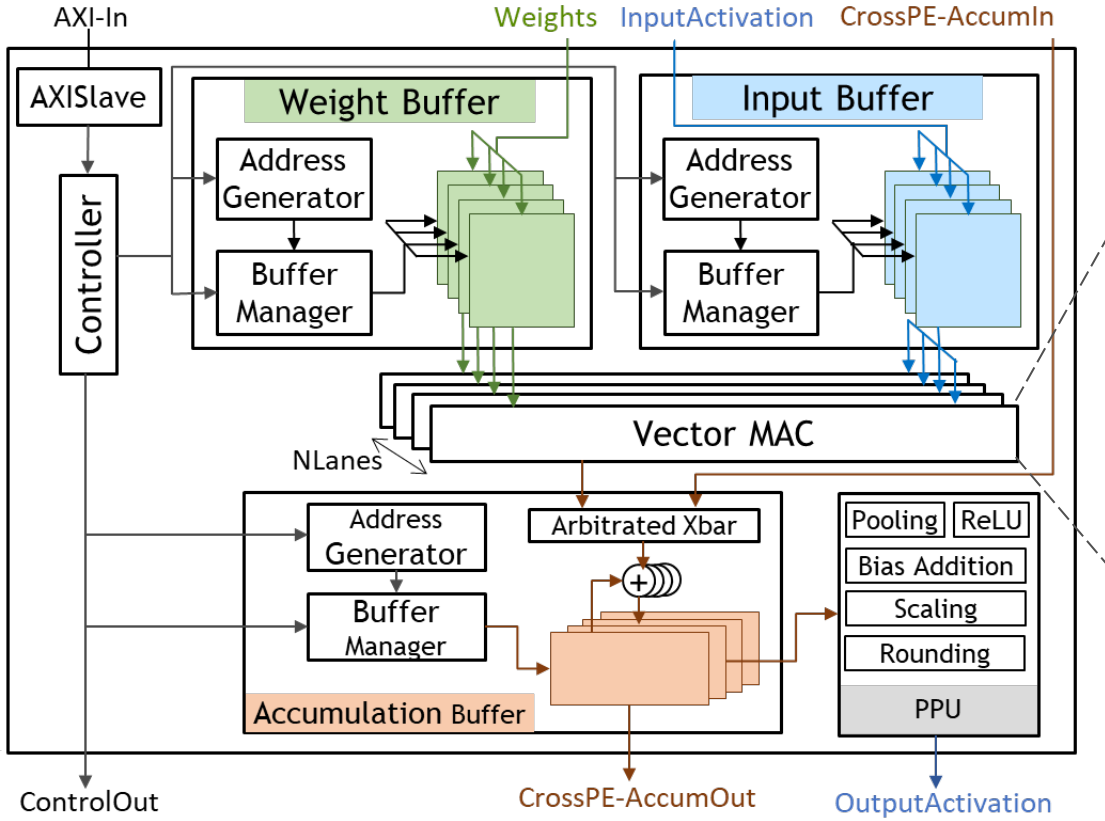
Scaling DL Inference across NoP and NoC



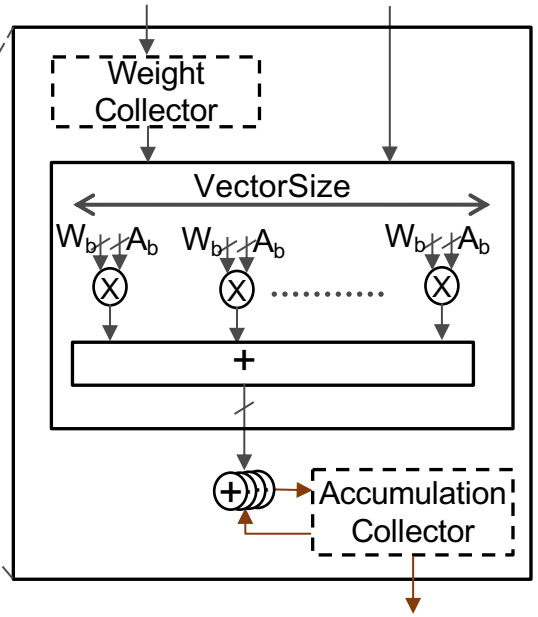
MAGNET



MAGNet System



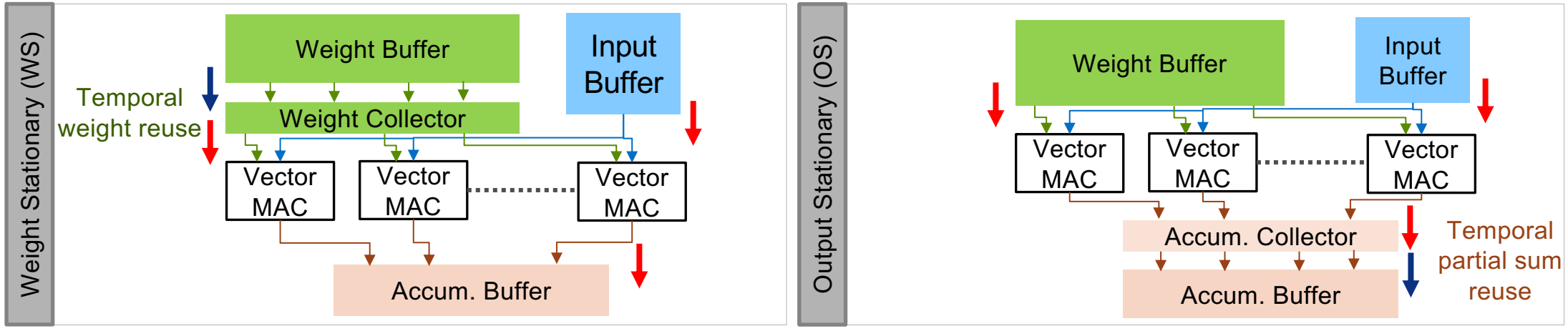
Processing Element (PE)



Vector MAC unit

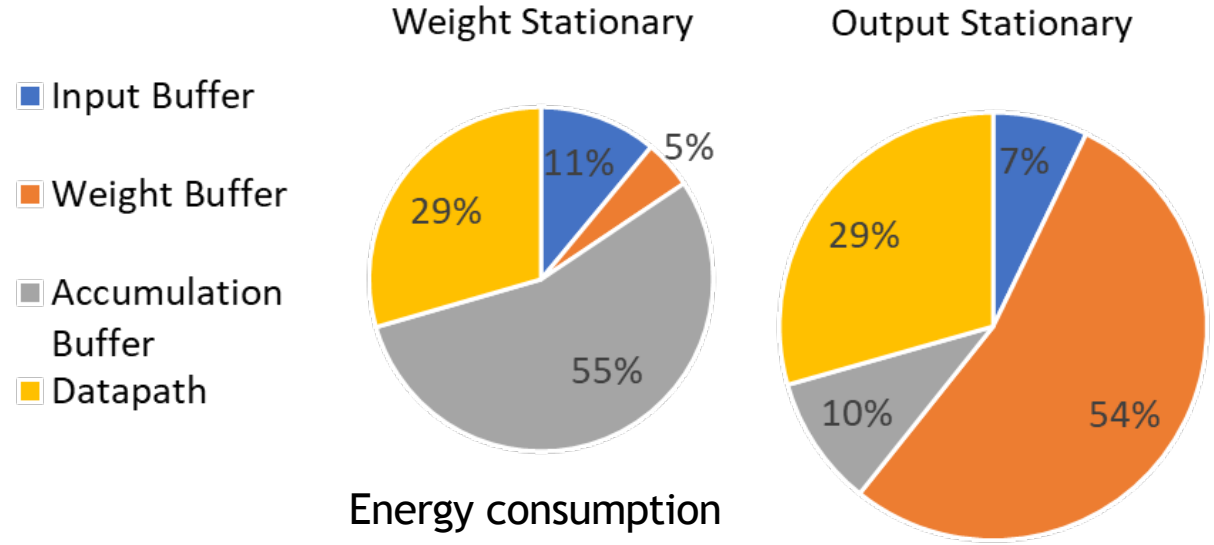
Venkatesan, et al. "MAGNet: A Modular Accelerator Generator for Neural Networks." *ICCAD*. 2019.

DataFlow Options



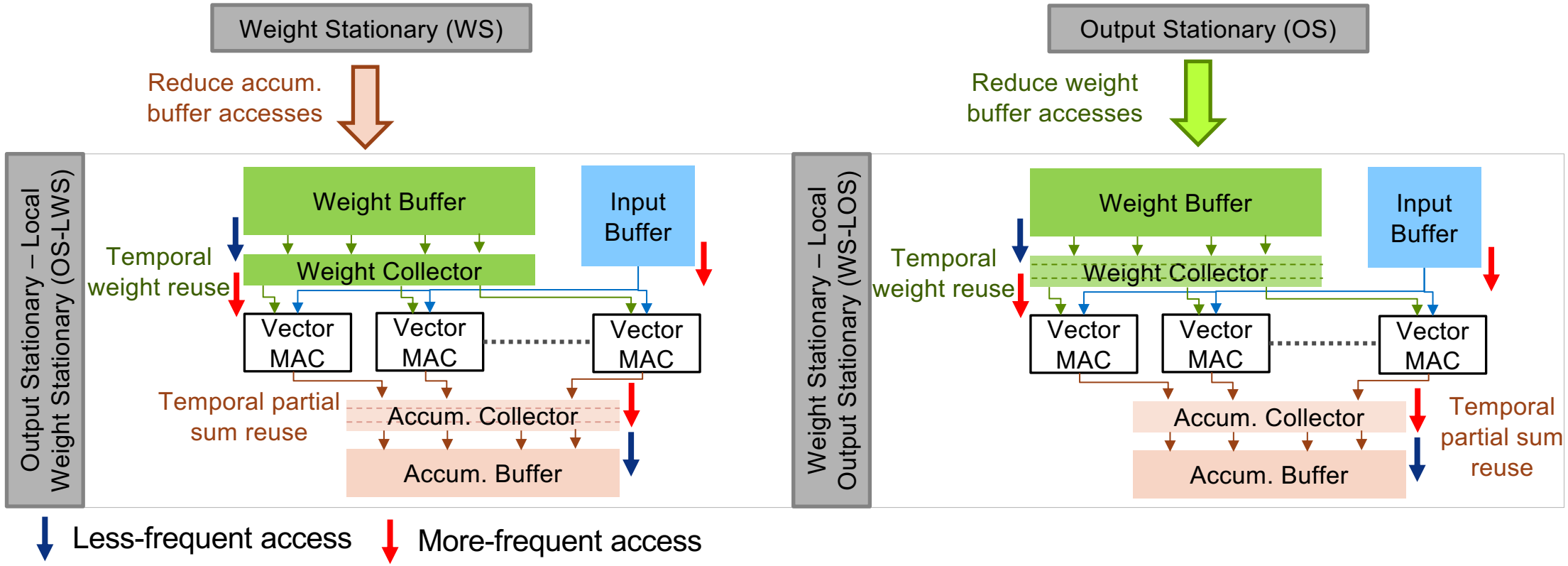
Less-frequent access

More-frequent access



Energy consumption

Multi-Level DataFlows

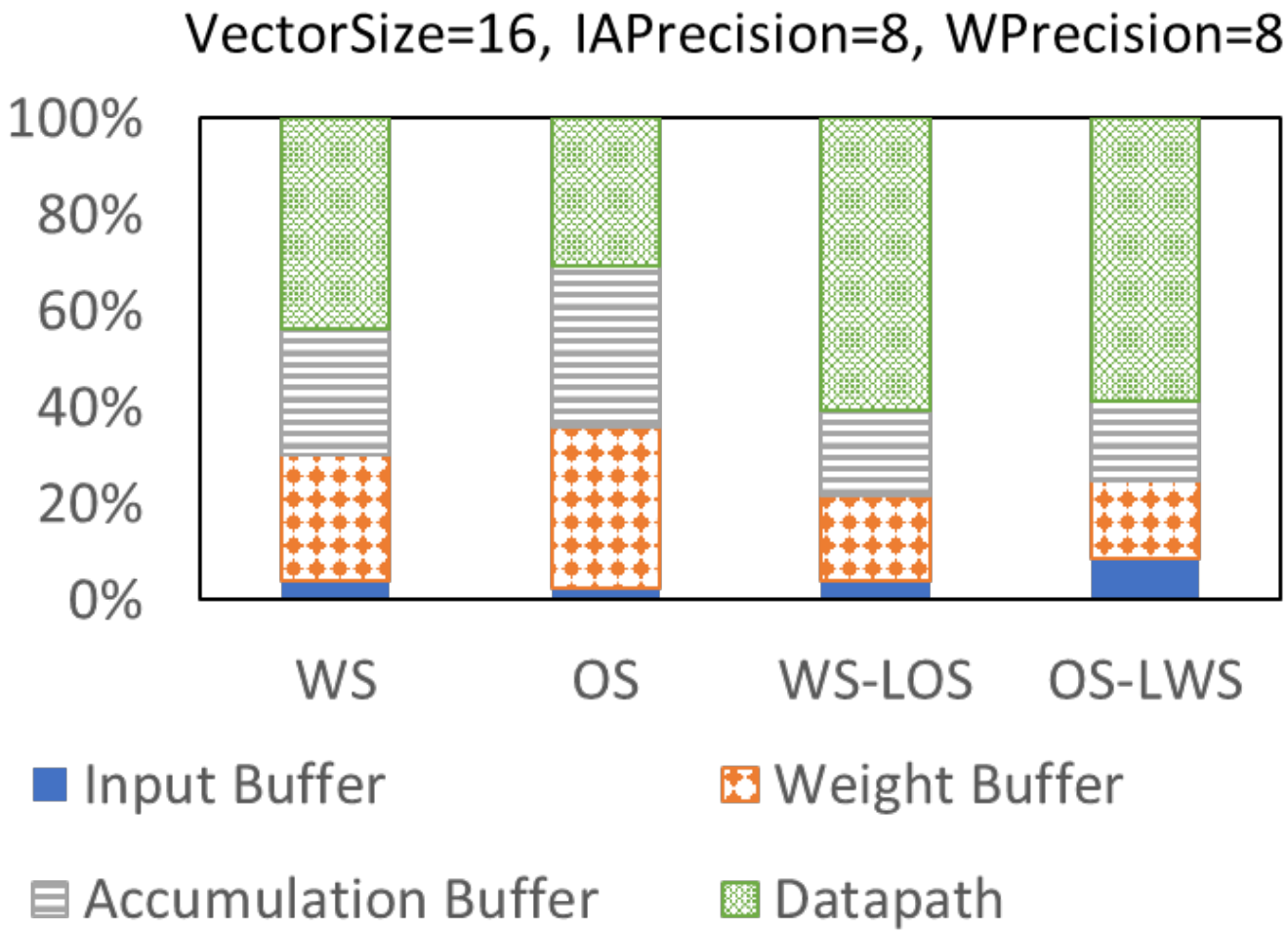


Multi-Level DataFlows

70 fJ/MAC

35 fJ/OP

29 TOPS/W



Conclusion

Summary

- **Moore's Law is over,** but we must **continue scaling perf/W**
- **Accelerators** are the future
  - **Specialization, Customized Memories** -> Efficiency
  - **Parallelism ->** Speedup
  - **Co-Design:** The algorithm has to change
  - **Memory dominates**
- **GPUs** as accelerator platforms
  - **GPUs** – efficient memory, communication and control
  - **Custom blocks** – instructions or clients
- DSA design is **programming** – with a hardware cost model