Domain Specific Accelerators

Workshop Agenda

- Lecture 1: Domain Specific Architectures
- Lecture 2: Kernel computation
- Lecture 3: Data-flow techniques
- Lecture 4: DNN accelerators architectures

Lecture 1 -- Agenda

- A bit of history
- Inefficiency in GP architectures
- Domain Specific Architectures

Session 1

- Source of acceleration
- Cost models
- Communication issues

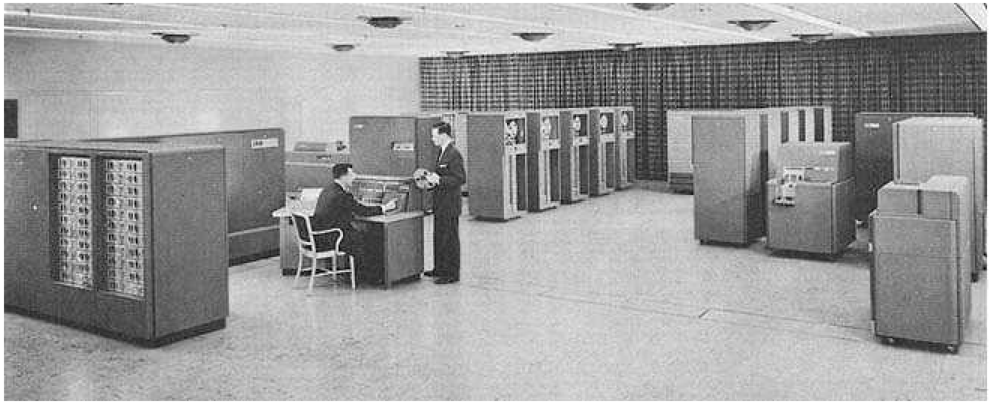
Session 2

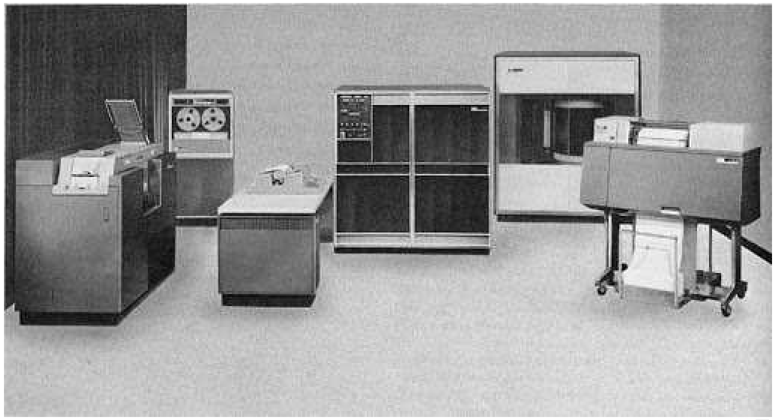
“Those who cannot remember the past are condemned to repeat it.”

George Santayana, 1905

IBM Compatibility Problem

- By early 1960's, IBM had four incompatible lines of computers





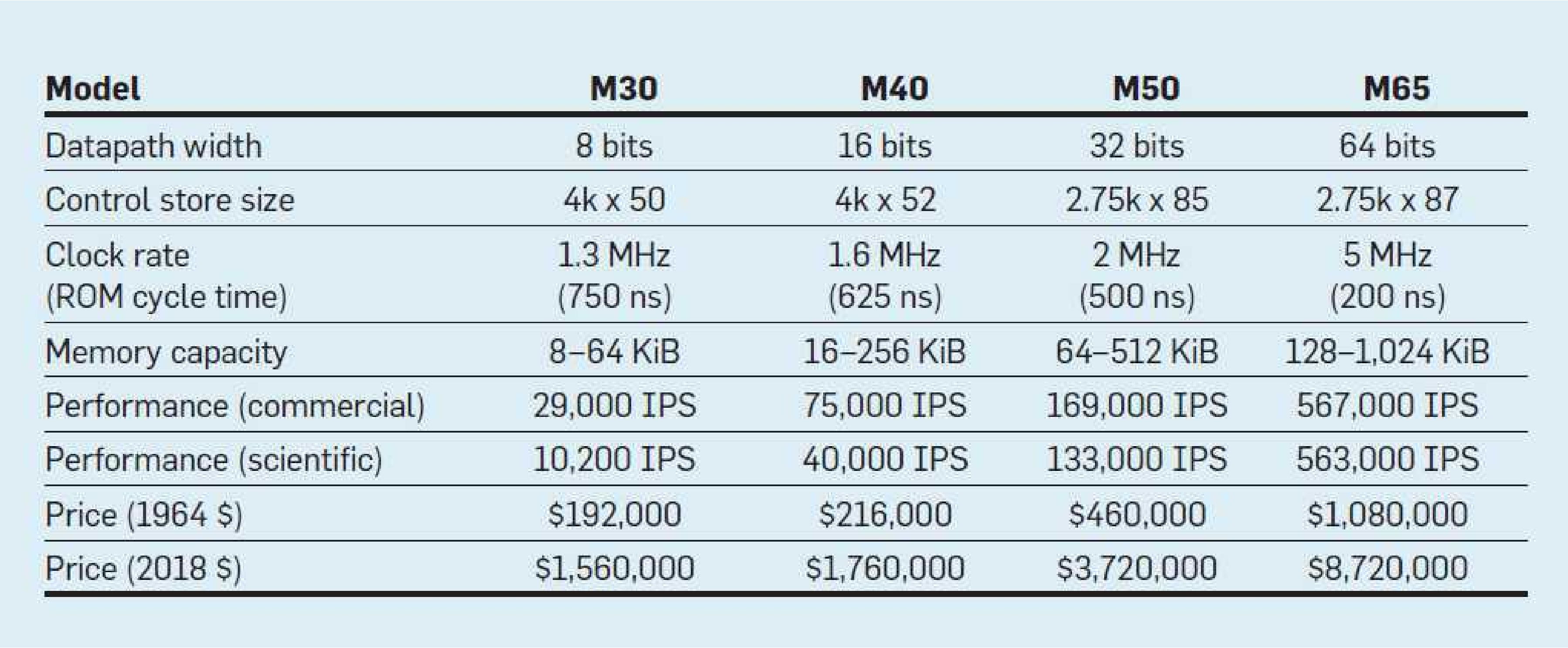
Unifying the ISA

- *How computers as inexpensive as those with 8-bit data paths and as fast as those with 64-bit data paths could share a single ISA?*
  - Datapath: not a big issue!
  - Control: the greatest challenge
- Microprogramming

Microprogramming

- ISA interpreter
  - Instruction executed by several microinstructions
  - Control store was implemented through memory
    - Much less costly than logic gates

IBM System/360 Family



Model	**M30**	**M40**	**M50**	**M65**
Datapath width	8 bits	16 bits	32 bits	64 bits
Control store size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock rate (ROM cycle time)	1.3 MHz (750 ns)	1.6 MHz (625 ns)	2 MHz (500 ns)	5 MHz (200 ns)
Memory capacity	8–64 KiB	16–256 KiB	64–512 KiB	128–1,024 KiB
Performance (commercial)	29,000 IPS	75,000 IPS	169,000 IPS	567,000 IPS
Performance (scientific)	10,200 IPS	40,000 IPS	133,000 IPS	563,000 IPS
Price (1964 $)	$192,000	$216,000	$460,000	$1,080,000
Price (2018 $)	$1,560,000	$1,760,000	$3,720,000	$8,720,000

CISC

- Moore's Law → Larger memories → Much more complicated ISAs
- VAX-11/780 (1977)
  - 5,120 words x 96 bits (its predecessor only 256 words x 56 bits)

The Intel 8800 Fault

- Design an ISA that would last the lifetime of Intel
  - Too ambitious and too late in the development
- Plan B: **8086 ISA**
  - 10 person-weeks over three regular calendar weeks
  - Essentially by extending the 8-bit registers and instruction set of the 8080 to 16 bits

8086 ISA

- IBM used an 8-bit bus version of the 8086
- IBM announced the PC on August 12, 1981
  - Hope: sell 250,000 PCs by 1986
  - ...but...**Sold 100 million worldwide!**

From CISC to RISC

- Unix experience: high-level languages could be used to write OSs
- Critical question became

  “What instructions would compilers generate?”

  instead of

  “What assembly language would programmers use?”

From CISC to RISC

- Observation 1
  - It was found that 20% of the VAX instructions needed 60% of the microcode and represented only 0.2% of the execution time
- Observation 2
  - Large CISC ISA → Large microcode → high probability of bugs in microcode
- **Opportunity to switch from CISC to RISC**

RISC

- RISC instructions simple as microinstructions
  - Can be executed directly by the hardware
- Fast memory (formerly used for microcode)
  - Repurposed to be a cache of RISC instructions
- Register allocators based graph-coloring
  - Allows compilers to efficiently use registers
- Moore's Law
  - Enough transistors in the 1980s to include a full 32-bit datapath, along with I$ and D$ in a single chip

RISC

- IEEE International Solid-State Circuits Conference, in 1984
  - Berkeley, RISC-I and Stanford MIPS
  - Superior in performance than commercial processors
  - (*RISC-I and MIPS developed by few graduate students!*)

RISC Supremacy

- x86 shipments have fallen almost 10% per year since the peak in 2011
- Chips with RISC processors have skyrocketed to 20 billion!
- CISC based x86 ISA
  - x86 instructions converted on-the-fly to RISC instructions

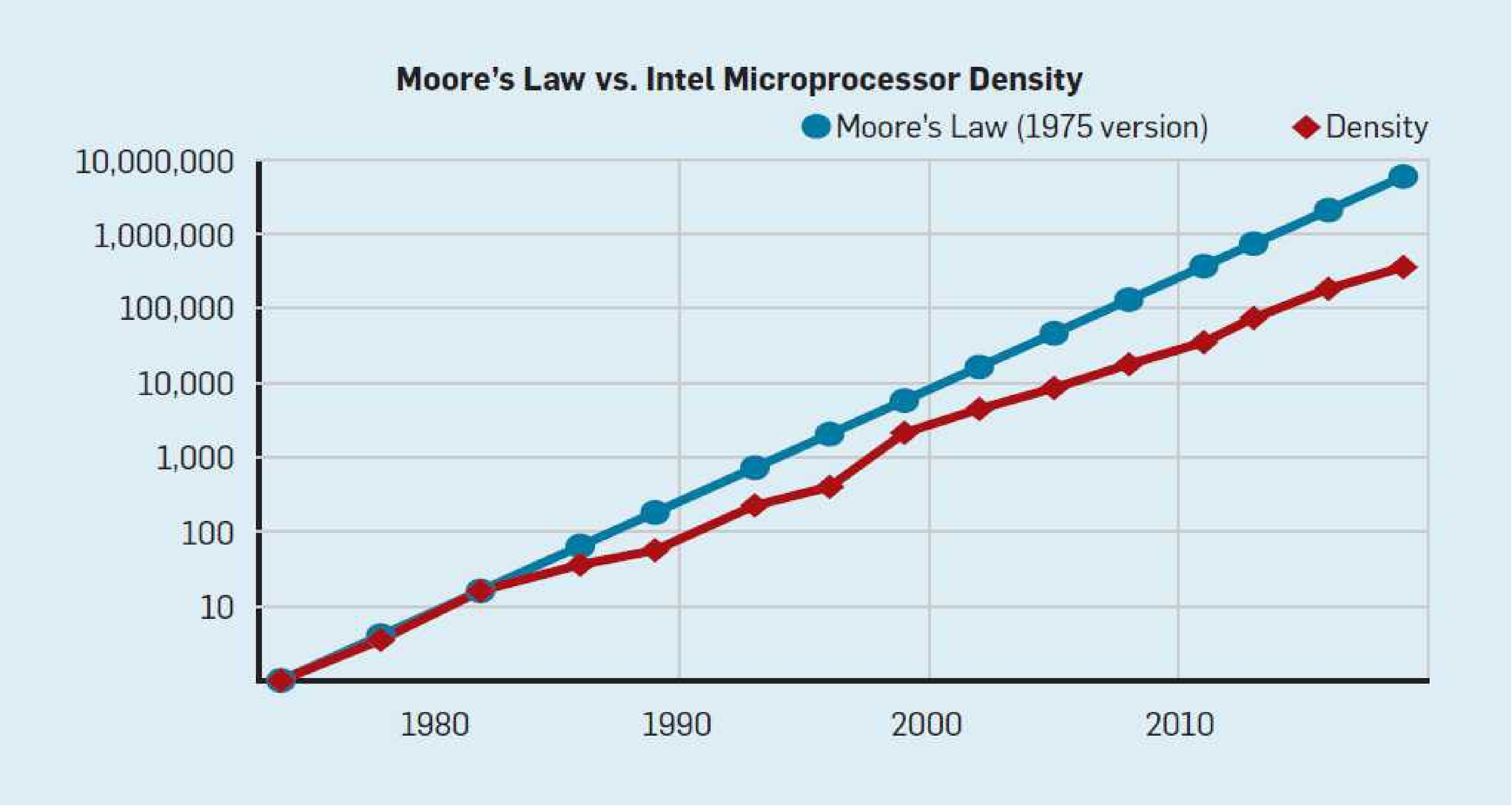
Quantitative Approach

$$\boxed{\textbf{CPU Time} = \textbf{IC} \times \textbf{CPI}}$$

- $IC_{CISC} \approx 75\%\ IC_{RISC}$
- $CPI_{CISC} \approx 6 \times CPI_{RISC}$
- **$\text{CPU Time}_{CISC} \approx 4 \times \text{CPU Time}_{RISC}$**

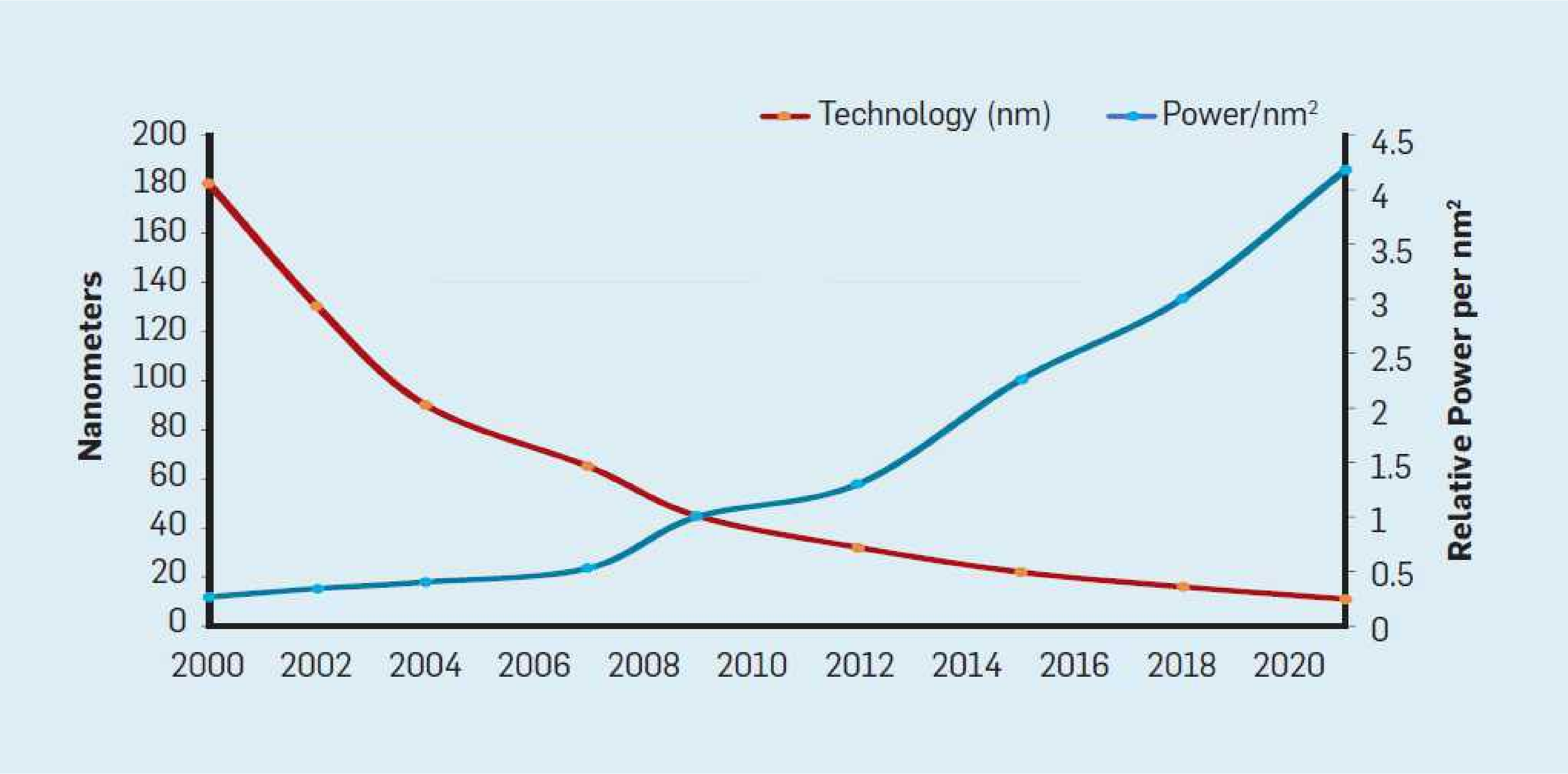
[Flynn, M. “Some computer organizations and their effectiveness”. IEEE Transactions on Computers 21, 9 (Sept. 1972)]

End of Moore's Law



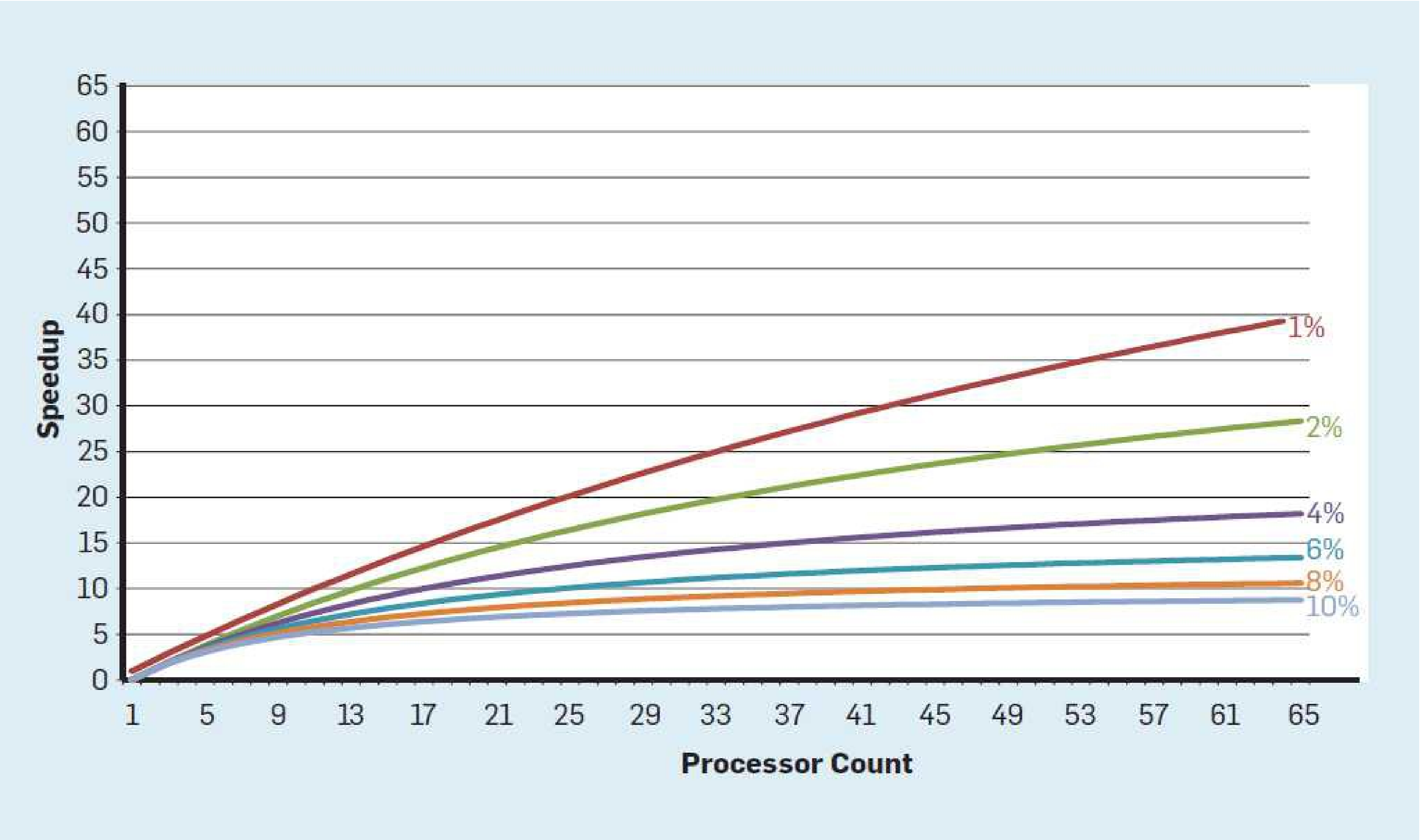
[Moore, G. No exponential is forever: But 'forever' can be delayed! [semiconductor industry]. In Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (San Francisco, CA, Feb. 13). IEEE, 2003, 2023.]

End of Dennard Scaling

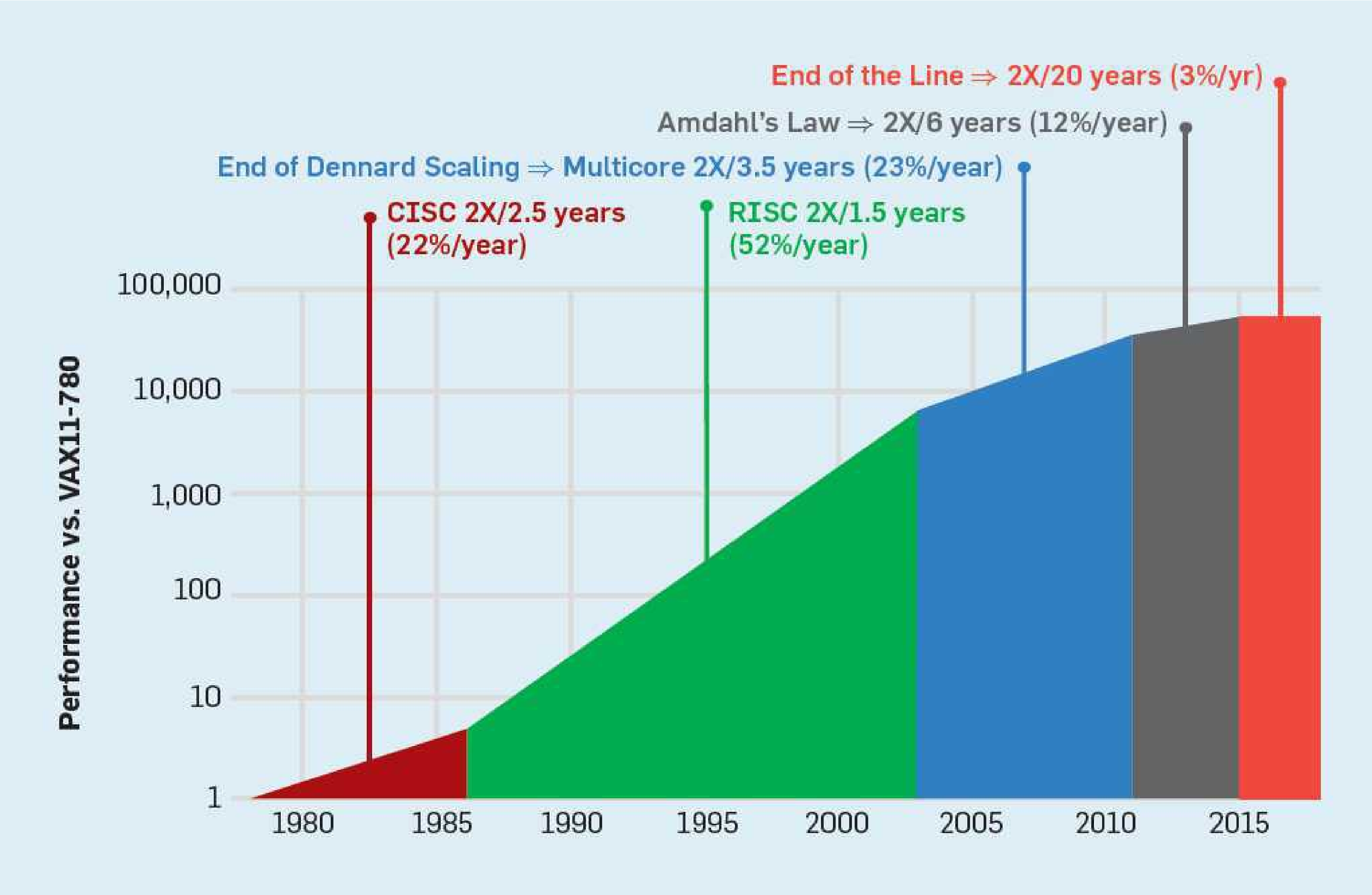


[Dennard, R. et al. Design of ion-implanted MOSFETs with very small physical dimensions. IEEE Journal of Solid State Circuits 9, 5 (Oct. 1974), 256268]

Amdahl's Law for Parallel Computing



End of Growth of Single Program Speed



[J. L. Hennessy and D. A. Patterson. A New Golden Age for Computer Architecture. Communications of the ACM, 62(2), Feb. 2019.]

Agenda

- A bit of history
- Inefficiency in GP architectures
- Domain Specific Architectures
- Source of acceleration
- Cost models
- Communication issues

General-Purpose CPU

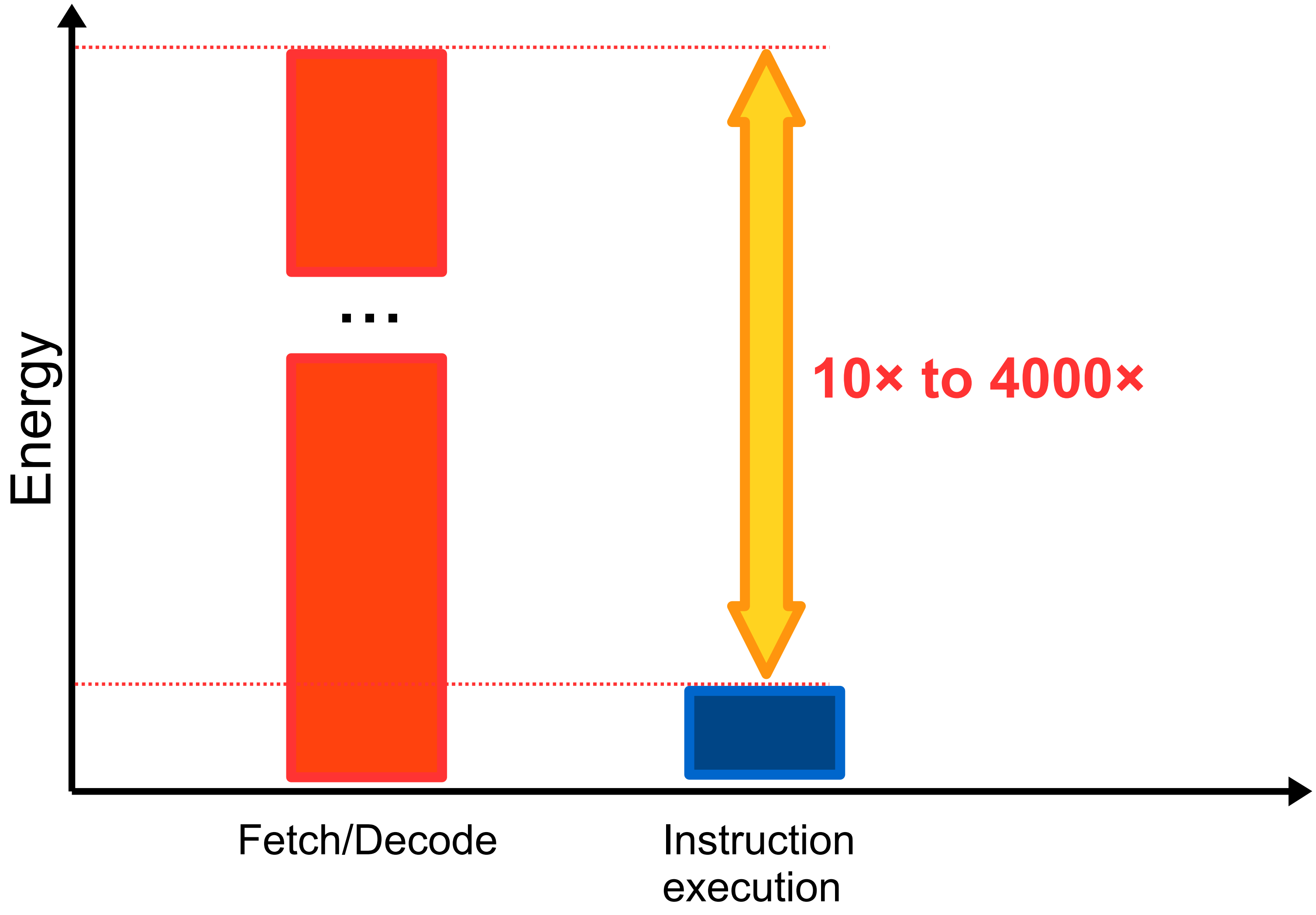
- Easy to program
- Large code bases exist

The “Turing Tariff”

- Refers to the cost of performing functions using GP hardware
- The theoretical machine proposed by Alan Turing could perform any function, but not necessarily efficiently

Prof. Paul Kelly, *Imperial College, London*

The “Turing Tariff”



What Opportunities Left?

- Hardware-Centric approach
- Software-Centric approach
- Combination

Hardware-Centric Approach

- Domain-Specific Architecture
  - *a.k.a.* Domain Specific Accelerator (DSA)
  - Tailored to a specific problem domain

ASIC *vs.* DSA

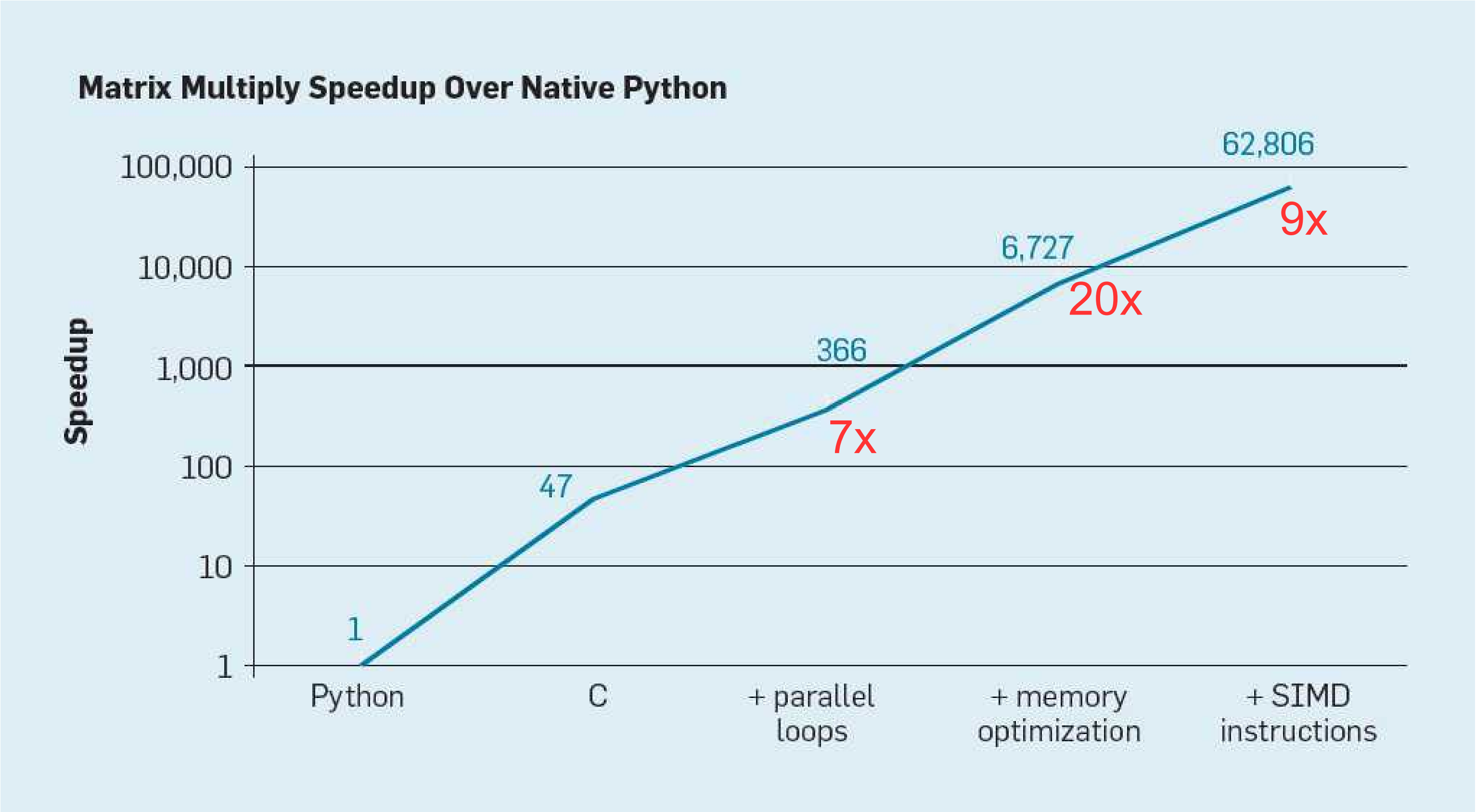
- **ASIC**: often used for a *single* function
  - With code rarely changes
- **DSA**: specific for a *class* of applications

DSA Examples

- Graphic Processing Units (GPUs)
- Neural Network Processors
- Processors for Software-Defined Networks (SDNs)
- ...

Inefficiencies of HL Languages

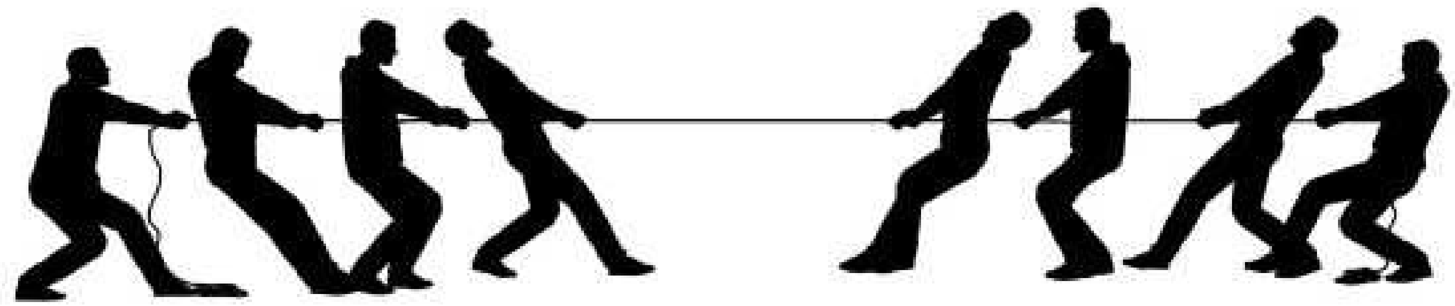
- SW makes extensive use of HL languages
  - Typical interpreted → Inefficient



[Leiserson, C. et al. There's plenty of room at the top. Science, June 2020, Vol 368(6495)]

Productivity *vs.* Efficiency

- Big gap between
  - Modern languages: emphasizing productivity
  - Traditional approaches: emphasizing performance



Modern languages
(**productivity** first objective)

Traditional approaches
(**performance** first objective)

Software-Centric Approach

- Domain-Specific Languages
- DSAs require targeting high-level operations to the architecture
  - Too difficult to extract structure information form general-purpose languages (Python, C, Java, ...)
- Domain-Specific Languages
  - Make vector, dense/sparse matrix operations explicit
  - Help compiler to map operations to the processor efficiently

DSLs Examples

- Matlab: for operating on matrices
- TensorFlow: dataflow language for programming DNNs
- P4: for programming SDNs
- Halide: for image processing specifying high-level transformations

DSLs Challenges

- Architecture independence
  - SW written in a DSL can be ported to different architectures achieving high efficiency in mapping the SW to the underlying DSA
- Example XLA system
  - Translates TensorFlow to heterogeneous processors that use Nvidia GPUs or Tensor Processor Units (TPUs)

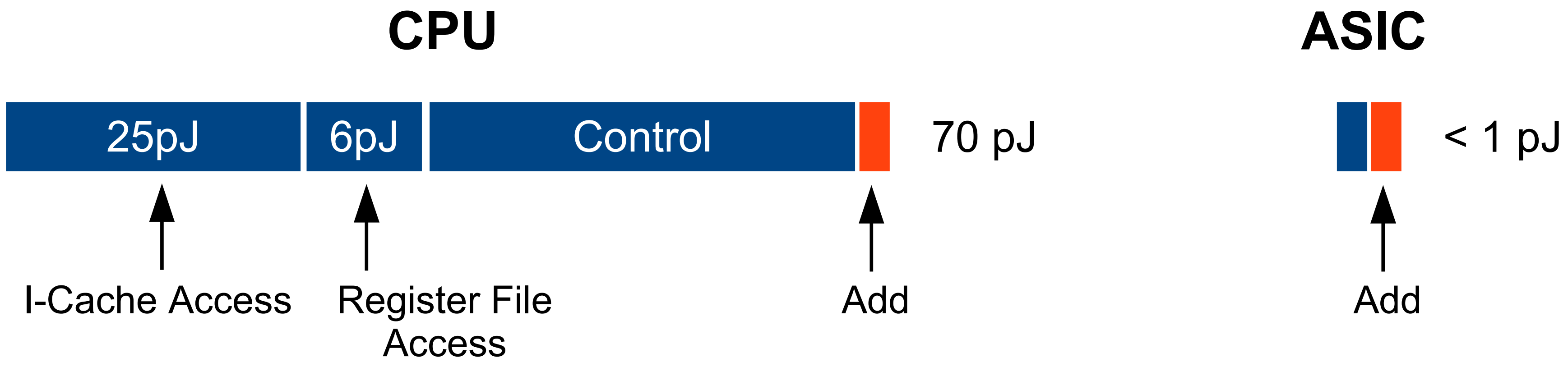
Combination

- Domain Specific Languages & Architectures

Agenda

- A bit of history
- Inefficiency in GP architectures
- Domain Specific Architectures
- Source of acceleration
- Cost models
- Communication issues

Reduced Overhead



CPU

25pJ

6pJ

Control

70 pJ

I-Cache Access

Register File Access

Add

ASIC

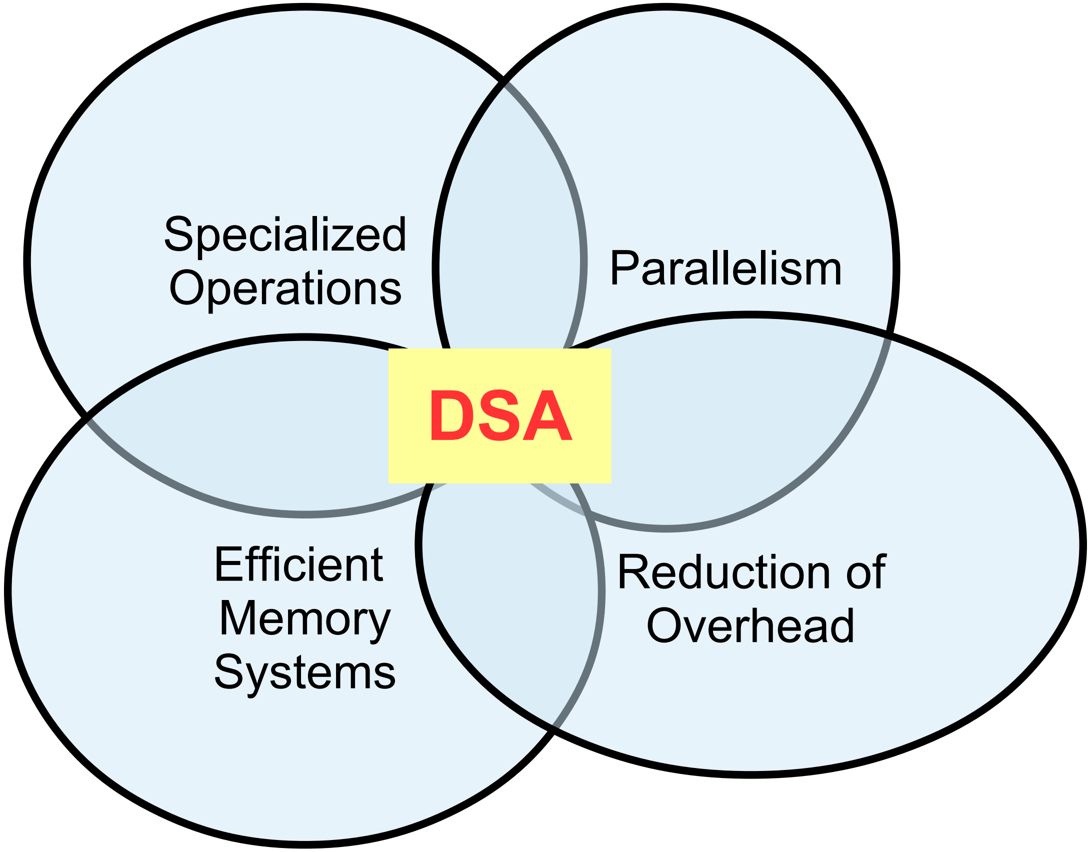
< 1 pJ

Add

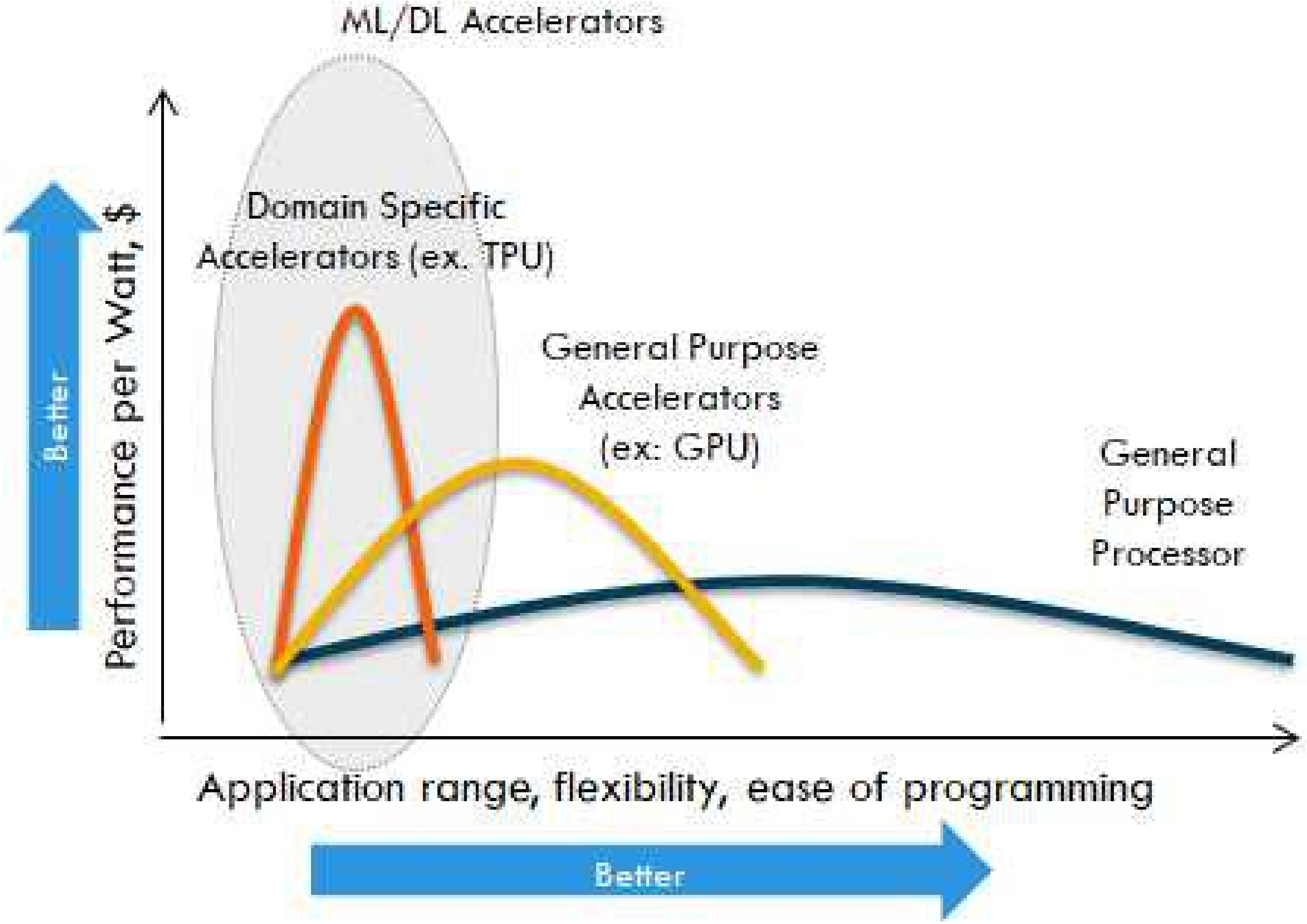
[M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014, pp. 10-14]

Domain Specific Accelerators

- A hardware computing engine that is **specialized** for a particular **domain of applications**



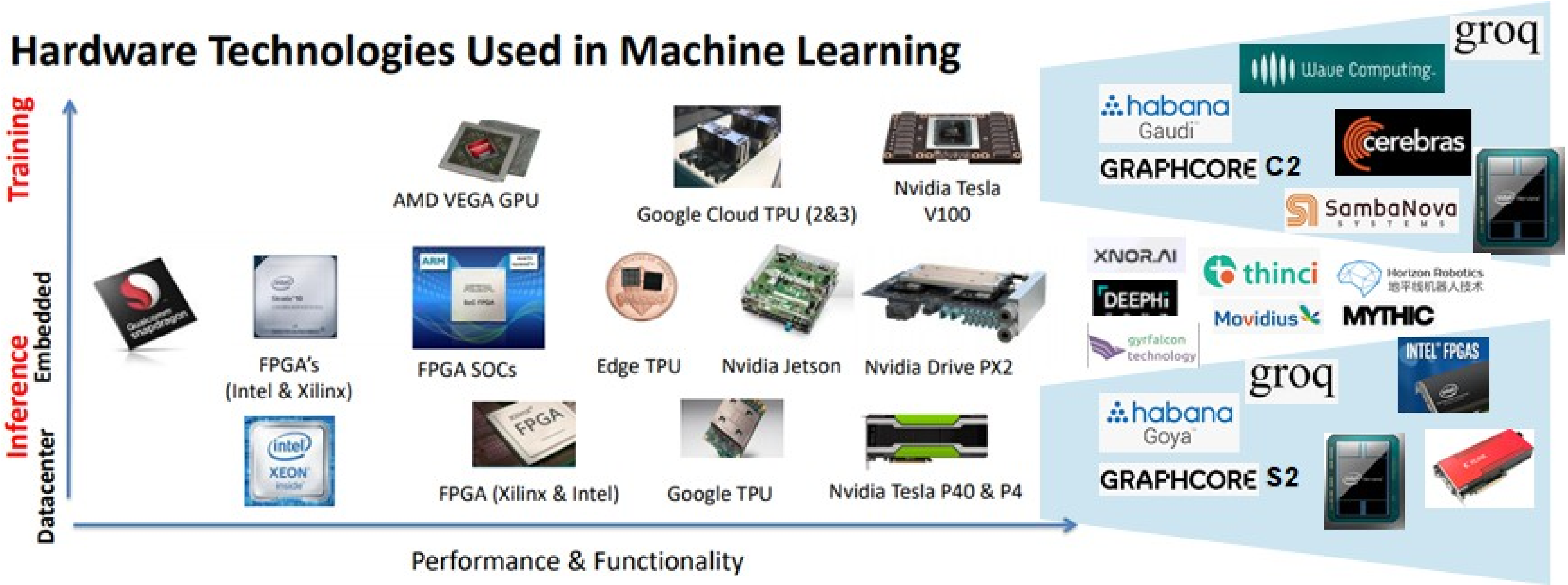
Domain Specific Accelerators



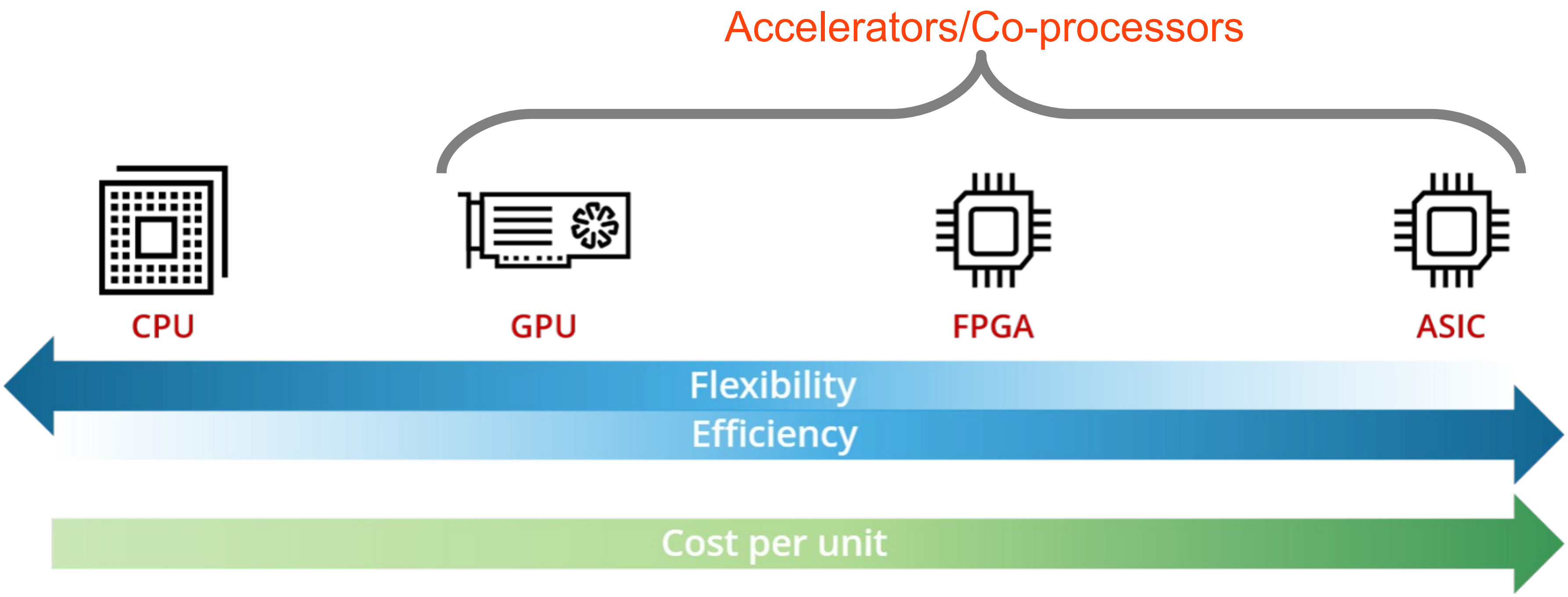
Domain Specific Accelerators

- Several different domains
  - Graphics
  - Deep learning
  - Simulation
  - Bioinformatics
  - Image processing
  - Security
  - ...

Machine Learning Domain



Landscape of Computing



Accelerators/Co-processors

CPU

GPU

FPGA

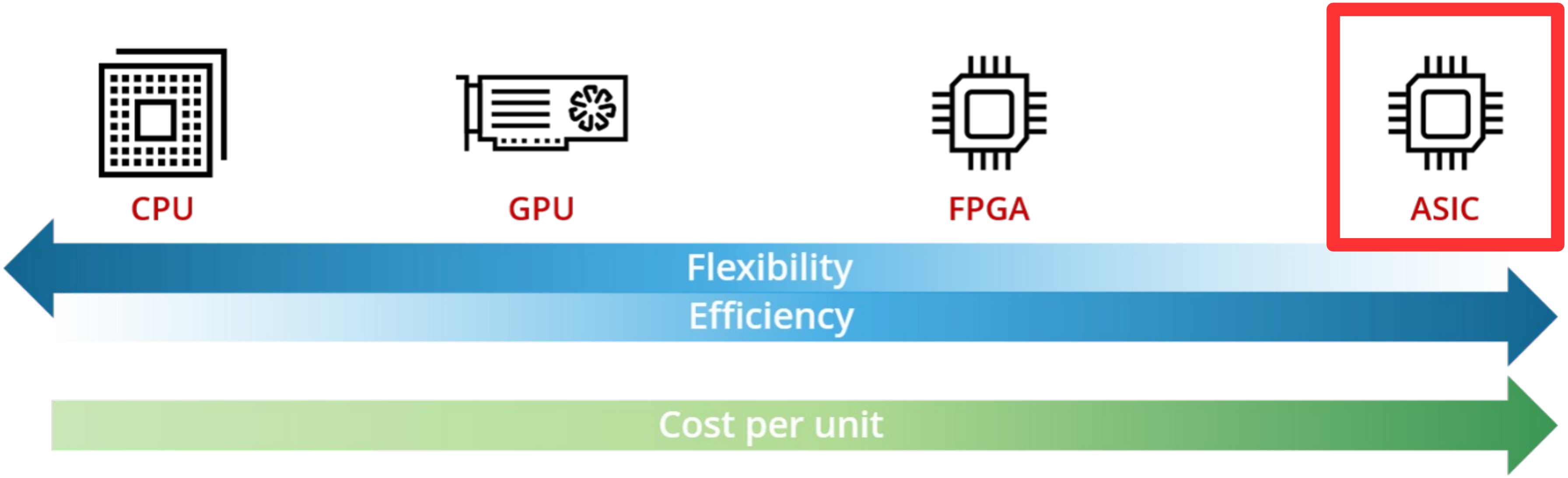
ASIC

Flexibility

Efficiency

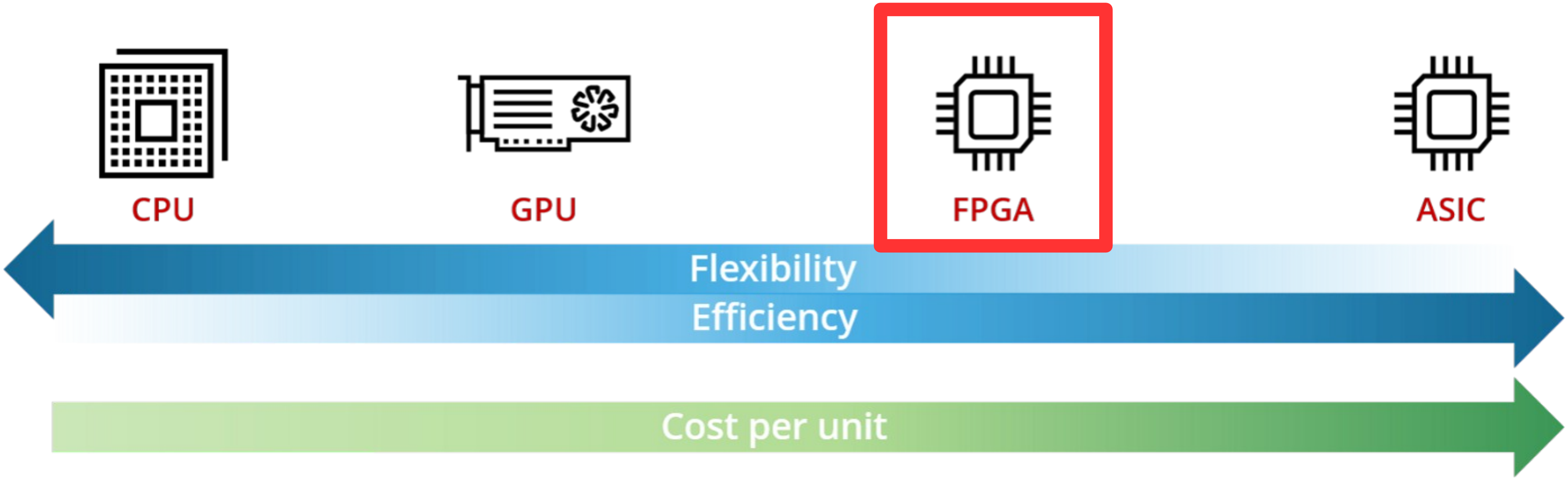
Cost per unit

Acceleration Options -- ASIC



- ☺ Highest efficiency
- ☹ High nonrecurring engineering (NRE) cost
- ☹ Poor programmability
- ☹ Hardwired logic for a single application domain

Acceleration Options -- FPGA

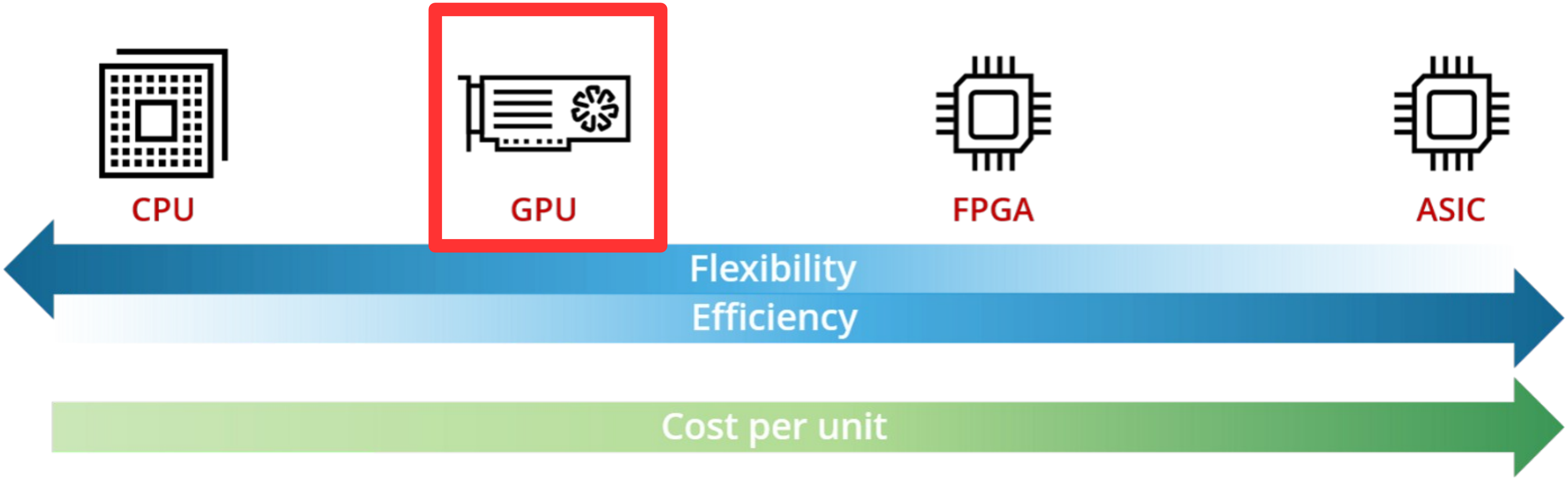


☹ šowers the efficiency by 10–100×

☺ Dynamically configured for different applications

☺ Allows for an accelerator to be instantiated near the data it operates on, reducing communication cost

Acceleration Options -- GPU

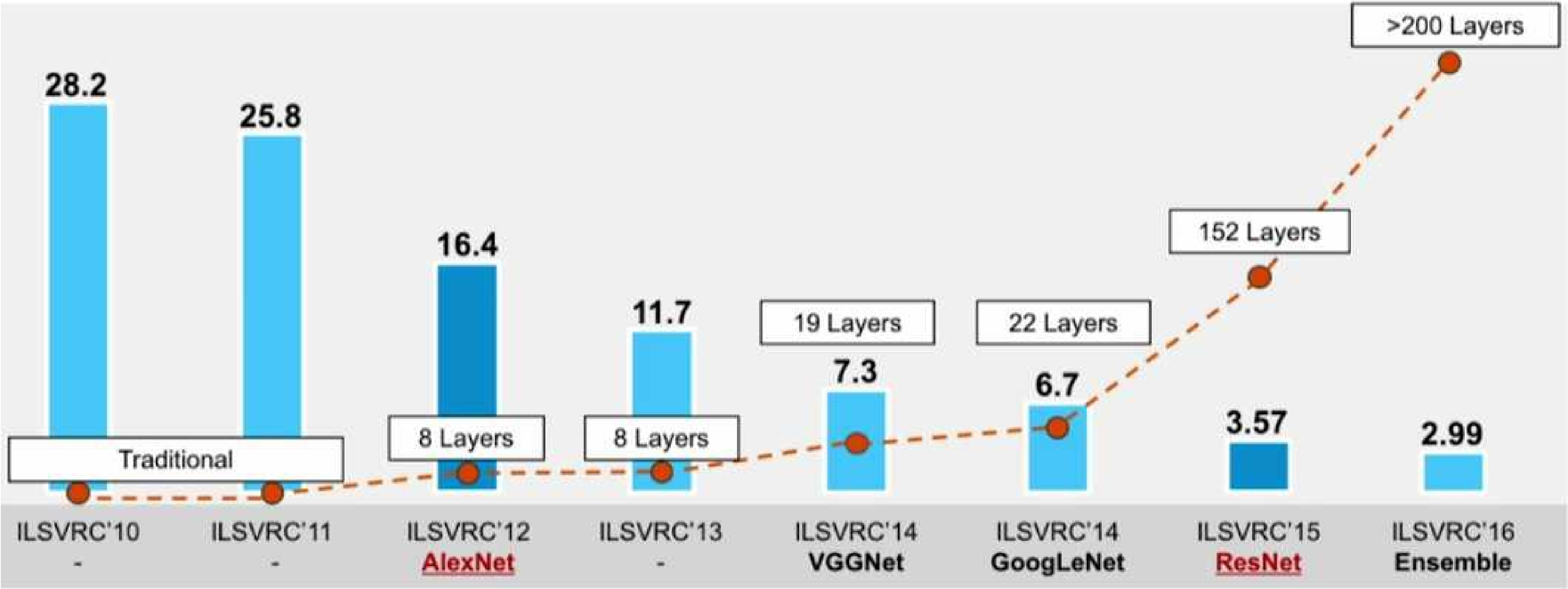


- ☺ Accelerate multiple domains by incorporating specialized operations
- ☺ Offers order of magnitude better efficiency than CPUs (near-ASIC efficiency for the application they accelerate)
- ☹ ˙ŒË° ›ℜ› %₫Nℓ£ C € ε ‹ › đ £C› %₫ Cℓ ₡ℑ ℐngle-thread application

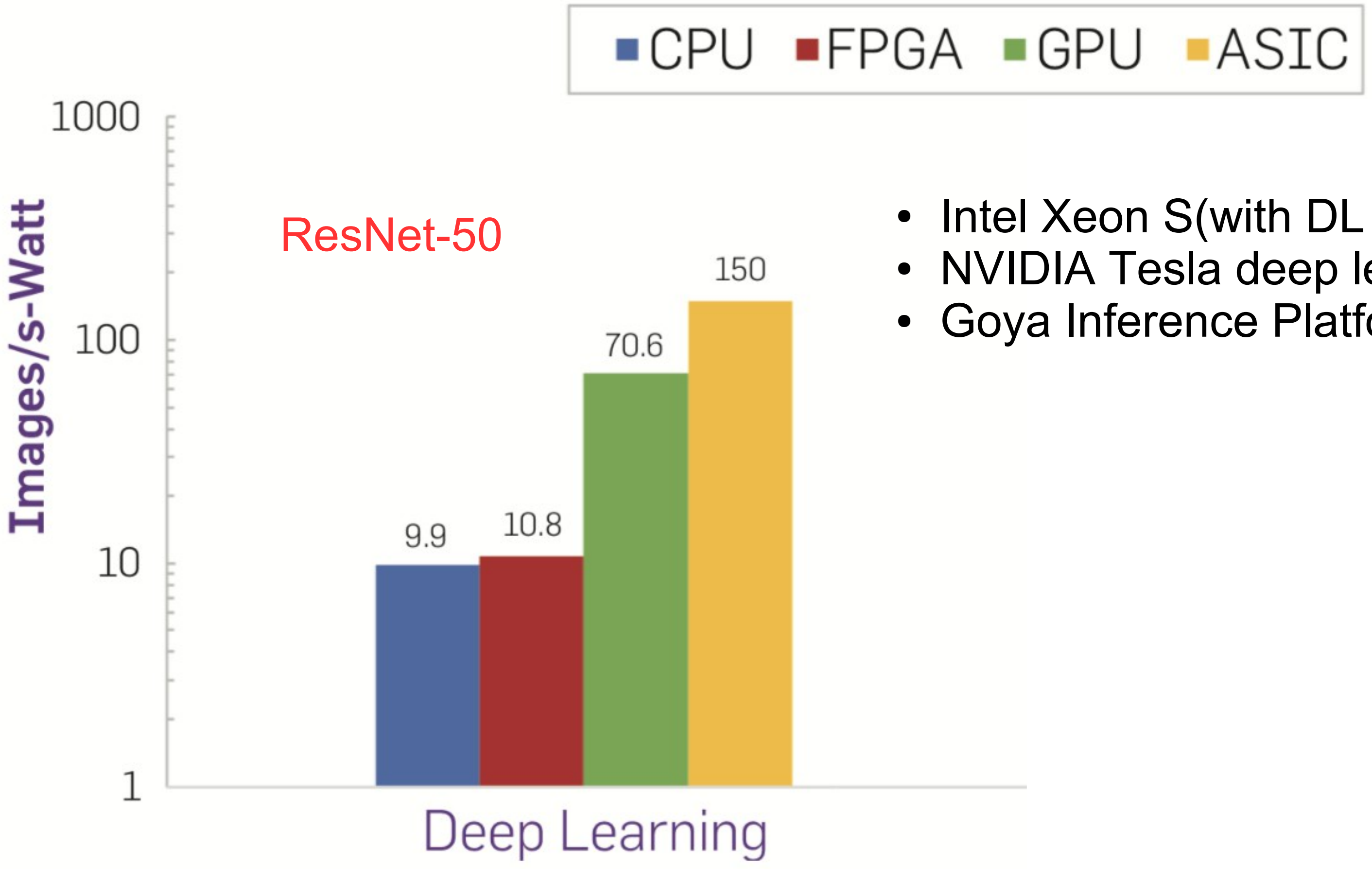
Example

- Deep Learning
- Genomics

Deep Neural Networks



Comparison

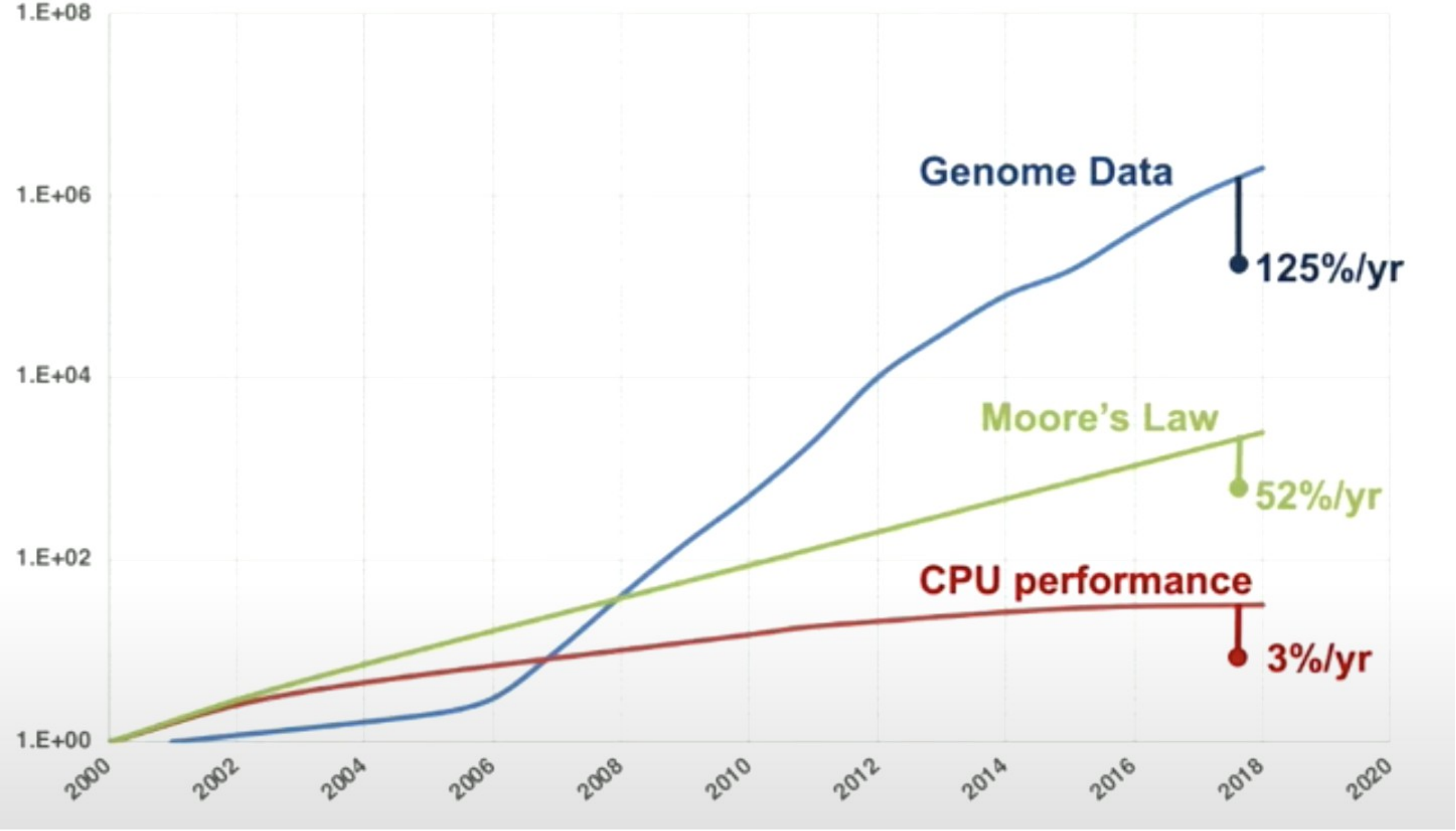


ResNet-50

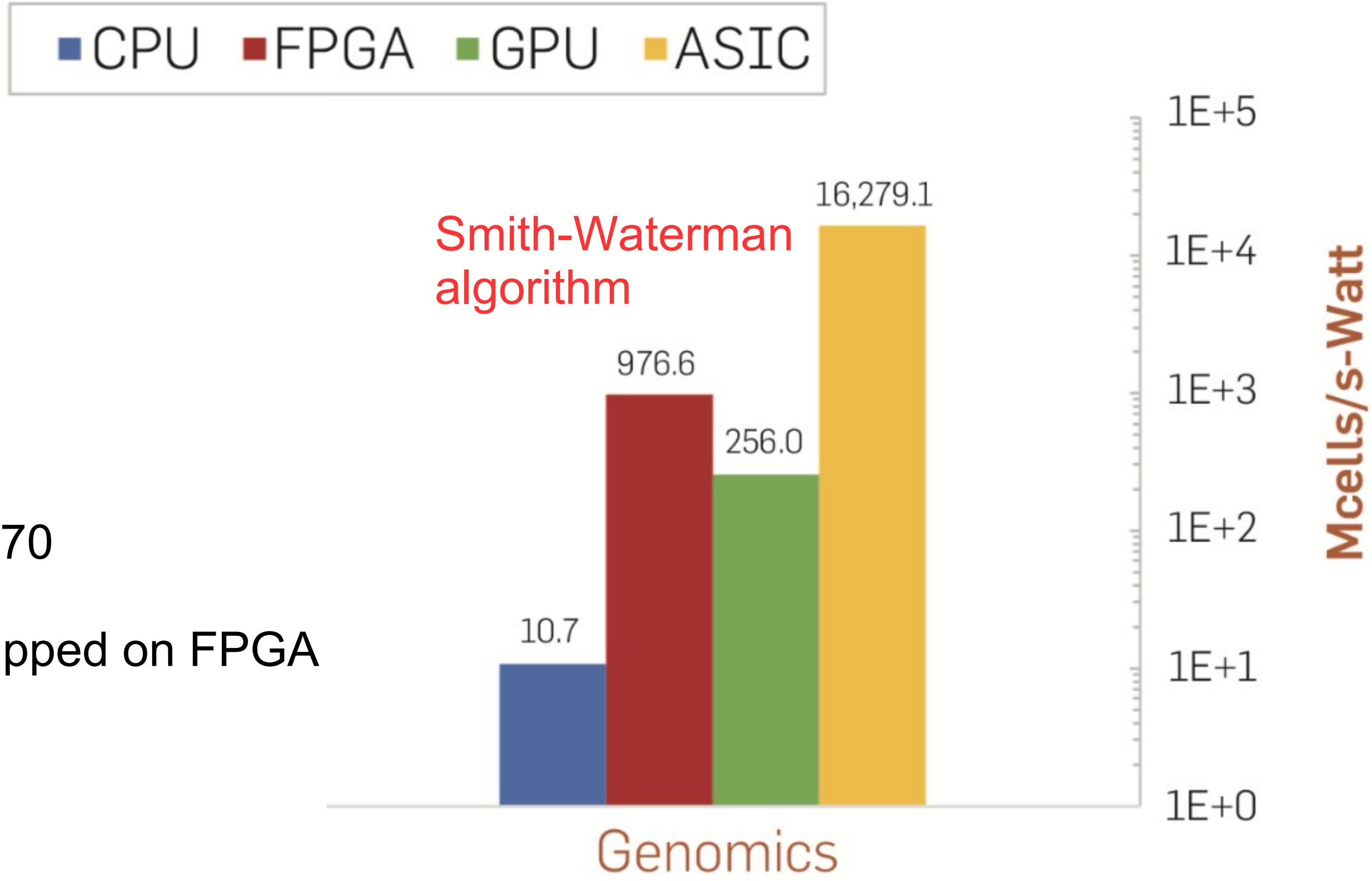
- Intel Xeon S(with DL Boost technology)
- NVIDIA Tesla deep learning product
- Goya Inference Platform

[W. J. Dally, *et al*., Domain-Specific Hardware Accelerators. Communications of the ACM, 63(7), pp. 48-57, July 2020.]

Genomic Data



Comparison



- Intel Xeon E5-2670
- NVIDIA V100
- Darwin-WGA mapped on FPGA

[W. J. Dally, *et al*., Domain-Specific Hardware Accelerators. Communications of the ACM, 63(7), pp. 48-57, July 2020.]

Cost *vs.* Performance

- Banded Smith-Waterman algorithm
  - In CUDA for the GPU in one day
    - 25x improvement in efficiency over the CPU
  - On an FPGA in two months of RTL design and performance tuning
    - 4x the efficiency of the GPU
  - RTL into an ASIC gives
    - 16x the efficiency of the FPGA but with significant nonrecurring costs and lack of flexibility

[Turakhia, Y., et al. “Darwin-WGA: A co-processor provides increased sensitivity in whole genome alignments with high speedup. HPCA (2019)]

Application Porting

- Applications require modifications to achieve high speed up on DSA
  - These applications are highly tuned to balance the performance of CPU with their memory systems
- Specialization reduces the cost of processing to near zero
  - They become memory limited

Agenda

- A bit of history
- Inefficiency in GP architectures
- Domain Specific Architectures
- Source of acceleration
- Cost models
- Communication issues

Sources of Acceleration

- Techniques for performance/efficiency gain
  - Data Specialization
  - Parallelism
  - Local and optimized memory
  - Reduced overhead

Data Specialization

- Specialized operations on domain-specific data types can do in one cycle what may take tens of cycles on a conventional computer

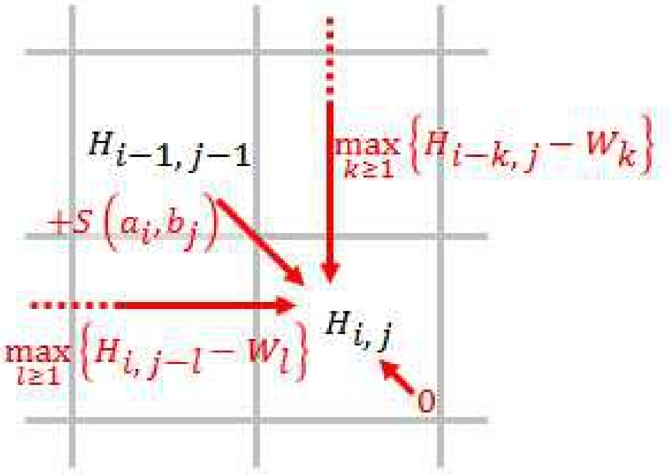
Data Specialization

- Example 1 – Smith-Waterman algorithm
  - Used in genome analysis to align two gene sequences

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

$$D(i,j)=max\{H(i-1,j)-o, D(i-1,j)-e\} \quad (2)$$

$$H(i,j)=max\begin{cases}0\\ I(i,j)\\ D(i,j)\\ H(i-1,j-1)+W(r_i,q_j)\end{cases} \quad (3)$$



  - Computation performed in 16-bit integer arithmetic

Data Specialization

- Conventional x86 processor without SIMD vectorization
  - 37 cycles
    - 35 arithmetic and logical operations
    - 15 load/store operations

Data Specialization

- Intel Xeon E5-2620 4-issue, out-of-order, 14 nm
  - 37 cycles and 81nJ (mostly for spent fetching, decoding, and reordering instructions)
- Darwin accelerator, 40 nm
  - 1 cycle, 3.1 pJ (0.3 pJ is consumed computing the recurrence equations)
- **37× speedup, 26,000× energy reduction**

Data Specialization

- Example 2 – EIE accelerator for sparse NNs
  - Store dense networks in compressed sparse
  - Run-length coding for feature maps
  - Compress weights using a 16-entry codebook
  - 30x reduction in size allowing the weights of most networks to fit into efficient, local, on-chip memories
    - Two orders of magnitude less energy to access than off-chip memories

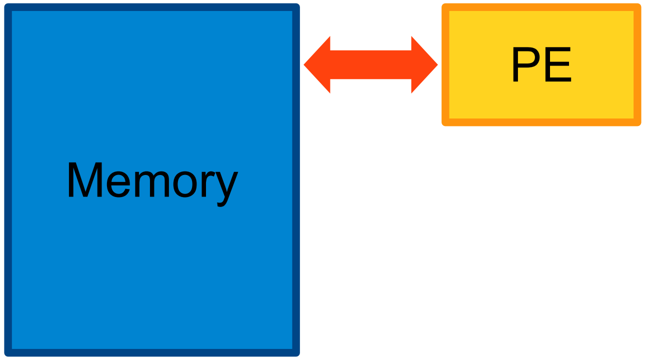
[Han, S., *et al.* EIE: Efficient inference engine on compressed deep neural network. ISCA 2016]

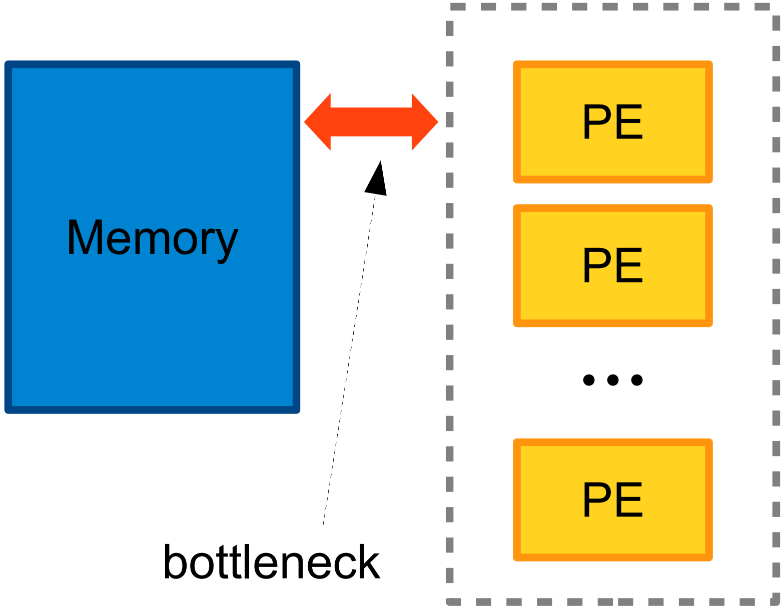
Sources of Acceleration

- Techniques for performance/efficiency gain
  - Data Specialization
  - Parallelism
  - Local and optimized memory
  - Reduced overhead

Parallelism

- High degrees of parallelism provide gains in performance
- Parallel units must exploit locality
  - Make very few global memory references or their performance will be memory bound





Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop

Parallelism

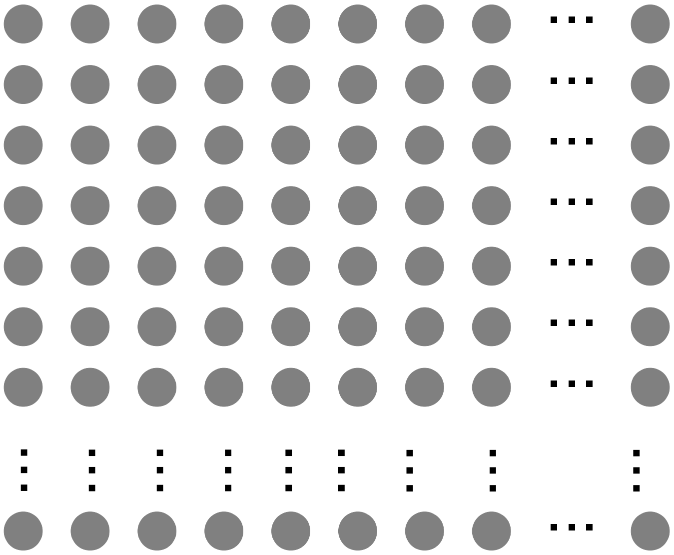
- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
    - 64 separate alignment problems in parallel
    - No communication between subproblems
    - Synchronization required only upon completion of each subproblem
    - Typical billions of alignments → Ample outer-loop parallelism
  - Inner-loop

Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop
    - 64 PEs compute 64 elements of H, I, and D in parallel
    - Element (i, j) depends only on the elements above ($i-1$,j), directly to the left (i,$j-1$), and above to the left ($i-1$,$j-1$)
    - Only nearest neighbor communication between the processing elements is required

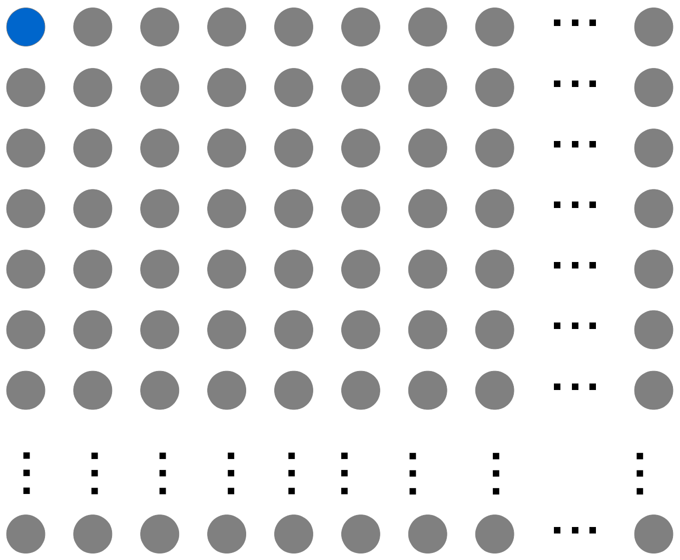
Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop



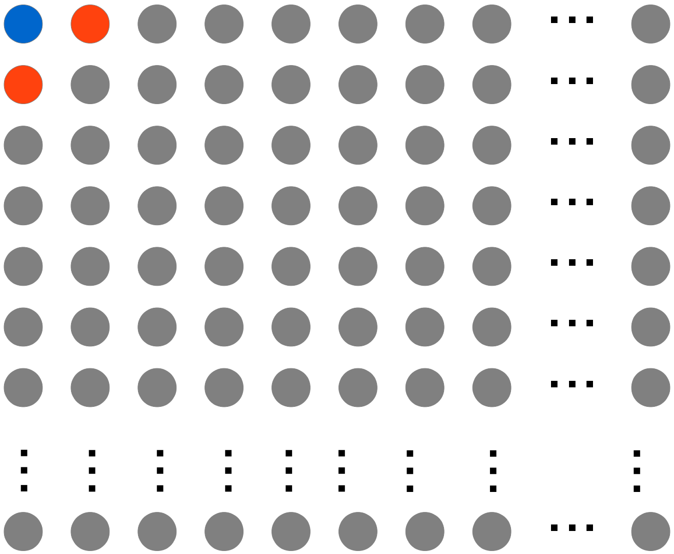
Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop



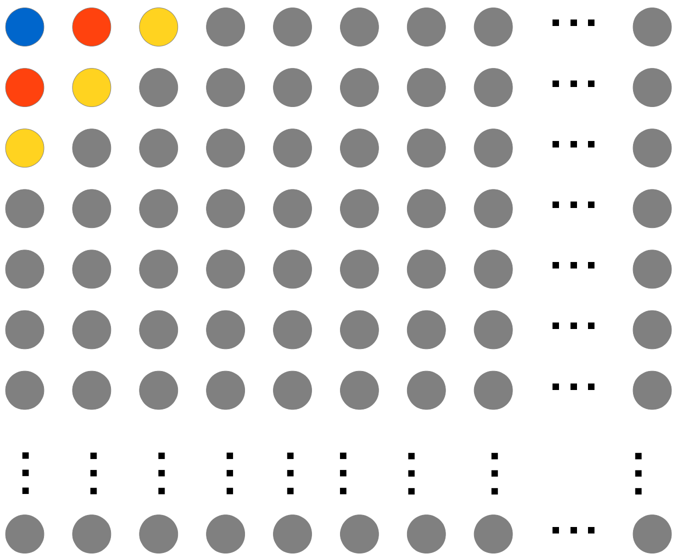
Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop



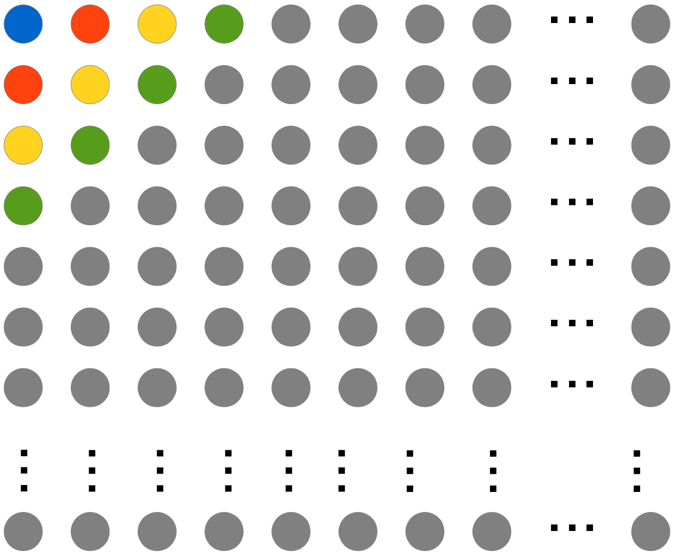
Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop



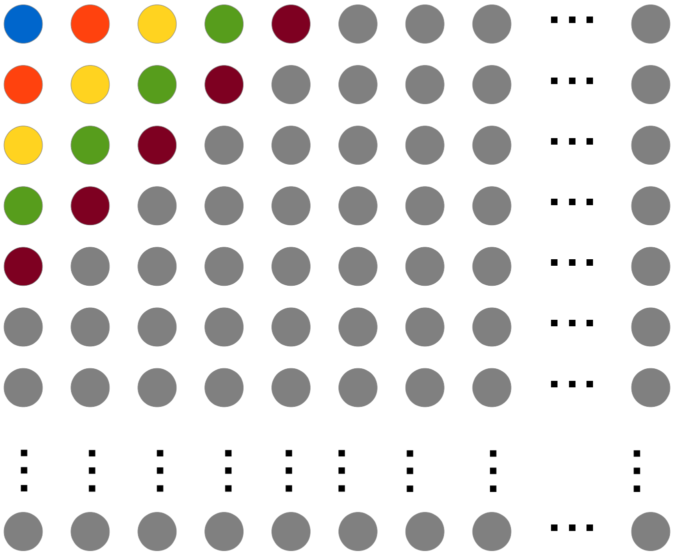
Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop



Parallelism

- Example 1 -- Smith-Waterman algorithm
- Parallelism exploited at two levels
  - Outer-loop
  - Inner-loop



Parallelism

- Very high utilization
  - Outer-loop
    - Utilization close to 100%
    - Until the very end of the computation, there is always another subproblem to process as soon as one finishes
    - With double buffering of the inputs and outputs, the arrays are working continuously
  - Inner-loop
    - Utilization 98.5%
    - Loss of utilization at the start and end of computation (due to the systolic nature of the accelerator)

Parallelism

- Speedup
  - Parallelization speed-up 4,034×
  - Data specialization speed-up 37×
- **Total speed-up 150,000×**

Sources of Acceleration

- Techniques for performance/efficiency gain
  - Data Specialization
  - Parallelism
  - Local and optimized memory
  - Reduced overhead

Local and Optimized Memory

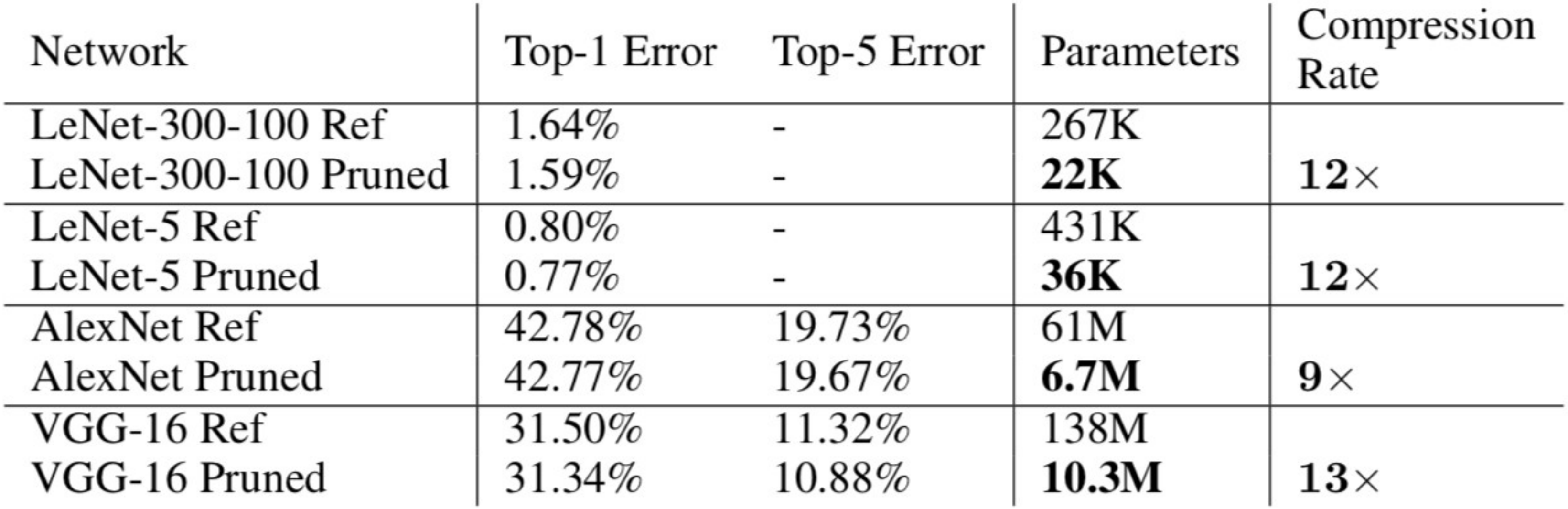
- Storing key data structures in many small, local memories
  - Very high memory bandwidth
  - Low cost and energy

Local and Optimized Memory

- Data compression
  - Increase the effective size of a local memory
  - Increase the effective bandwidth of a memory interface
- Example, NVDLA
  - Weights as sparse data structures → 3×-10× increase in the effective capacity of on-chip memories
- Example, EIE
  - Weights are compressed using a 16-entry codebook → 8× savings compared to a 32-bit float

Weights as Sparse Data Structures

- Pruning techniques
  - Remove not useful neurons and/or connections

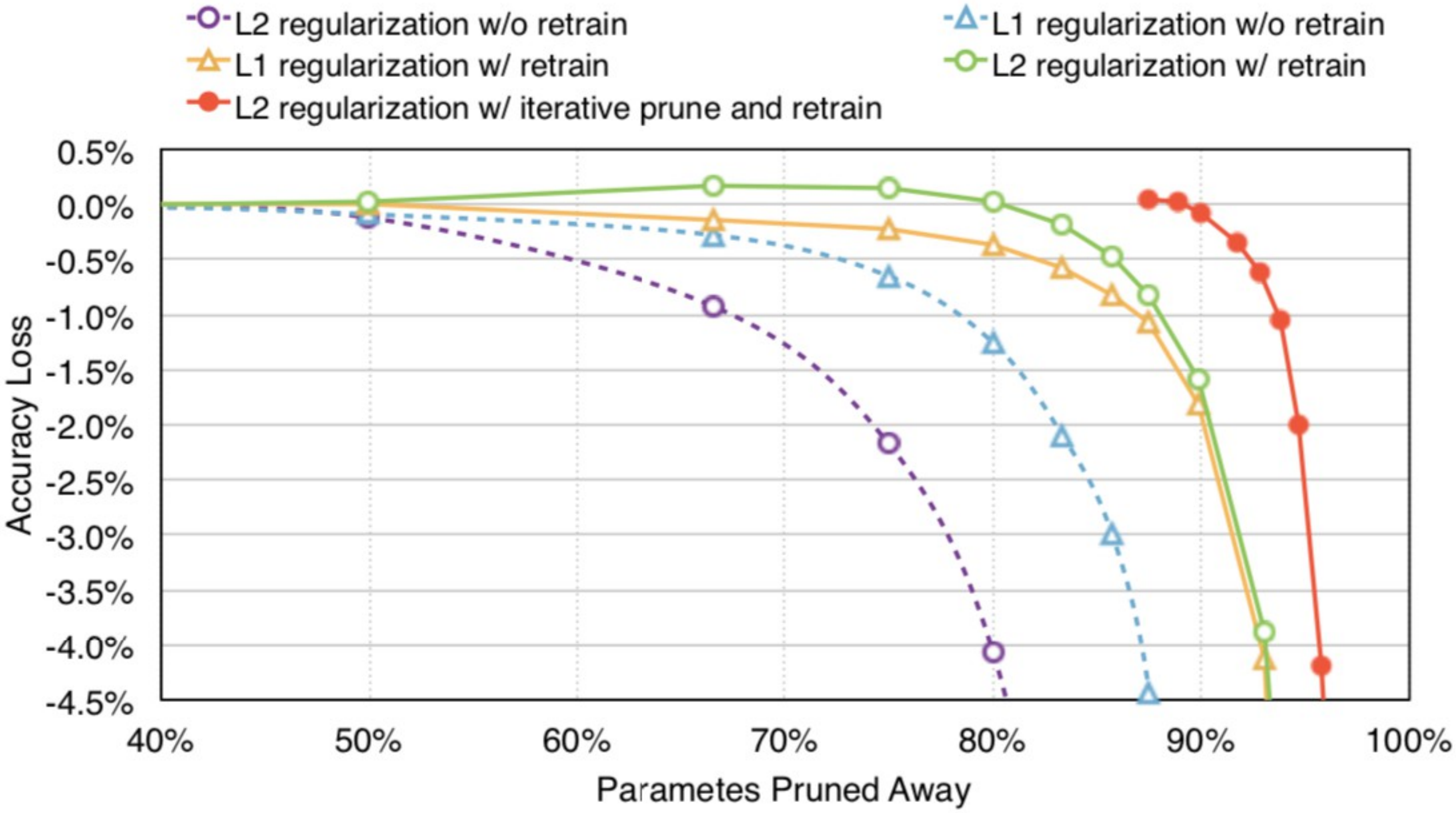


Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG-16 Ref	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13**×

[S. Han, *et al.*, Learning both weights and connections for efficient neural networks. NIPS 2015]

Weights as Sparse Data Structures

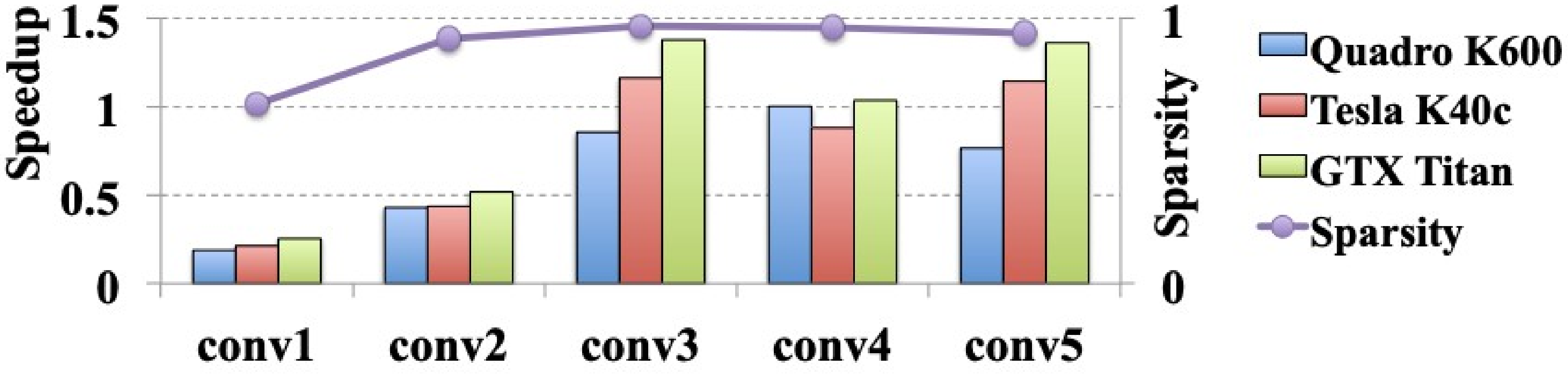
- Pruning techniques



[S. Han, *et al.*, Learning both weights and connections for efficient neural networks. NIPS 2015]

Weights as Sparse Data Structures

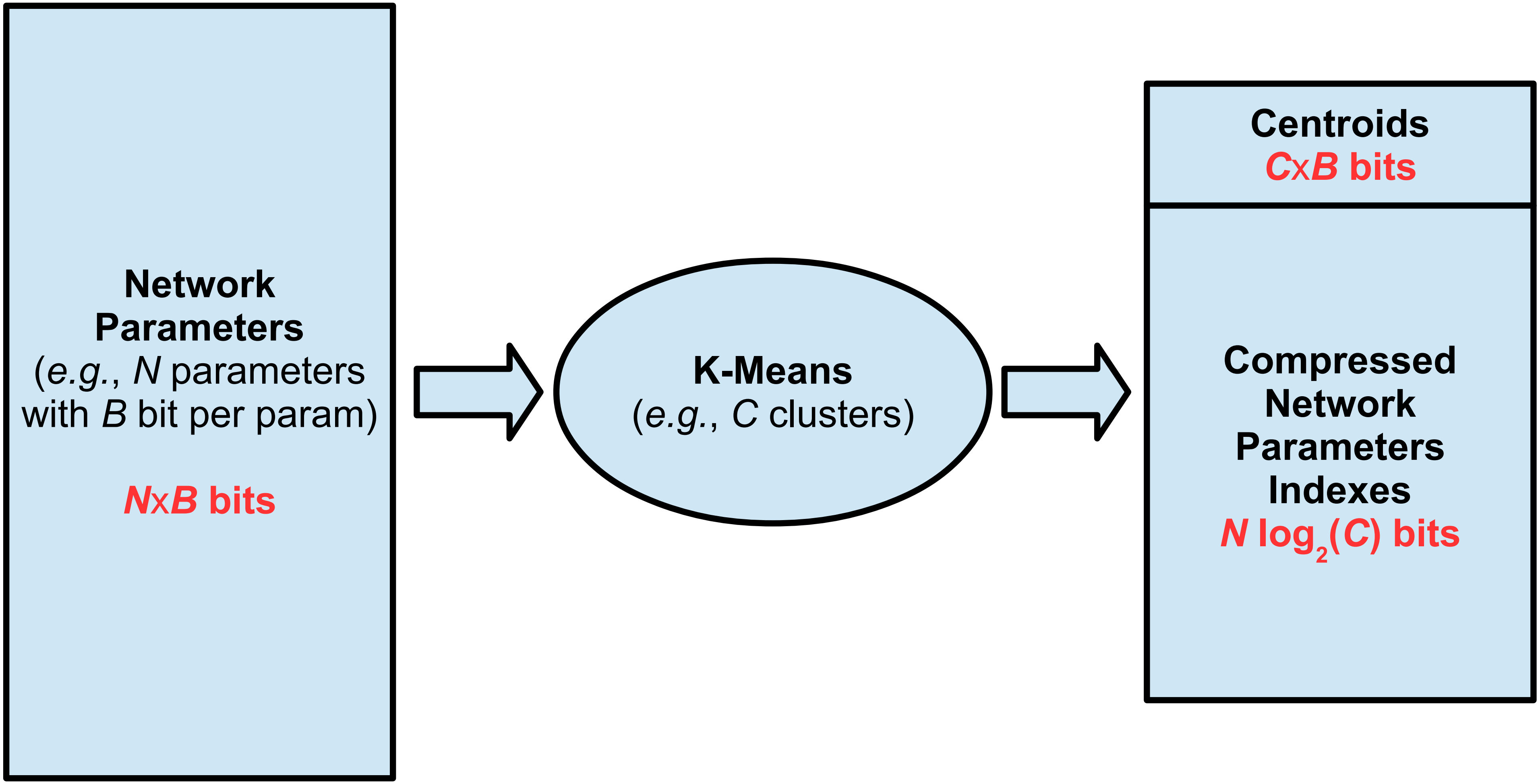
- Pruning techniques
  - Irregular memory access
    - Adversely impacts acceleration in hardware platforms
  - Achieved speedups are either very limited or negative even the actual sparsity is high, >95%



Local and Optimized Memory

- Data compression
  - Increase the effective size of a local memory
  - Increase the effective bandwidth of a memory interface
- Example, NVDLA
  - Weights as sparse data structures → 3×-10× increase in the effective capacity of on-chip memories
- Example, EIE
  - Weights are compressed using a 16-entry codebook → 8× savings compared to a 32-bit float

Weights Compressed with Codebook



Network Parameters
(*e.g.*, N parameters with B bit per param)

$N \times B$ bits

K-Means
(*e.g.*, C clusters)

Centroids
$C \times B$ bits

Compressed Network Parameters Indexes
$N \log_2(C)$ bits

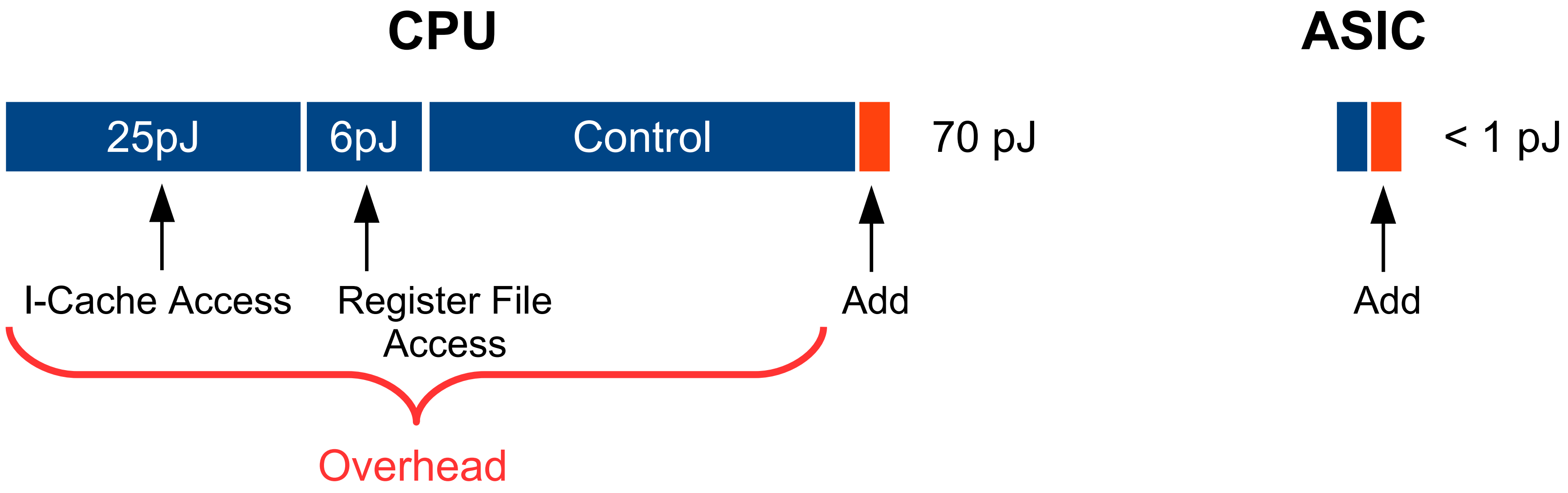
compression ratio
$B/(C \times B + N \times \log_2(C))$

Sources of Acceleration

- Four main techniques for performance and efficiency gains
  - Data Specialization
  - Parallelism
  - Local and optimized memory
  - Reduced overhead

Reduced Overhead

- Specializing hardware reduces the overhead of program interpretation



[M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014, pp. 10-14]

Reduced Overhead

- A simple in-order processor spends over 90% of its energy on overhead
  - Instruction fetch, instruction decode, data supply, and control
- A modern out-of-order processor spends over 99.9% of its energy on overhead
  - Adding costs for branch prediction, speculation, register renaming, and instruction scheduling

[Dally, *et al.* “Efficient embedded computing”, Computer 2008]

[Vasilakis, E. “An instruction level energy characterization of ARM processors”, Tech. Rep. FORTHICS/TR-450, 2015]

Reduced Overhead

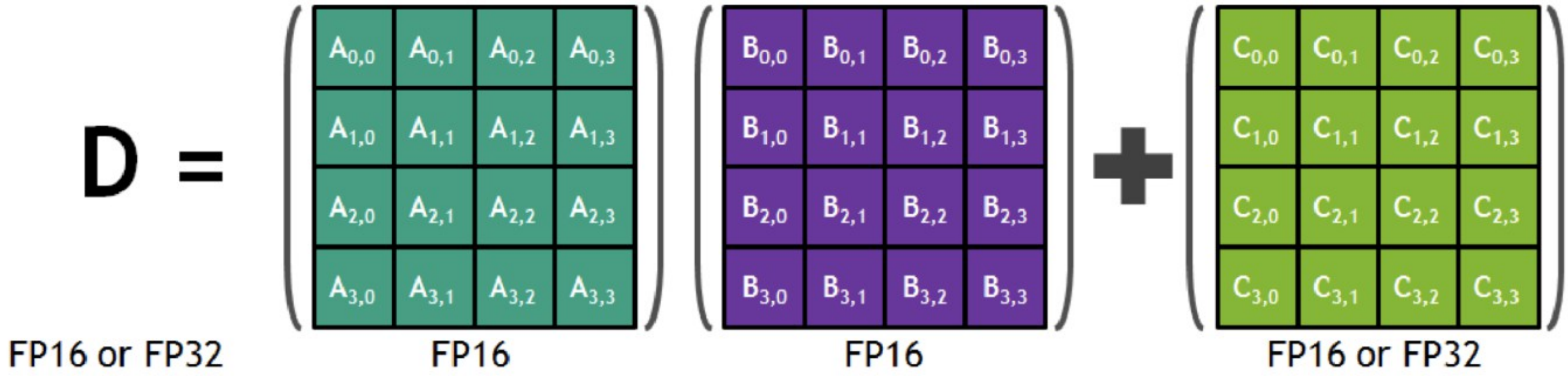
- Example
  - 32 bit integer add @ 28 nm CMOS → **68 fJ**
  - Integer add on 28 nm ARM A-15 → **250 pJ**
    - 4000× the energy of the add itself!

Reduced Overhead

- Overhead reduction in DSAs
  - Most adds do not need full 32-bit precision
  - No instructions to be fetched → no instructions fetch and decode energy
  - No speculation → no work lost due to mis-speculation
  - Most data is supplied directly from dedicated registers → no energy is required to read from a cache or from a large, multi-ported register file

Reduced Overhead

- Complex instructions
  - Matrix-multiply-accumulate instruction (HMMA) of the NVIDIA Volta V100
    - 128 floating-point operations in a single instruction
    - Operation energy many times the instruction overhead



$$D = \left(\begin{matrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{matrix}\right) \left(\begin{matrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{matrix}\right) + \left(\begin{matrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{matrix}\right)$$

128 FP ops: 64 half-precision multiplies and 64 single-precision adds

Codesign is Needed

- Achieving high speedups and gains in efficiency from specialized hardware usually requires modifying the underlying algorithm

Codesign is Needed

- Existing algorithms are highly tuned for conventional general-purpose processors
  - Tuned to balance the performance of conventional processors with their memory systems
- Specialization makes cost of processing nearly zero
  - Algorithm becomes memory dominated

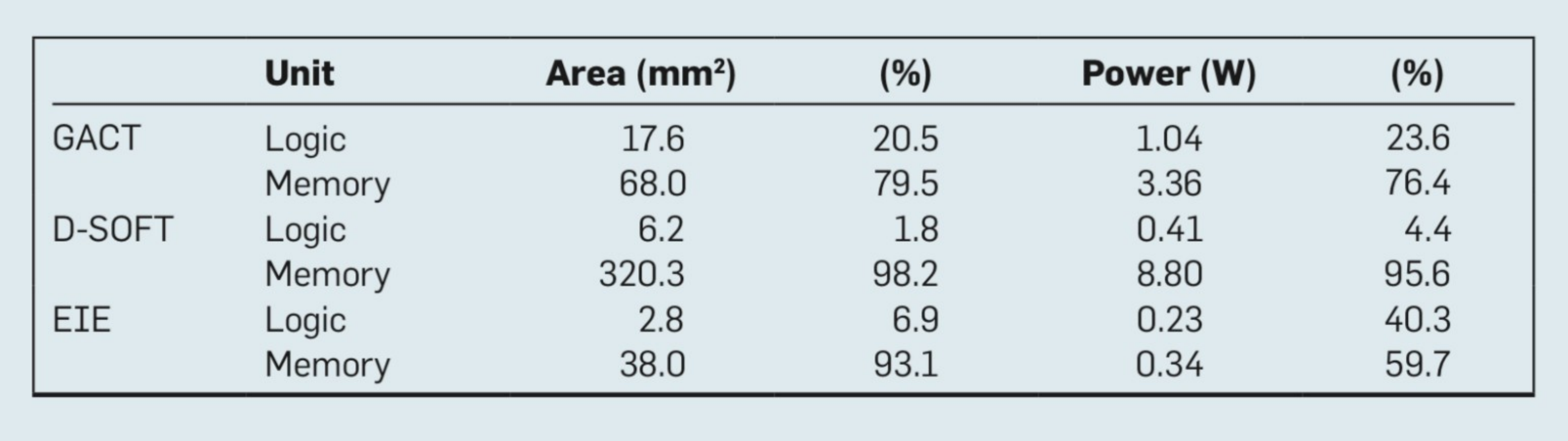
Memory Dominates Accelerators

- GACT
  - Dynamic programming module in Darwin platform
  - Kernel: 16-bit additions and comparisons
- D-SOFT
  - D-SOFT filtering hardware module in Darwin platform
  - Kernel: simple arithmetic and comparisons

Darwin Accelerator

- EIE Sparse NN Accelerator
  - Kernel: Matrix Vector Multiplication

Memory Dominates Accelerators



	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

TSMC 40 nm technology

When logic is “free,” memory dominates!

[Turakhia, Y. et al. , “Darwin: A genomics co-processor provides up to 15,000× acceleration on long read assembly”. ASPLOS 2018]

[Han, S. *et al.*, “EIE: Efficient inference engine on compressed deep neural network”. ISCA 2016]

Specialization *vs.* Generality

☺ Engine specialized for just one application → highest possible efficiency

☹ Range of use may be too limited to generate enough volume to recover design costs

☹ New algorithm may be developed rendering the accelerator obsolete

Specialization *vs.* Generality

- Smoothing the transition... Accelerates a domain of applications not a single application

Special Instructions *vs.* Special Engines

- Building accelerators for broad domains by adding specialized instructions to a general-purpose processor

Special Instructions *vs.* Special Engines

- Example, NVIDIA Volta V100 GPU
  - HMMA (half-precision matrix multiply-accumulate)
    - Multiplies two 4x4 half-precision (16-bit) FP matrices accumulating the results in a 4x4 single-precision (32-bit) FP matrix
    - 128 FP operations: 64 half-precision multiplies and 64 single-precision adds
  - Turing IMMA (integer matrix multiply accumulate)
    - Multiplies 8×8 8-bit integer matrices accumulating an 8x8 32-bit integer result matrix
    - 1024 integer operations
  - Accelerating training and inference for convolutional, fully-connected, and recurrent layers of DNNs

Special Instructions *vs.* Special Engines

- NVIDIA Volta V100 GPU *vs.* Google TPU

Special Instructions *vs.* Special Engines

- NVIDIA Volta V100 GPU
  - HMMA 77% of the energy consumed by arithmetic
  - IMMA 87% of the energy is consumed by arithmetic
- Energy consumed by instruction overhead and fetching the data operands from the large GPU register files and shared memory

Special Instructions *vs.* Special Engines

- Google TPU
  - 23% and 13% more efficient on matrix multiply compared to HMMA and IMMA
  - Use of on-chip memories and optimized data movement

[Jouppi, N.P., *et al.*, Domain-specific architecture for deep neural networks. Commun. ACM 2018]

Special Instructions *vs.* Special Engines

- NVIDIA Volta V100 GPU *vs.* Google TPU
  - GPU die will be larger and hence more expensive
  - It includes area for the general-purpose functions, and for other accelerators, which are unused when doing matrix multiply

Agenda

- A bit of history
- Inefficiency in GP architectures
- Domain Specific Architectures
- Source of acceleration
- Cost models
- Communication issues

Cost Model

- Arithmetic @ 14 nm technology
  - 10 fJ and 4 μm^2 for an 8-bit add operation
  - 5 pJ and 3600 μm^2 for a DPFP multiply

[Horowitz, M. Computing’s energy problem (and what we can do about it). In ISSCC (2014), IEEE, 10–14]

Cost Model

- Local Memory @ 14 nm technology
  - SRAM 8 KB, 50 fJ/bit
  - 0.013 µm^2 per bit
  - Larger on-chip memories
    - Communication cost of getting to and from a small 8 KByte subarray → 100 fJ/bit-mm
    - Several hundred megabytes with today’s technology
      - 100 MB memory 0.7 pJ/bit

Cost Model

- Off-chip Global Memory
  - LPDDR4, 4 pJ/bit
  - Higher-speed SDDR4, 20 pJ/bit
  - Bandwidth limited
    - Memory bandwidth off of an accelerator chip is limited to about 400 GB/s
    - Placing memories on interposers can give bandwidths up to 1 TB/s, but at the expense of limited capacity

[MICRON. System power calculators, 2019. https://tinyurl.com/y5cvl857]

Cost Model

- Local Communication
  - Increases linearly with distance at a rate of 100 fJ/bit-mm

Cost Model

- Global Communication
  - High-speed off-chip channels use SerDes that have an energy of about 10 pJ/bit

Cost Model

- Tools
  - DSENT
    https://github.com/mit-carbon/Graphite/tree/master/contrib/dsent/dsent-core
  - CACTI
    https://github.com/HewlettPackard/cacti
  - Ramulator

    https://github.com/CMU-SAFARI/ramulator

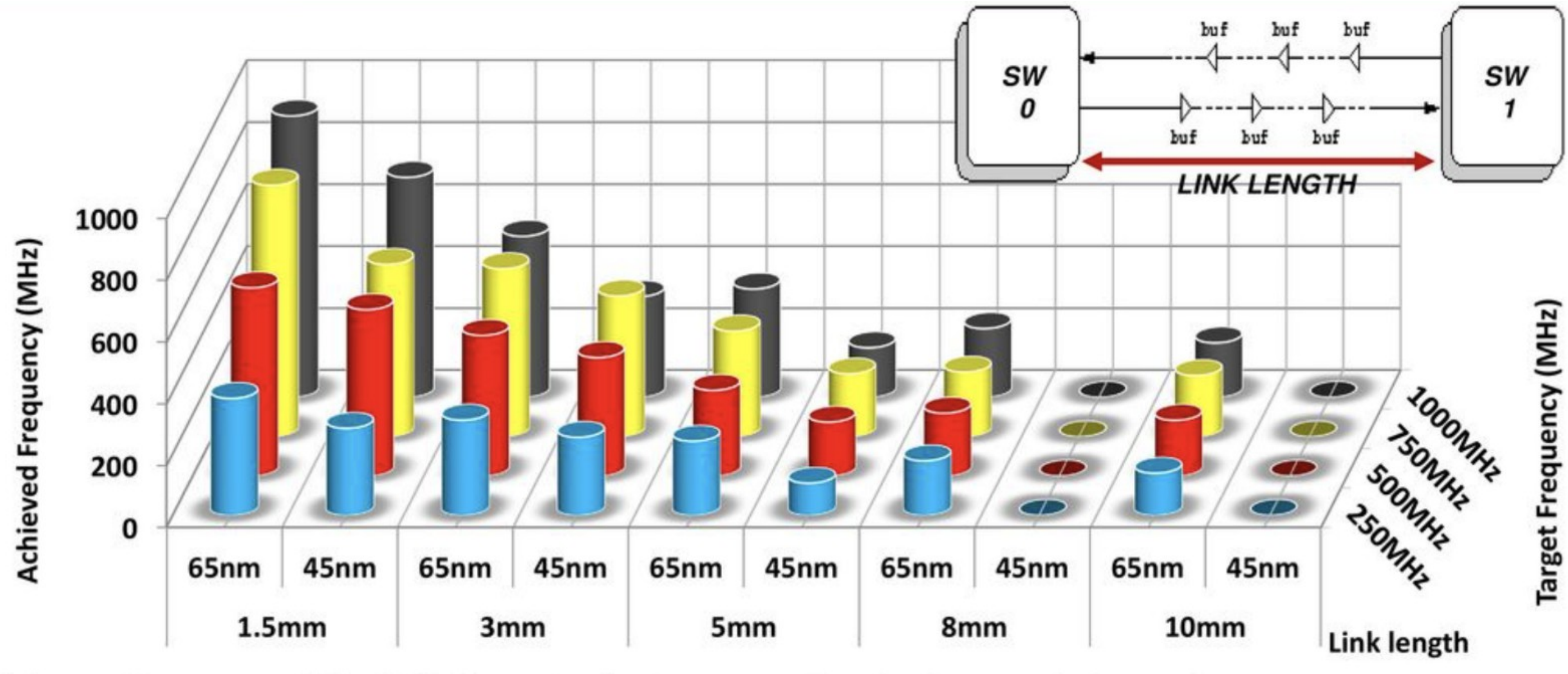
Agenda

- A bit of history
- Inefficiency in GP architectures
- Domain Specific Architectures
- Source of acceleration
- Cost models
- Communication issues

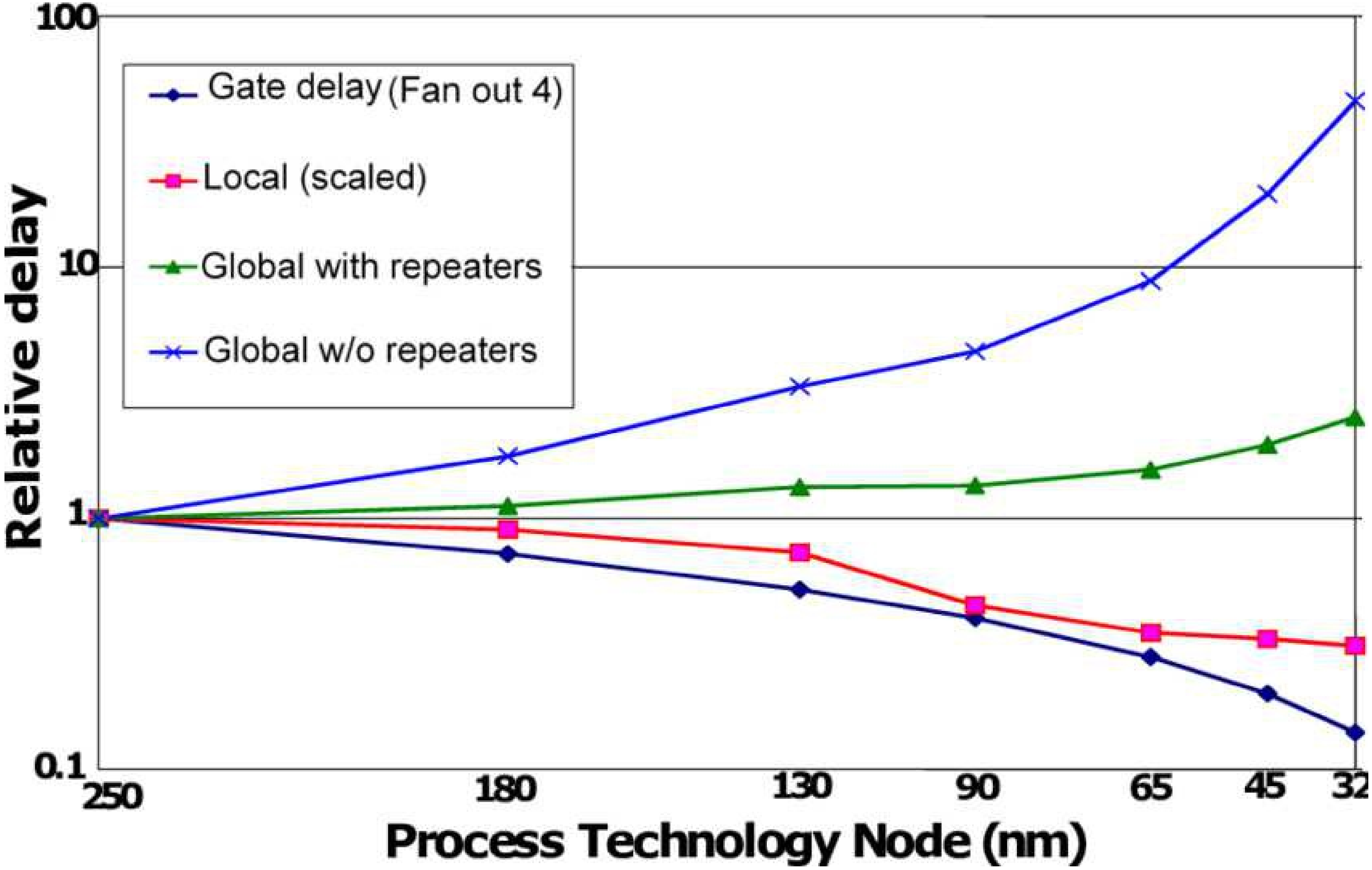
Communication Issues

- Logic and local memory energies scale linearly with technology
- Communication energy remains roughly constant!
- This nonuniform scaling makes communication even more critical in future systems

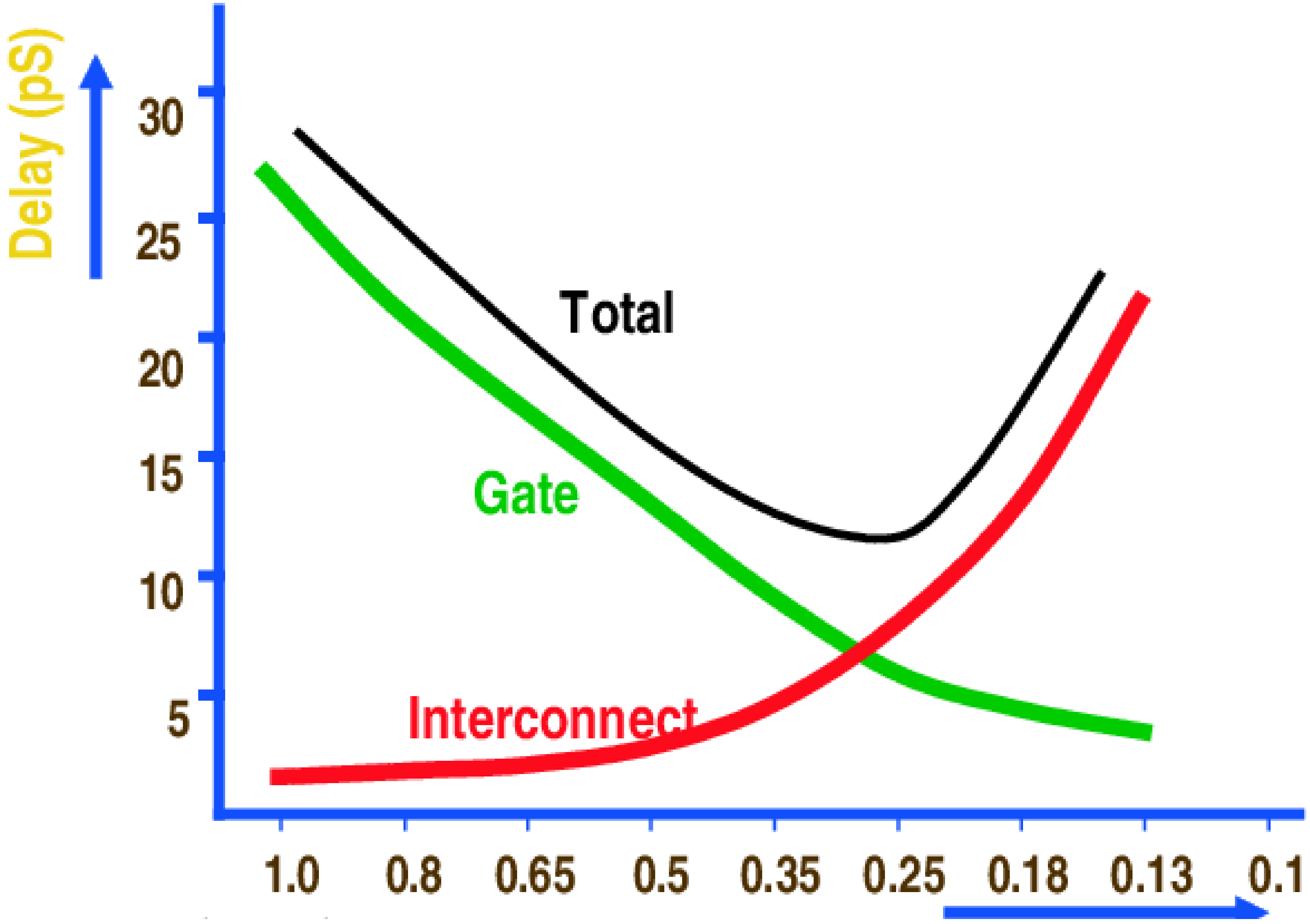
Link Performance



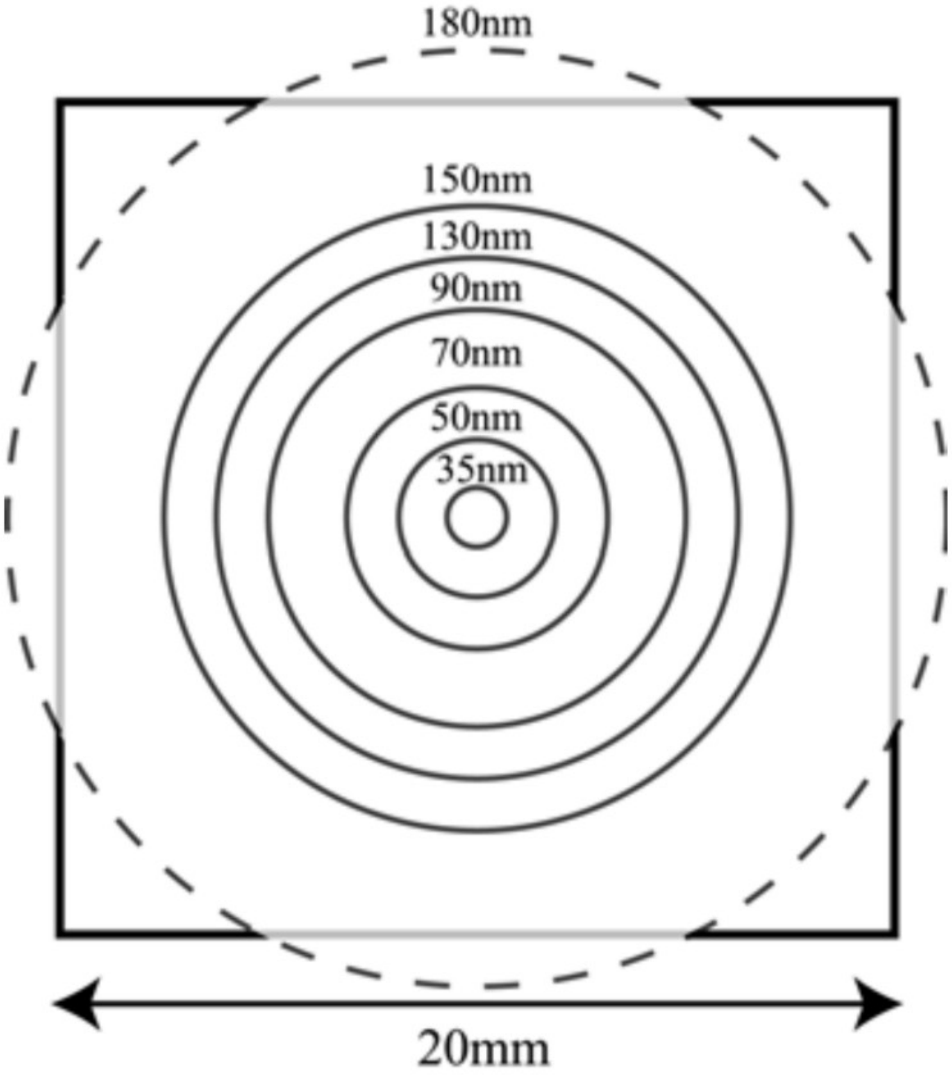
Interconnect Delay Bottleneck



Interconnect Delay Bottleneck



Interconnect Delay Bottleneck

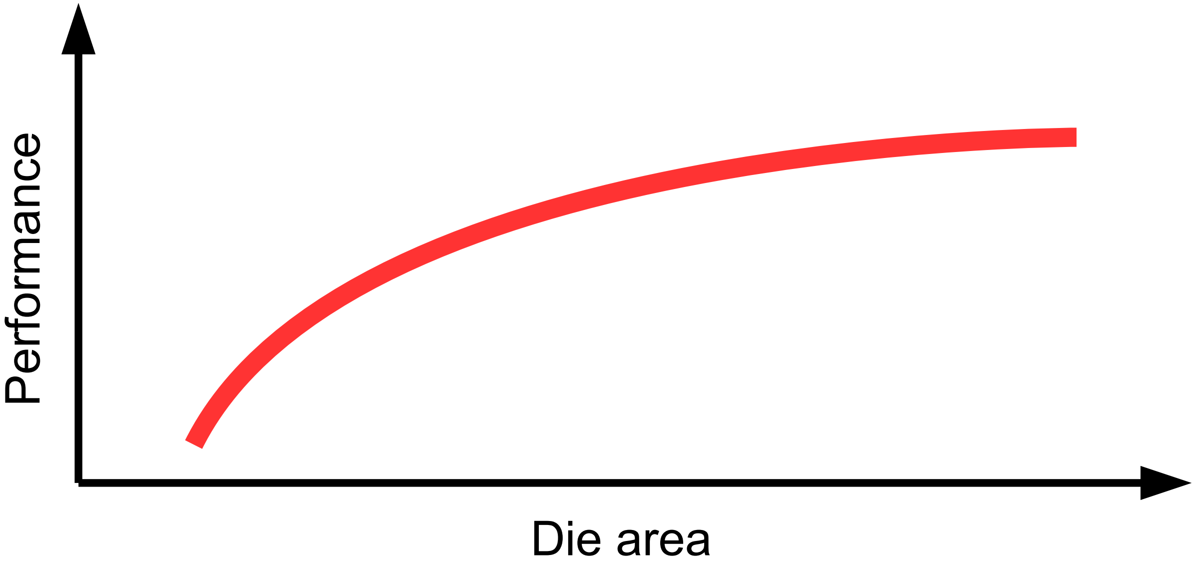


Fraction of chip reachable in one cycle with an 8FO4 clock period

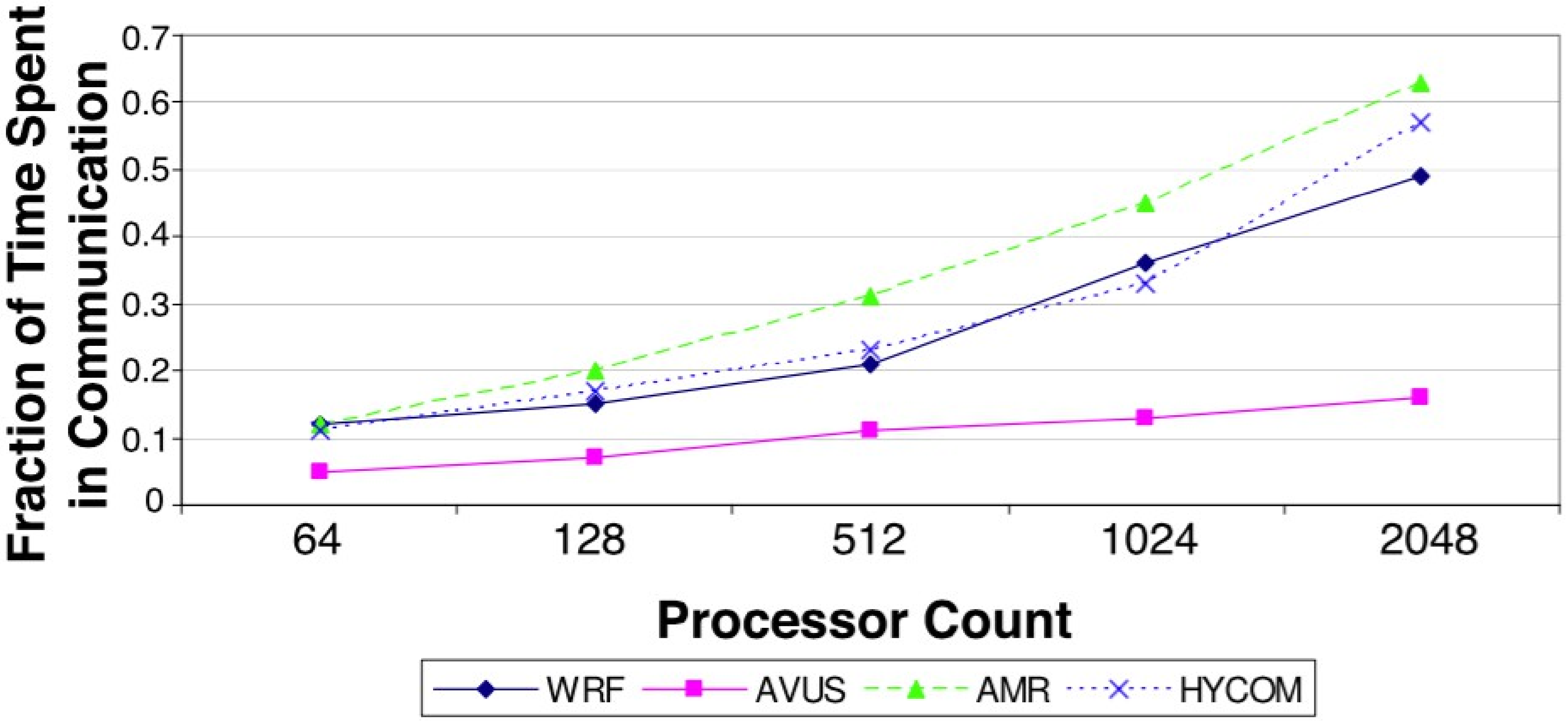
[S. W. Keckler *et al.*, "A wire-delay scalable microprocessor architecture for high performance systems," ISSCC 2003]

Uniprocessor Architecture Inefficiency

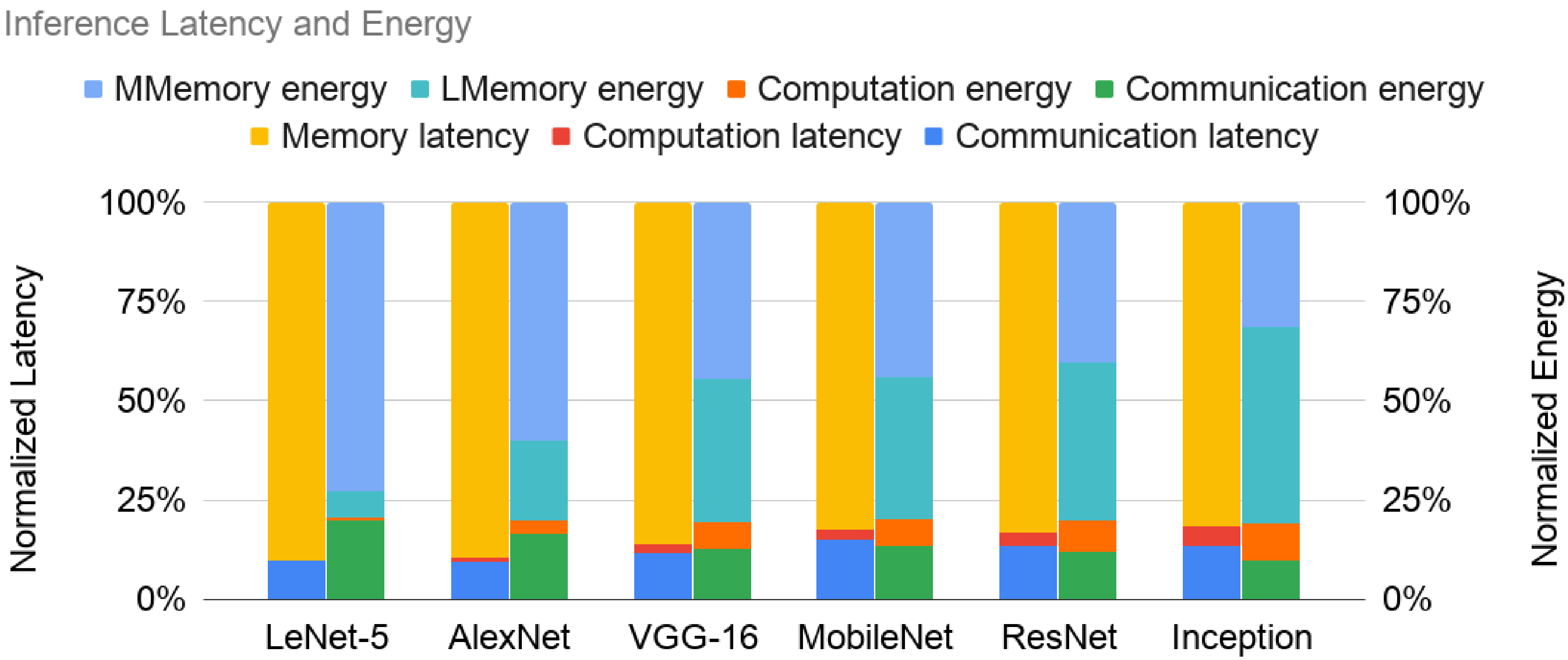
- Pollack's rule
  - New architectures take a lot more area for just a little more performance
  - **...global interconnect is part of this problem!**



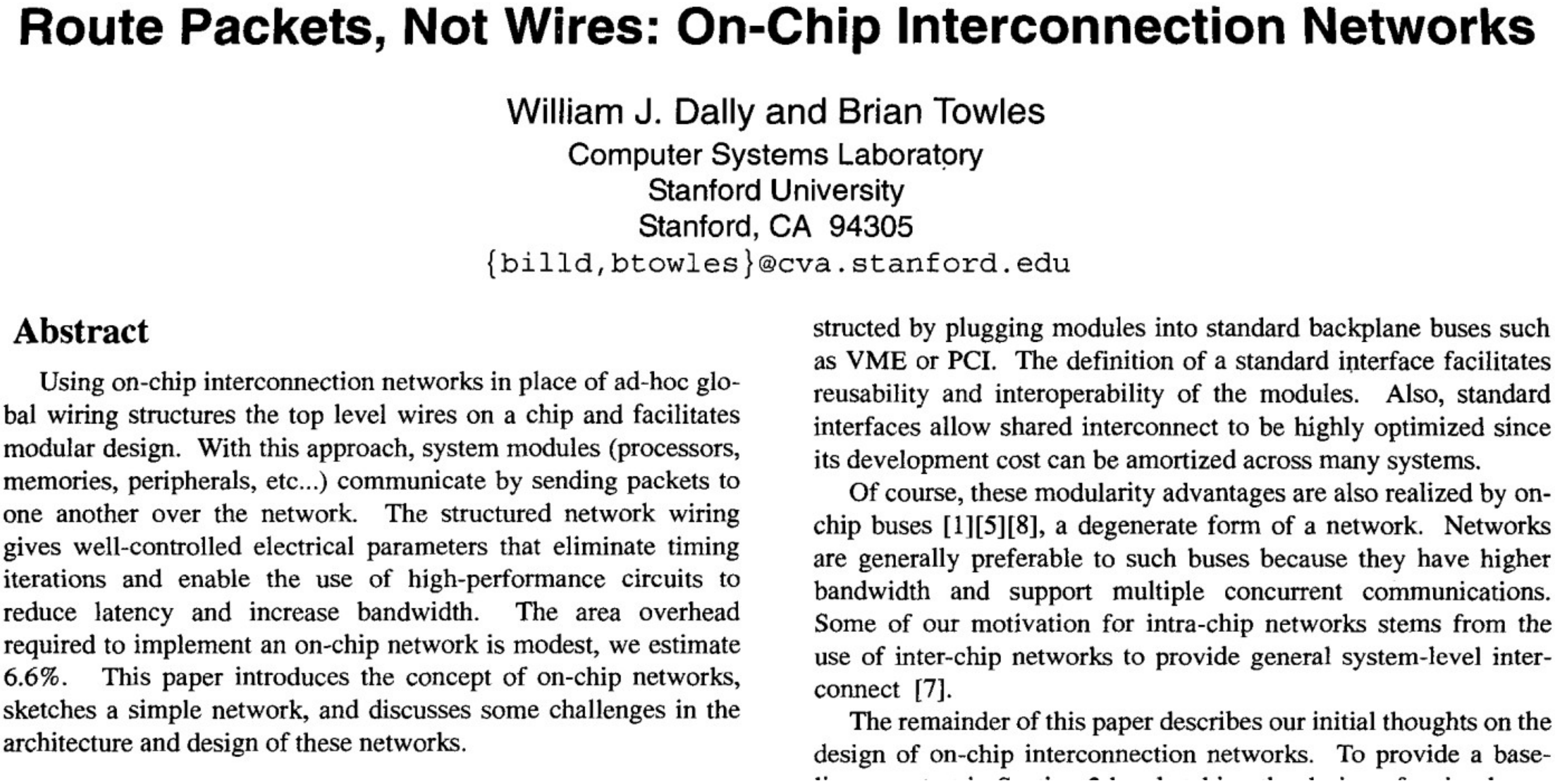
Communication Impact



Inference Latency/Energy



[M. Palesi, *et al.*, "Improving Inference Latency and Energy of Network-on-Chip based Convolutional Neural Networks through Weights Compression", IPDPS 2020]



Route Packets, Not Wires: On-Chip Interconnection Networks

William J. Dally and Brian Towles
Computer Systems Laboratory
Stanford University
Stanford, CA 94305
{billd,btowles}@cva.stanford.edu

Abstract

Using on-chip interconnection networks in place of ad-hoc global wiring structures the top level wires on a chip and facilitates modular design. With this approach, system modules (processors, memories, peripherals, etc...) communicate by sending packets to one another over the network. The structured network wiring gives well-controlled electrical parameters that eliminate timing iterations and enable the use of high-performance circuits to reduce latency and increase bandwidth. The area overhead required to implement an on-chip network is modest, we estimate 6.6%. This paper introduces the concept of on-chip networks, sketches a simple network, and discusses some challenges in the architecture and design of these networks.

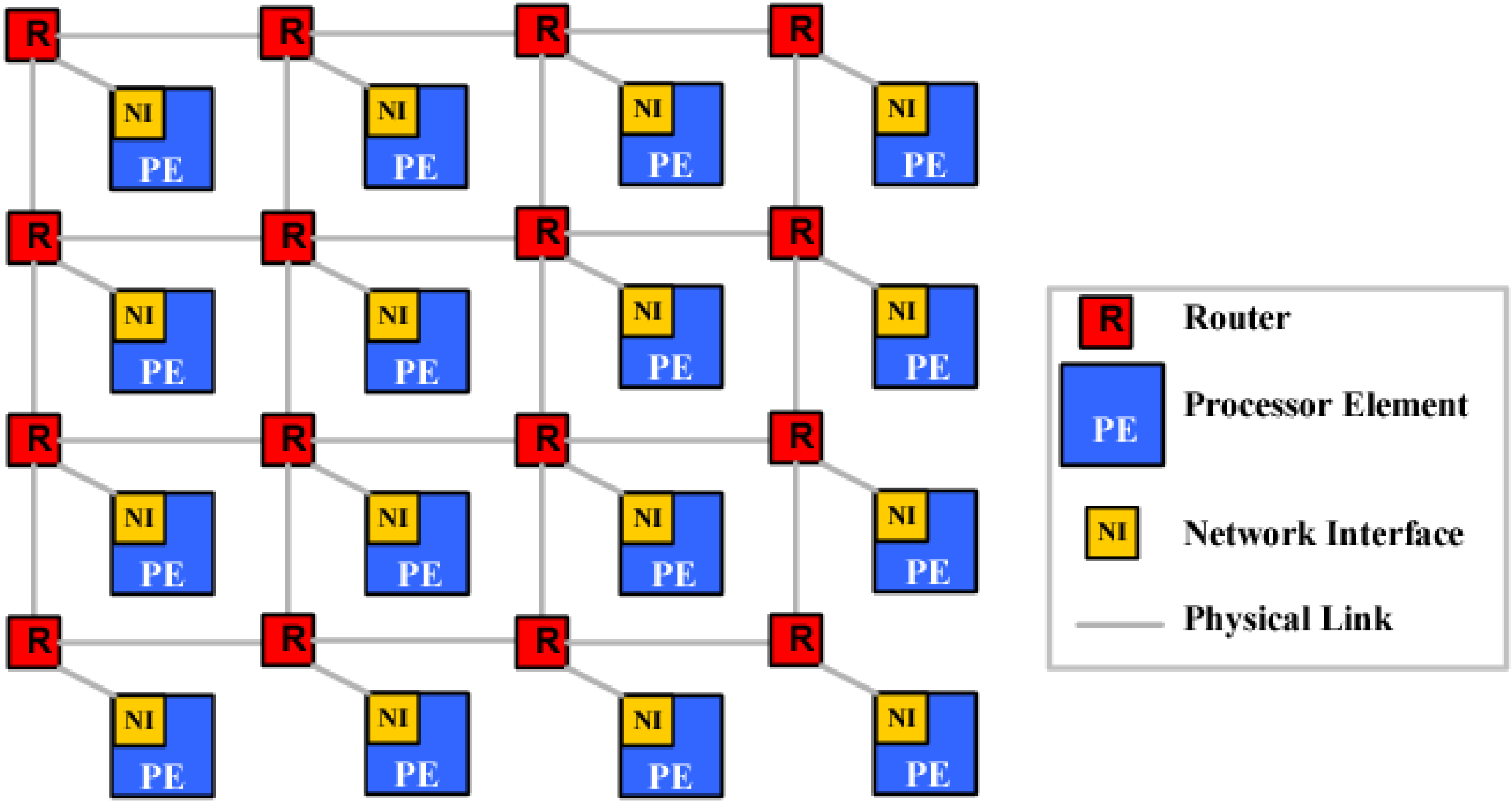
structed by plugging modules into standard backplane buses such as VME or PCI. The definition of a standard interface facilitates reusability and interoperability of the modules. Also, standard interfaces allow shared interconnect to be highly optimized since its development cost can be amortized across many systems.

Of course, these modularity advantages are also realized by on-chip buses [1][5][8], a degenerate form of a network. Networks are generally preferable to such buses because they have higher bandwidth and support multiple concurrent communications. Some of our motivation for intra-chip networks stems from the use of inter-chip networks to provide general system-level interconnect [7].

The remainder of this paper describes our initial thoughts on the design of on-chip interconnection networks. To provide a base-

[W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," DAC 2001]

Network-on-Chip Paradigm



Conclusions

- Technology related issues
  - End of Moore's law, Dennard Scaling, ...
- Turing Tariff
- Need for architectural innovations!
- A new golden age for computing architectures
  - Domain Specific Architectures
  - Domain Specific Languages