

Scaling Machine Learning Performance with Moore's Law

Kunle Olukotun
Stanford University

CS and EE

The DAWN Project

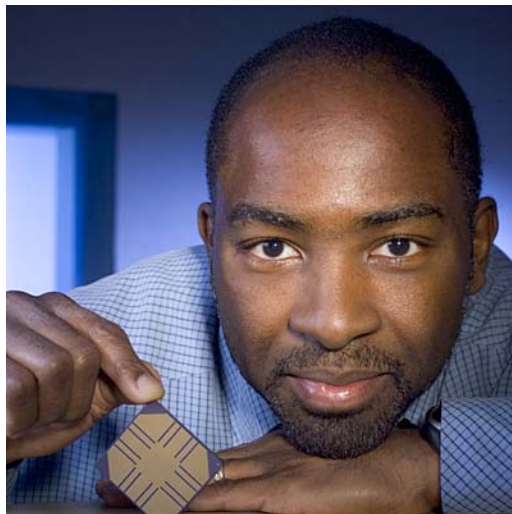
Peter Bailis
Streaming &
Databases



Chris Ré
MacArthur Genius
Databases + ML



Kunle
Olukotun
Father of Multicore
*Domain Specific
Languages*



Matei
Zaharia
Co-Creator of
Spark and Mesos



It's the Golden Era of Data *

- Incredible advances in image recognition, natural language processing, planning, info retrieval
- Society-scale impact: autonomous vehicles, personalized medicine, human trafficking
- No end in sight for advances in ML

***for the best-funded, best-trained engineering teams**

The DAWN Proposal

- What if *anyone* with domain expertise could build their own production-quality ML products?
 - Without a PhD in machine learning
 - Without being an expert in DB + systems
 - Without understanding the latest hardware

DAWN Goals

- Speed up machine learning by 100x
 - 1000x improvement in performance/watt
- Enable real-time/interactive ML on big data
 - Data center
 - Mobile
- Full stack approach:
Algorithms + PL/Compilers + Hardware

Approach 1

Hardware aware ML algorithms
that provide significant
improvements in efficiency

Incremental Gradient Methods

Lots of machine learning can be written as:

$$\min_x \sum_{i=1}^N f(x, y_i)$$

Loss function
Data
Model

N (number of y_i s, data) typically in the billions
E.g.: Classification, Recommendation, Deep Learning

Solving large-scale problems:

Stochastic Gradient Descent (SGD)

$$x^{k+1} = x^k - \alpha N \nabla f(x^k, y_j)$$

Select one term, j , and estimate gradient

Billions of tiny sequential iterations: how to parallelize?

Iterative Machine Learning: Two Kinds of Efficiency

- **Statistical efficiency:** **how many iterations** do we need to get a quality result?
 - Depends on **the problem** and implementation
- **Hardware efficiency:** **how long** does it take to run each iteration?
 - Depends on **the hardware** and implementation

trade off hardware and statistical efficiency
to maximize performance

Everything You Learned About Parallel
Computing is Wrong for Machine
Learning!

The HOGWILD! Strategy

- Run multiple worker threads **without locks**
 - Threads work together and modify a single copy of the model creating **many data races**
 - Improves **hardware efficiency**
- What about the **race conditions**?
 - Races introduce errors we can model as **noise**
 - Below existing noise floor → negligible effect on **statistical efficiency**

Applications of HOGWILD!

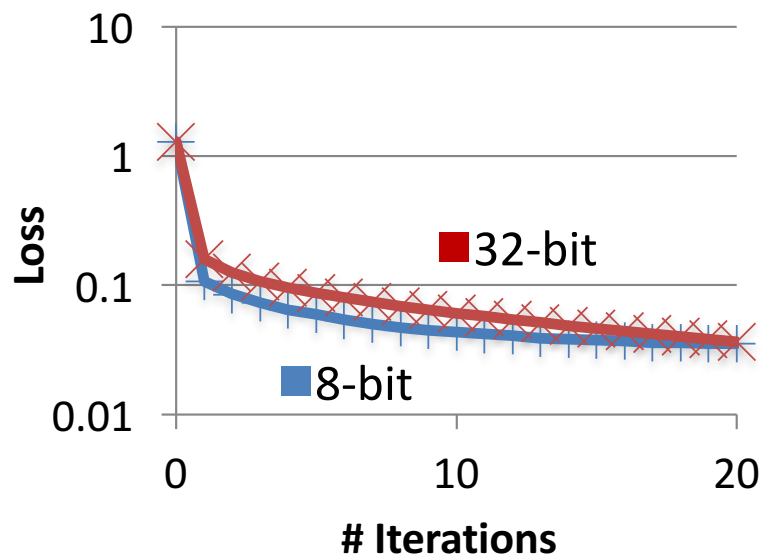
- **Optimization**: stochastic gradient descent
 - Theory: [Niu, Recht, Ré, Wright: *NIPS 2011*]
 - HOGWILD! SGD is a well-tested, popular technique
 - **Near-linear speedups** in terms of # of threads
 - Used in many real systems **in industry**
- **Sampling & inference**: Gibbs sampling
 - Theory: [De Sa, Olukotun, Ré: *ICML 2016, ICML Best Paper*]

The BUCKWILD! Strategy

- Use **8- or 16-bit fixed point numbers** for computing rather than 32-bit floating point
 - Fewer bits of data → better **hardware efficiency**
 - Stochastic rounding → keeps **statistical efficiency**
 - Theory: [De Sa, Zhang, Olukotun, Ré: *NIPS 2015*]
- What about the **quantization error**?
 - Round-off error can be modeled as **noise**.
 - Below existing noise floor → negligible effect on **statistical efficiency**

BUCKWILD! Statistical vs. Hardware Efficiency

Same statistical efficiency



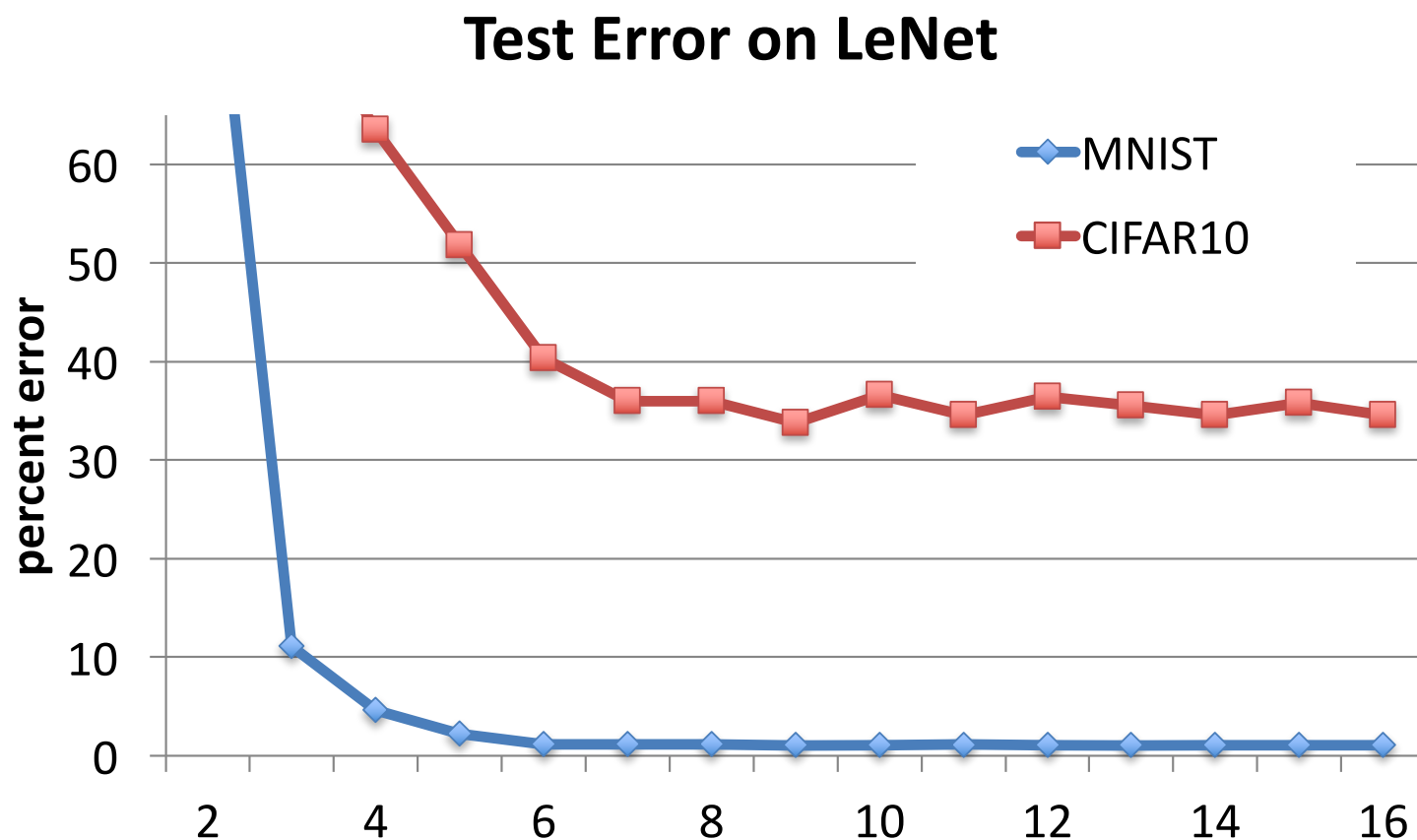
Logistic Regression using SGD

Improved hardware efficiency

- 8-bit gives about **3x** speed up!
- Even lower precision is possible
- Good for specialized or reconfigurable HW?

BUCKWILD! has same **statistical efficiency** with greater **hardware efficiency**

Low Precision for CNN



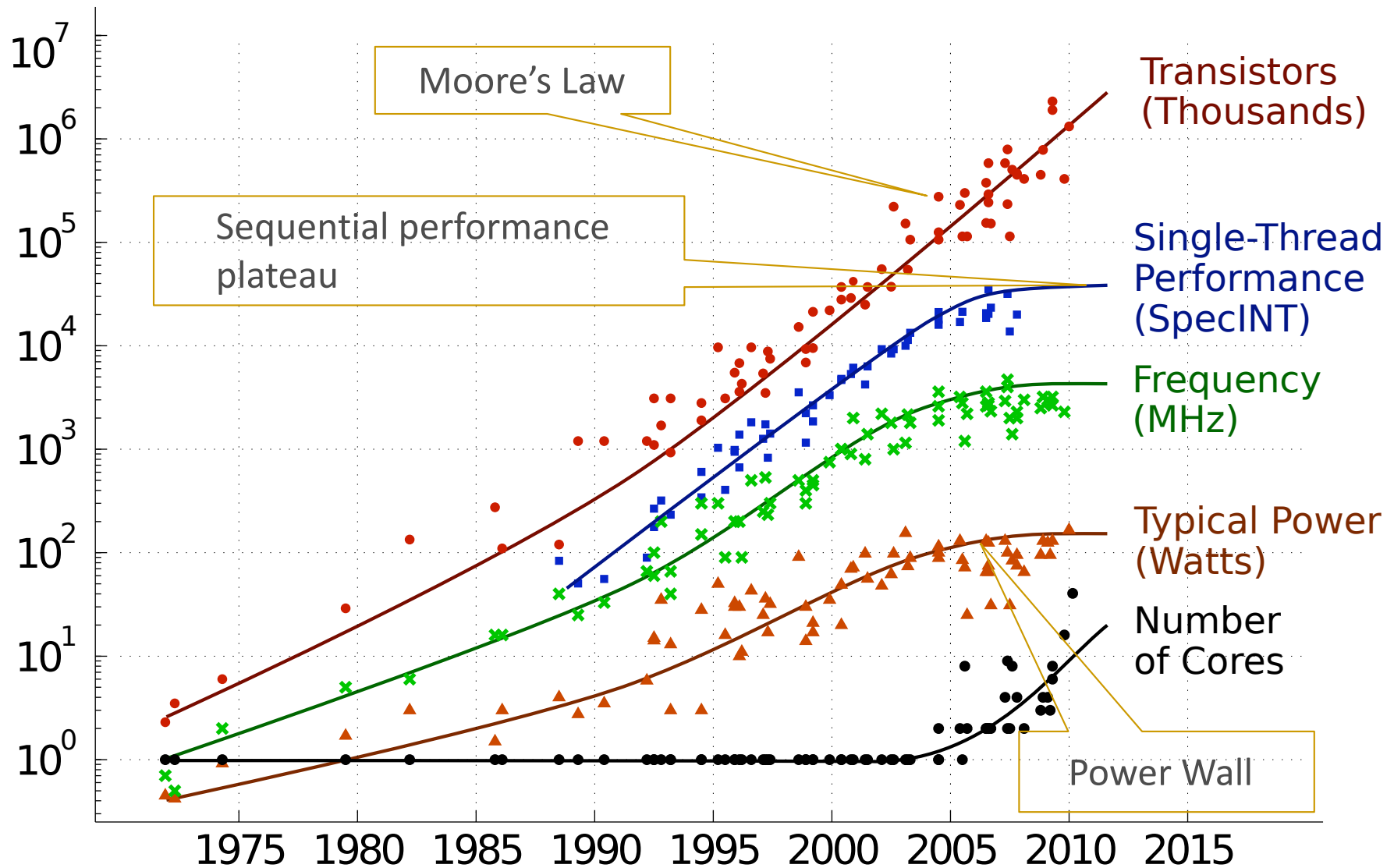
We can go down below 8-bits of precision without sacrificing accuracy

Relax, It's Only Machine Learning

- Relax synchronization: data races are better
- Relax precision: small integers are better
- Relax coherence: incoherence is better

Better hardware efficiency
with negligible impact on statistical efficiency

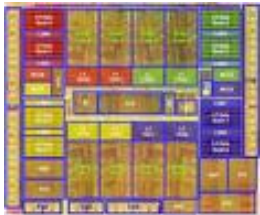
Microprocessor Trends



Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten

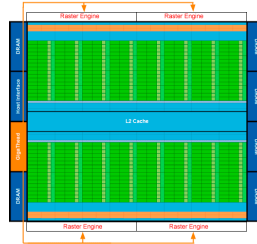
Accelerators to the Rescue

10s of cores



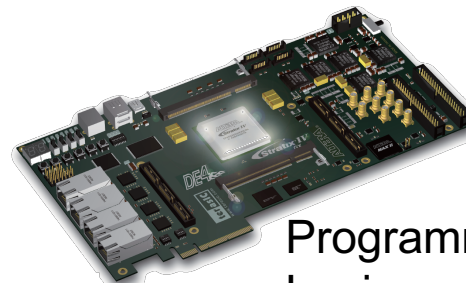
Multicore
Multi-socket

> 1 TFLOPS



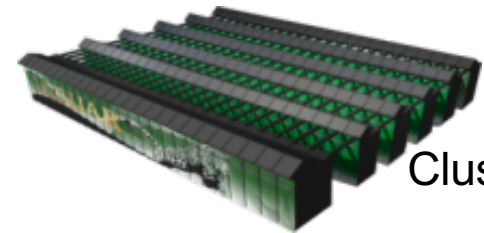
Graphics
Processing
Unit (GPU)

Accelerators



Programmable
Logic

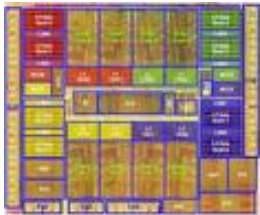
1000s of nodes



Cluster

Needs Expert Parallel Programming

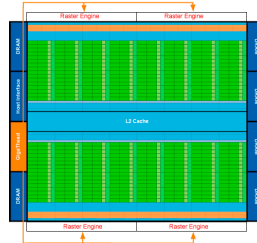
10s of cores



Multicore CPU
Multi-socket

Threads
OpenMP

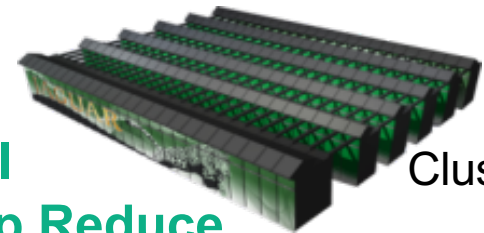
> 1 TFLOPS



Graphics
Processing
Unit (GPU)

CUDA
OpenCL

1000s of nodes

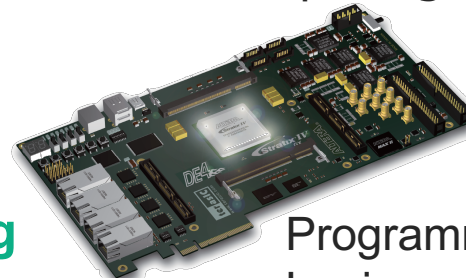


MPI
Map Reduce

Cluster

MPI: Message Passing Interface

Custom computing



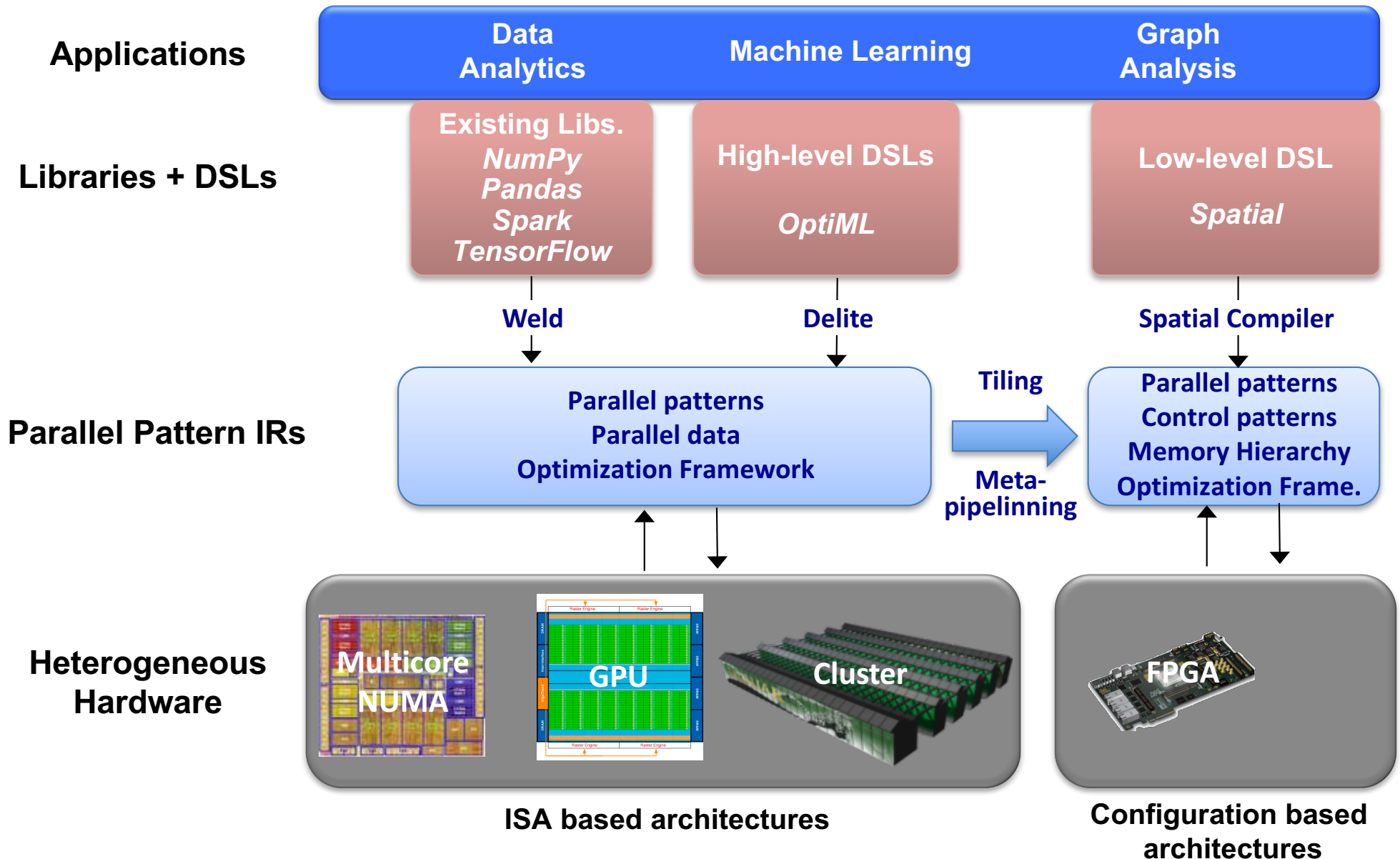
Verilog
VHDL

Programmable
Logic

Approach 2

Write one machine learning program and run it efficiently on all these architectures

Scaling Machine Learning with Moore's Law



OptiML: Overview

- Provides a familiar (MATLAB-like) language and API for writing ML applications
 - Ex. `val c = a * b` (`a, b` are `Matrix[Double]`)
- Implicitly parallel data structures
 - Base types
 - `Vector[T]`, `Matrix[T]`, `Graph[V,E]`, `Stream[T]`
 - Subtypes
 - `TrainingSet`, `IndexVector`, `Image`, ...
- Implicitly parallel control structures
 - `sum{...}`, `(0::end) {...}`, `gradient { ... }`, `untilconverged { ... }`
 - Allow anonymous functions with restricted semantics to be passed as arguments of the control structures

K-means Clustering in OptiM

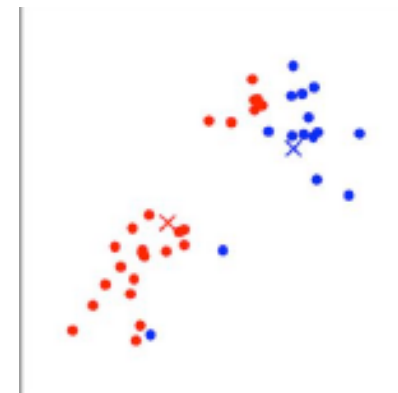
assign each sample to the closest mean

```
untilconverged(kMeans, tol){kMeans =>
  val clusters = samples.groupRowsBy { sample =>
    kMeans.mapRows(mean => dist(sample, mean)).minIndex
  }
  val newKmeans = clusters.map(e => e.sum / e.count)
  newKmeans
}
```

calculate distances to current means

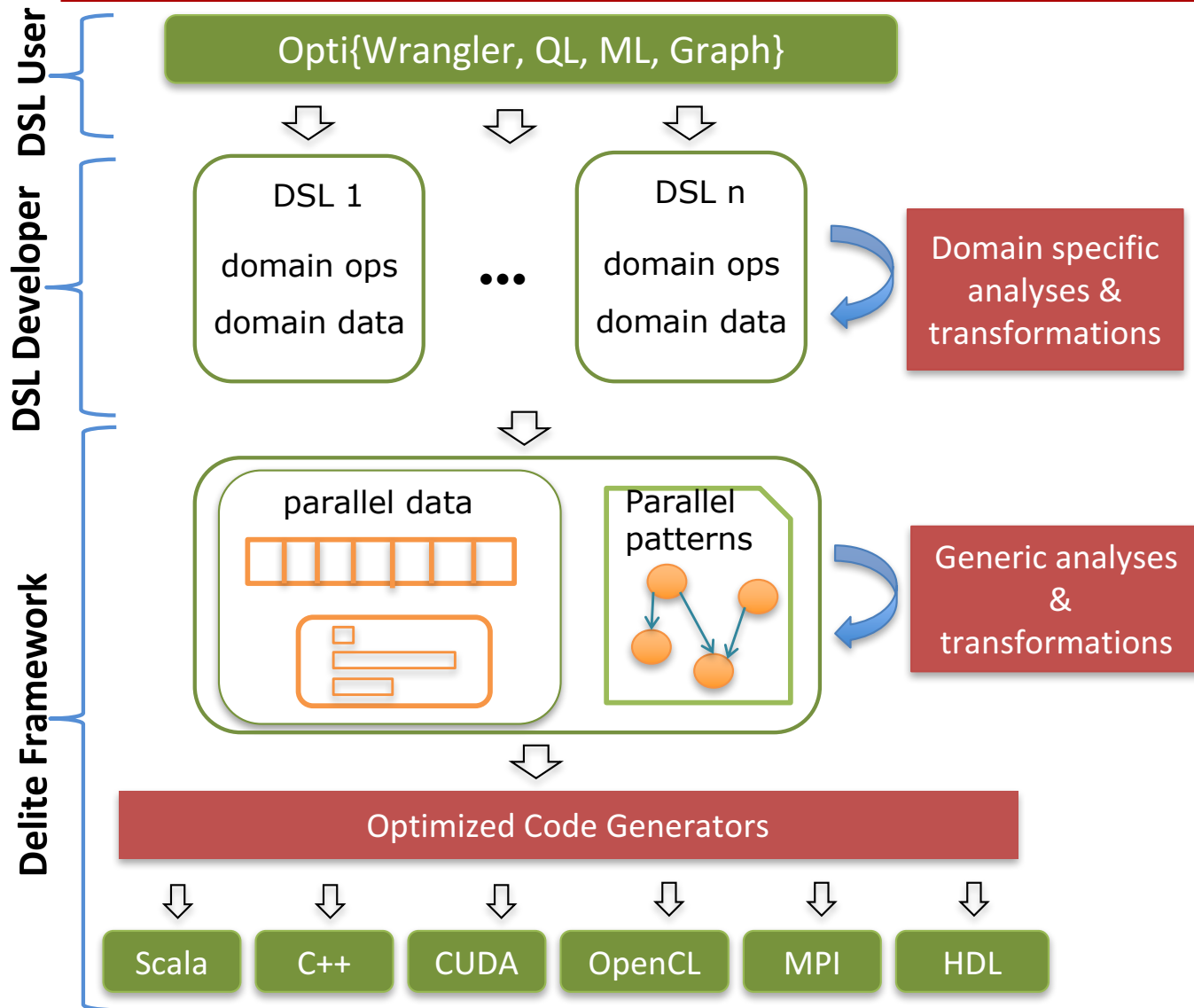
move each cluster centroid to the mean of the points assigned to it

- No explicit map-reduce, no key-value pairs
- No distributed data structures (e.g. RDDs)
- No annotations for hardware design
- Efficient multicore and GPU execution
- Efficient hardware implementation



Delite Overview

K. J. Brown et al., "A heterogeneous parallel framework for domain-specific languages," *PACT, 2011*.



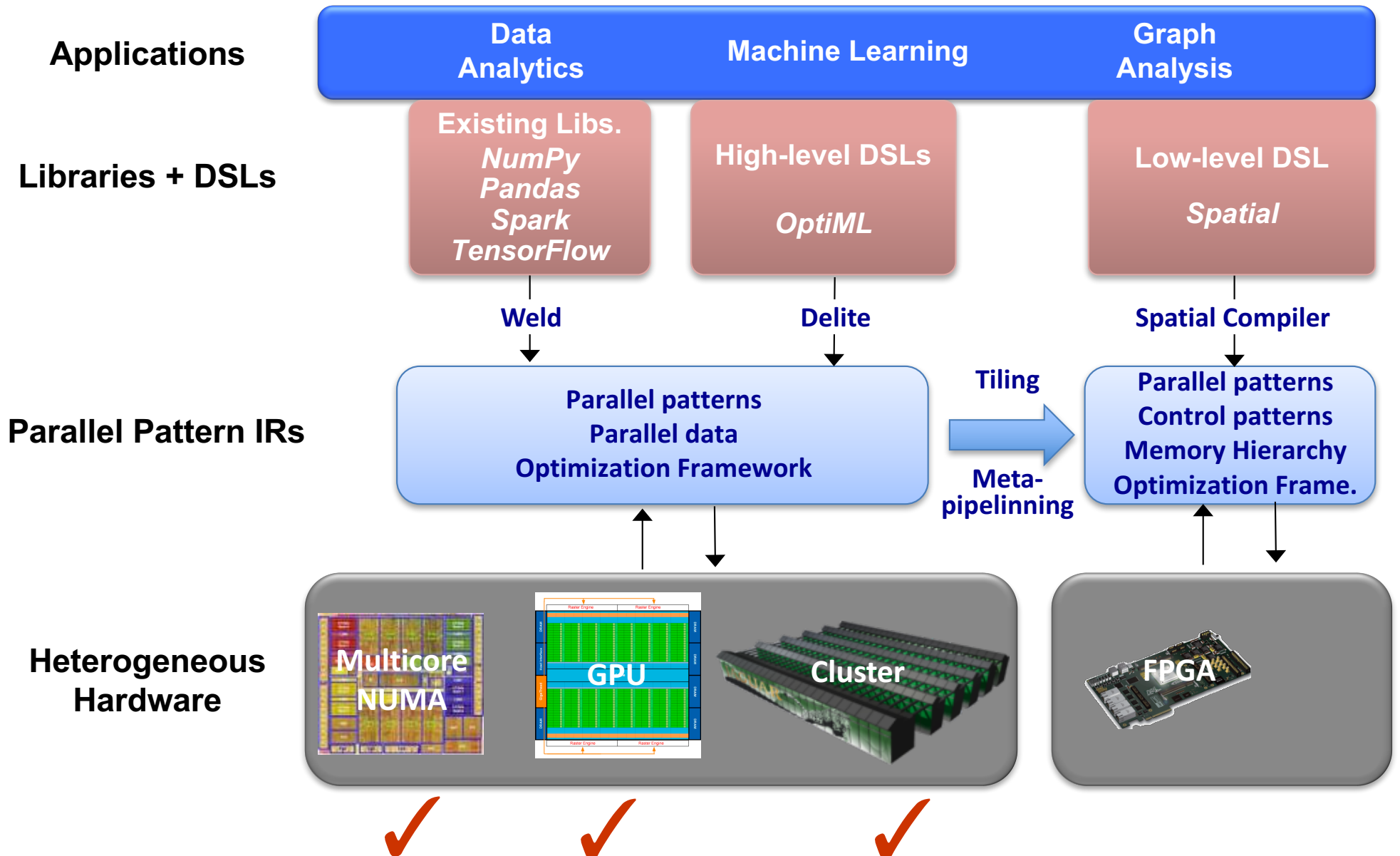
Key elements

- DSLs embedded in Scala
- IR created using type-directed staging
- Domain specific optimization
- General parallelism and locality optimizations
- Optimized mapping to HW targets

Parallel Patterns

- Data-parallel functional operators
- Capture the common loop-based programming patterns
 - All take functions as arguments and abstract over the actual loop implementation (control flow)
- Generate a new collection by transforming an existing one
 - Map, Filter, FlatMap
- Combine collection elements into a single value
 - Reduce, Fold
- Reorder elements in a collection
 - GroupBy, Sort
- Combine multiple collections into one collection
 - ZipWith, Join

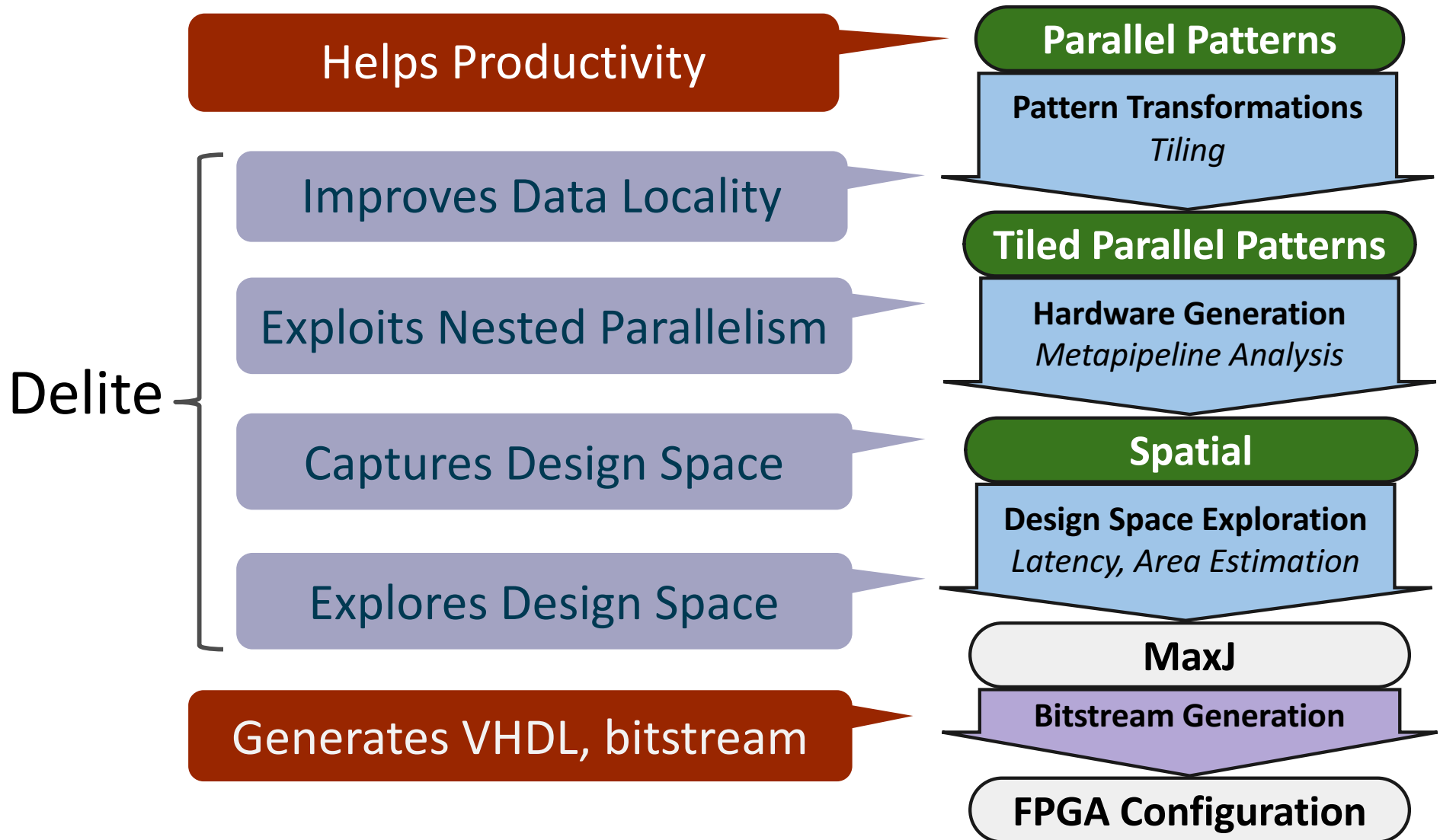
Scaling Machine Learning with Moore's Law



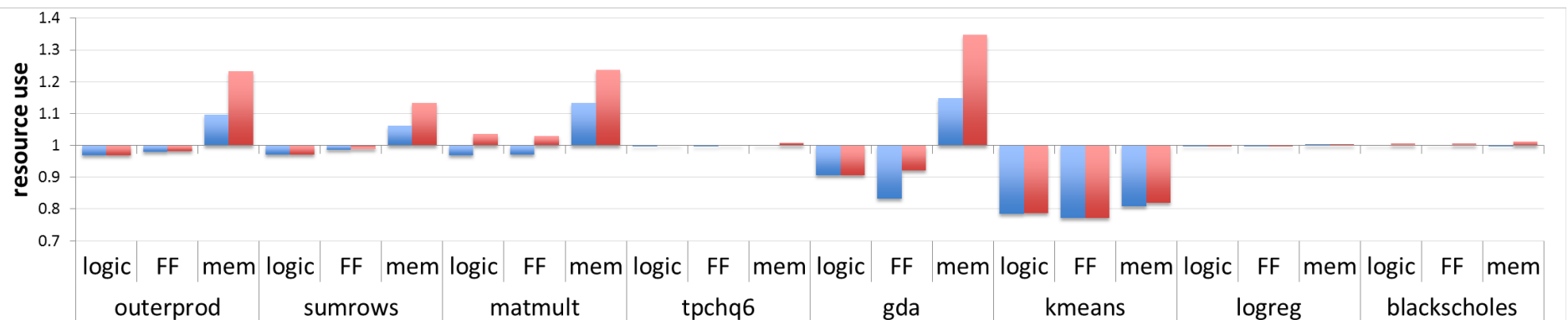
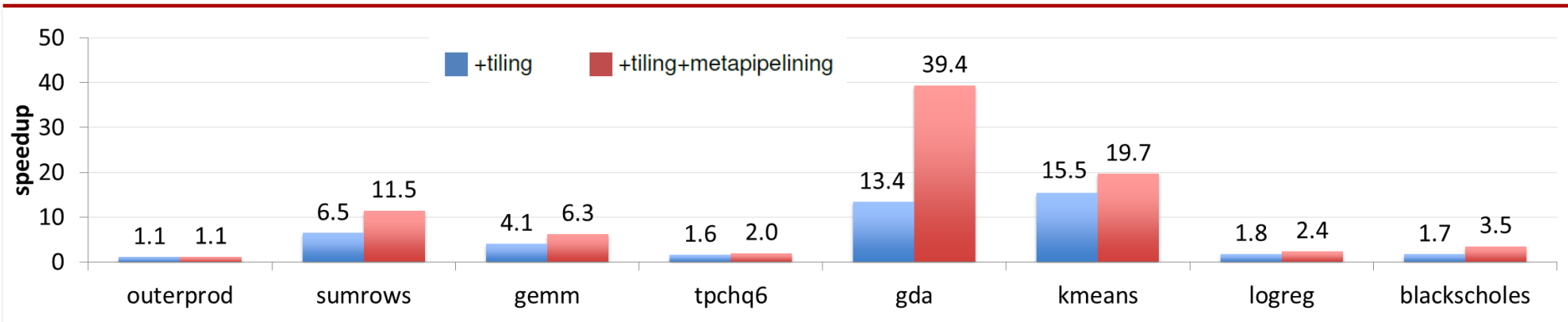
FPGA Accelerators?

- **FPGAs based accelerators**
 - Recent commercial interest from AWS, Baidu, Microsoft, and Intel
 - Key advantage: Performance, Performance/Watt
 - Key disadvantage: lousy programming model
- **Verilog and VHDL poor match for software developers**
 - High quality designs
- **High level synthesis (HLS) tools with C interface**
 - Medium/low quality designs
 - Need architectural knowledge to build good accelerators
 - Not enough information in compiler IR to perform access pattern and data layout optimizations
 - Cannot synthesize complex data paths with nested parallelism

Optimized Approach to HW Generation

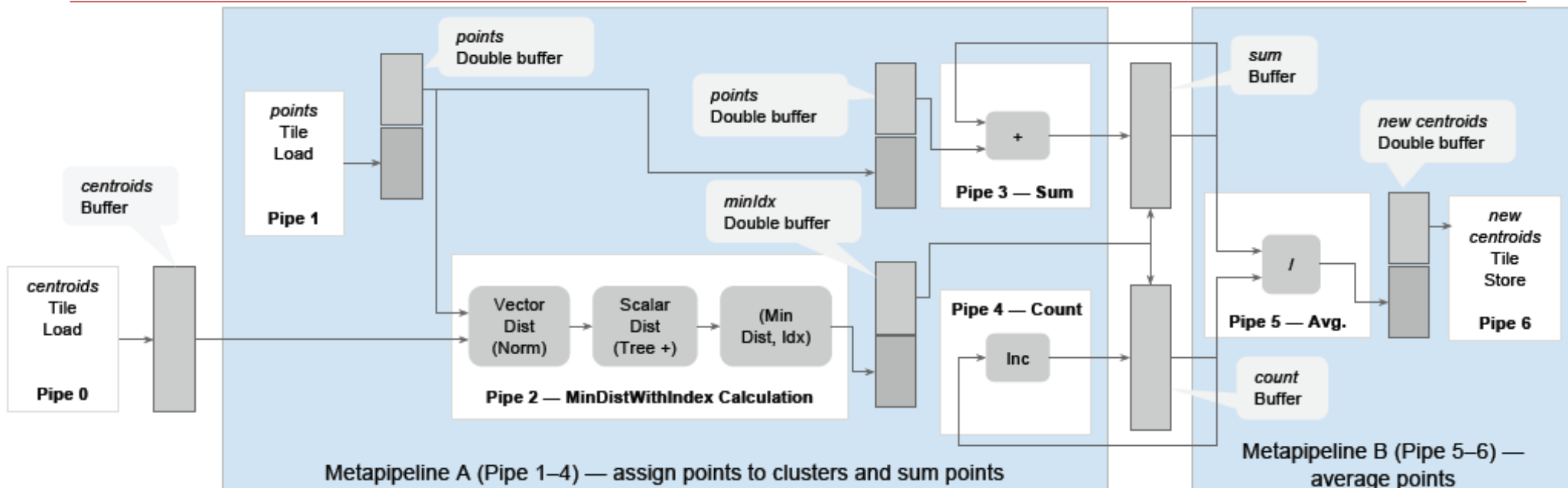


How Beneficial is Tiling and Metapipelining?



- Speedup with tiling: up to **15.5x**
- Speedup with tiling + metapipelining: up to **39.4x**
- Minimal (often positive!) impact on resource usage
 - Tiled designs have fewer off-chip data loaders and storers

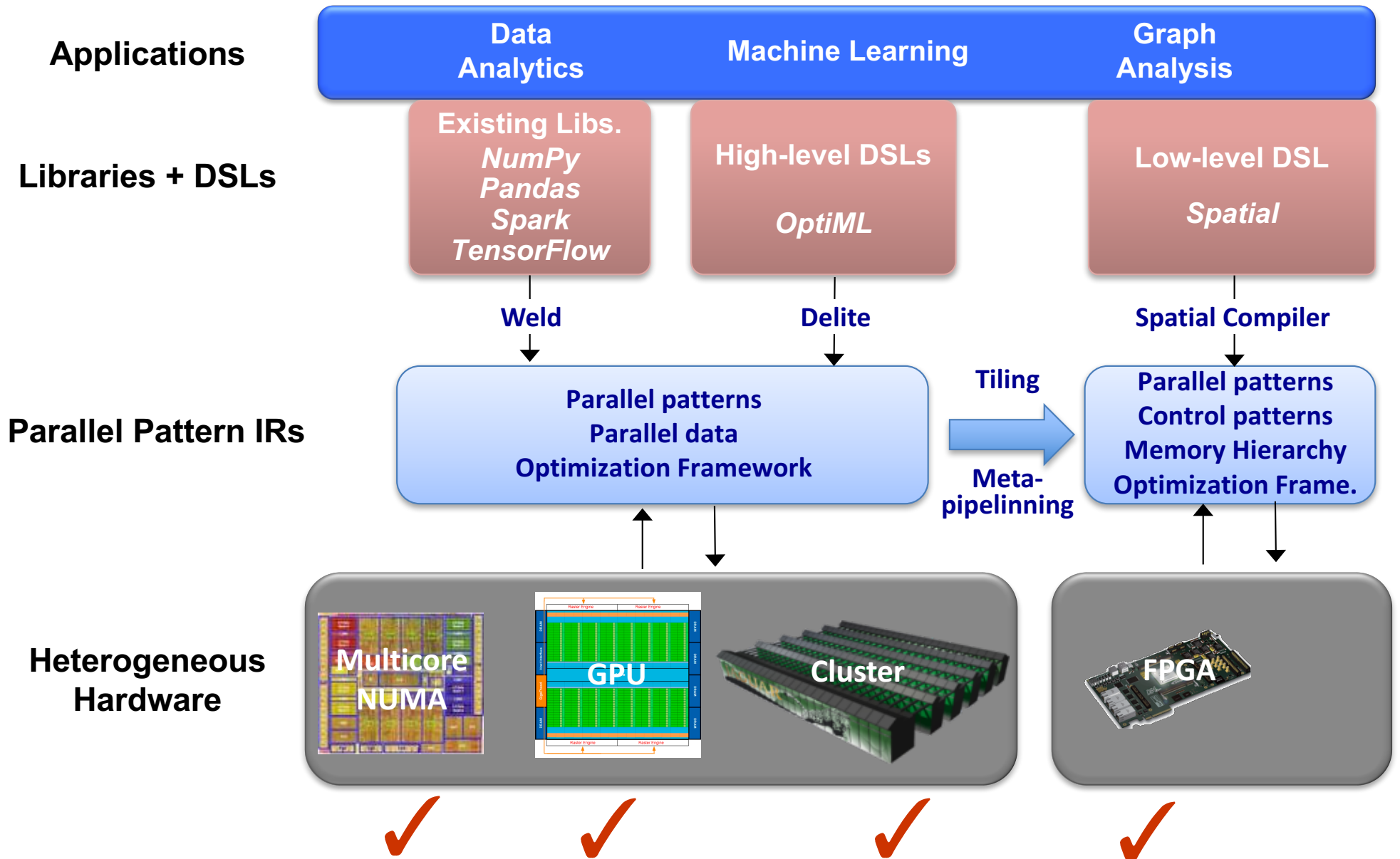
Generated *k*-means Hardware



■ High quality hardware design

- Hardware similar to Hussain et al. *Adapt. HW & Syst. 2011*
 - “FPGA implementation of *k*-means algorithm for bioinformatics application”
 - Implements a fixed number of clusters and a small input dataset
- Tiling analysis automatically generates buffers and tile load units to handle arbitrarily sized data
- Parallelizes across centroids and vectorizes the point distance calculations

Scaling Machine Learning with Moore's Law Today



Specialized Hardware for Machine Learning

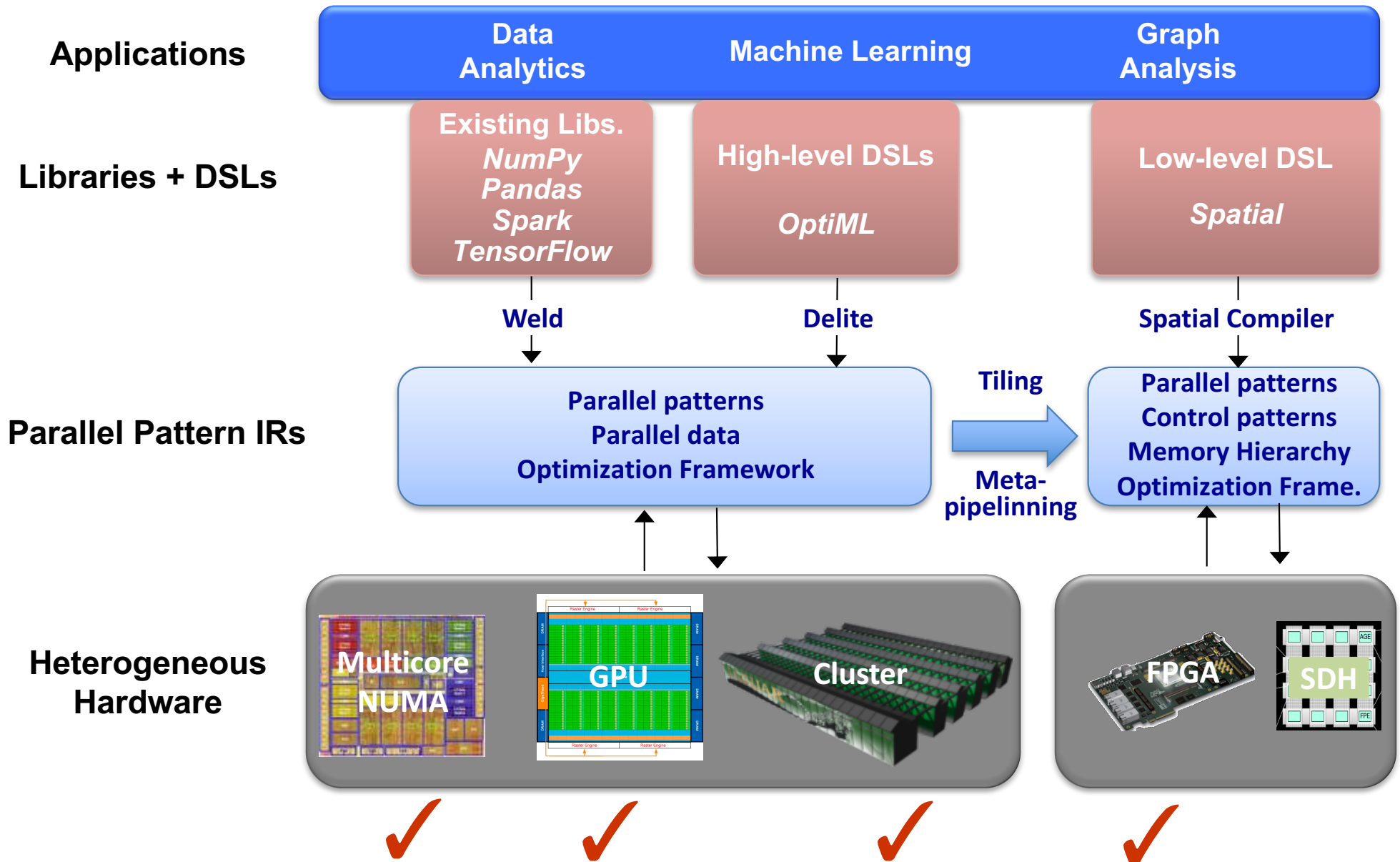





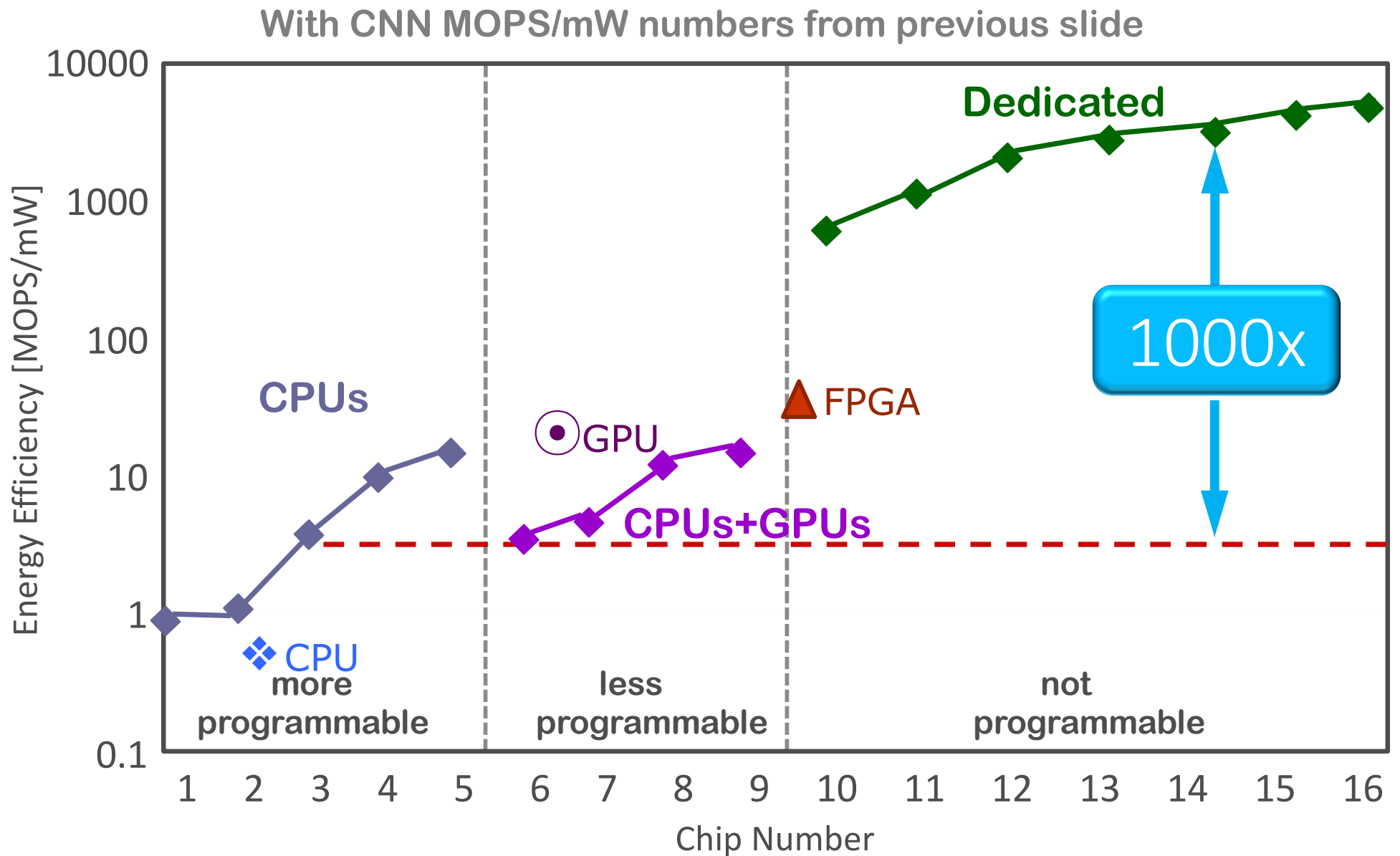
Image Classification using CNNs

Platform	Library/OS	ImageNet 1K Inference Throughput	Peak TFLOPs	Effective TFLOPs	Estimated Peak Power for CNN Computation	Estimated MOPS/mW (assuming peak power)
CPU  16-core, 2-socket Xeon E5-2450, 2.1GHz	Caffe + Intel MKL Ubuntu 14.04.1	106 images/s	0.54T	0.148T (27%)	~225W	~0.6
FPGA  Arria 10 GX1150	Windows Server 2012	369 images/s ~880 images/s	1.366T	0.51 T (38%) ~1.2T (89%)	~37W ~40W	~12.8 ~30.6
GPU  NervanaSys-32 on NVIDIA Titan X	NervanaSys-32 on Ubuntu 14.0.4	4129 images/s	6.1T	5.75T (94%)	~250W	~23.0

Source: Accelerating Large-scale data center services, Andrew Putnam, Microsoft

*Projected results

Programmability vs. Energy Efficiency



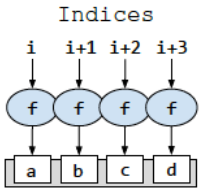
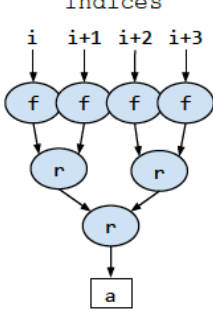
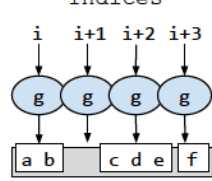
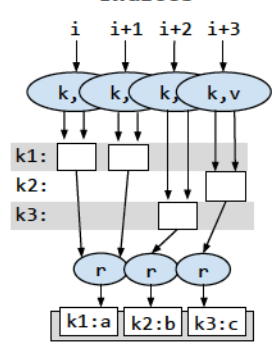
Approach 3

Design hardware that provides
programmability of CPUs and
energy efficiency of ASICs

Software Defined Hardware (SDH)

- All(Just) the advantages of conventional accelerators
 - Flexibility of FPGAs
 - Programmability of GPUs
 - Efficiency of ASICs
- SDH Goals
 - 100x performance/Watt vs. CPU
 - 10x performance/Watt vs. FPGAs/GPUs
 - 1000x programmability vs. FPGAs
- SDH key elements
 - Configurability
 - Specialization
 - *Spatial* programming language

Parallel Patterns → Reconfigurable HW

	Map	Fold	FlatMap	HashReduce
	<p>Indices</p> 	<p>Indices</p> 	<p>Indices</p> 	<p>Indices</p> 
Compute	Pipeline compute SIMD lanes			
On-Chip Memory	Distributed register files n-buffers			
	Banked scratchpads		Banked FIFOs	CAM (sparse) Scratchpad (dense)
Off-Chip Memory	Burst access for sequential reads/writes Gather/Scatter access for random reads/writes			
Interconnect		Reduction tree	Coalescing	
Control	Counter chains for generating indices Reconfigurable control for metapipelining			

Plasticine vs. FPGA

Benchmark	Speedup	Normalized Power	Normalized Perf/W
Black-Scholes (dense)	5.3	3.0	1.8
TPCH-Query 6 (dense)	8.9	3.9	2.3
GEMM (dense)	54.6	3.3	16.6
GDA (dense)	28.5	3.2	9.0
SGD (sparse)	84.9	3.3	25.8
Kmeans (mixed)	226.4	3.5	64.2
CNN (dense)	40.5	4.0	10.1
SMDV (mixed)	73.8	3.9	18.8
PageRank (sparse)	54.1	3.8	14.1
BFS (sparse)	28.0	3.8	7.3
Average	60.7	3.6	17.0

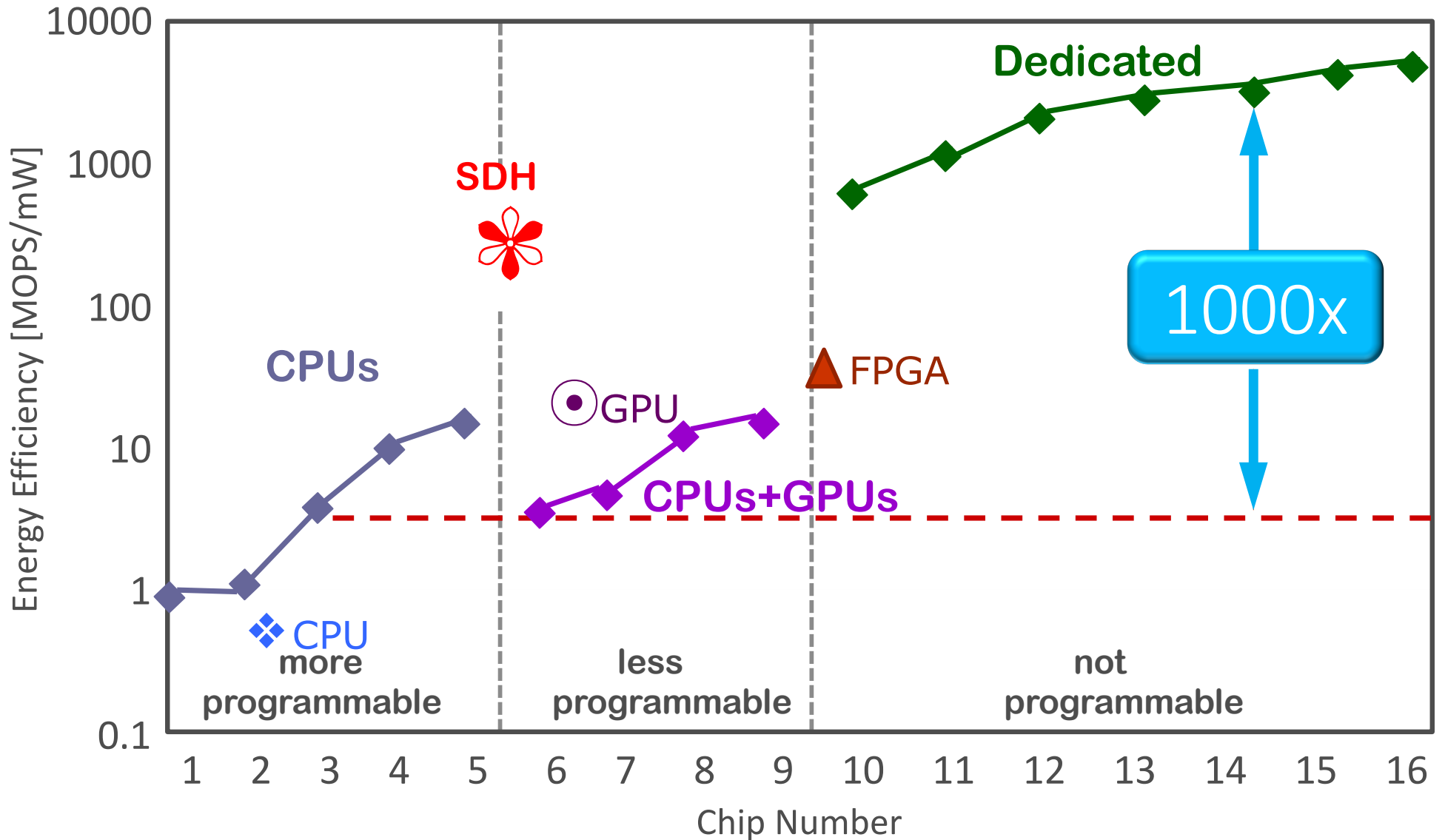
- In 28 nm technology: 1 GHz SDH (coarse grain) vs. 150 MHz FPGA (fine grain)
- 64 PCU/MCU, cycle-accurate simulation with same DRAM bandwidth
- More FLOPS with 8K dedicated FPUs in PCU
- Scatter-gather in DI benefits sparse applications
- Support for stencil applications

Accelerator Performance and Power Comparison for CNNs

Platform	Phi 7250	Titan X	Tegra K1	A-Eye	Arria-10	DaDi-anNao	EIE	Eye-riss
Year	2016	2015	2014	2015	2015	2014	2015	2016
Platform Type	CPU	GPU	mGPU	FPGA	FPGA	ASIC	ASIC	ASIC
Technology	14nm	28nm	28nm	-	20nm	28nm	45nm	65nm
Throughput (GOP/s)	3046	6100	365	188	1200	5580	102	84
Power (W)	215	250	8.0	9.63	40	15.97	0.59	0.450
Efficiency (MOP/s/mW)	14.2	24.4	45.6	19.5	30.0	349.4	172.9	186.7

Plasticine = 247 MOP/s/mW in 28nm

Software Defined Hardware



We Can Have It All!

- Power
- Performance
- Programmability
- Portability

Algorithms
(Hogwild!, Buckwild!)


App Developer

High Performance DSLs
(OptiML, ...)


High Level Compiler

Accelerators
(GPU, FPGA, SDH)