

Designing Computer Systems for Software 2.0

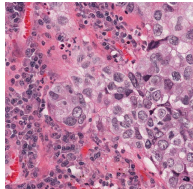
Kunle Olukotun
Stanford University
SambaNova Systems

ISCA '18 Keynote, June 5, 2018

The Era of Machine Learning

- Incredible advances in image recognition, natural language processing, planning and knowledge bases
- Society-scale impact: autonomous vehicles, personalized recommendations and personalized medicine
- Many applications of ML just with supervised learning

Pathology Images

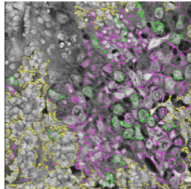


Pathology Reports

A.
Histologic
type:

Adenocarcinoma
Histologic
grade:
Moderately
differentiated
... ..

Visual Features



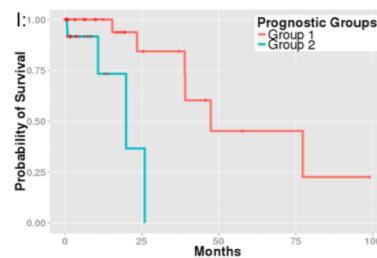
Text Extractions

A.
Histologic
type:

Adenocarcinoma
Histologic
grade:
Moderately
differentiated
... ..

Probability of
Survival

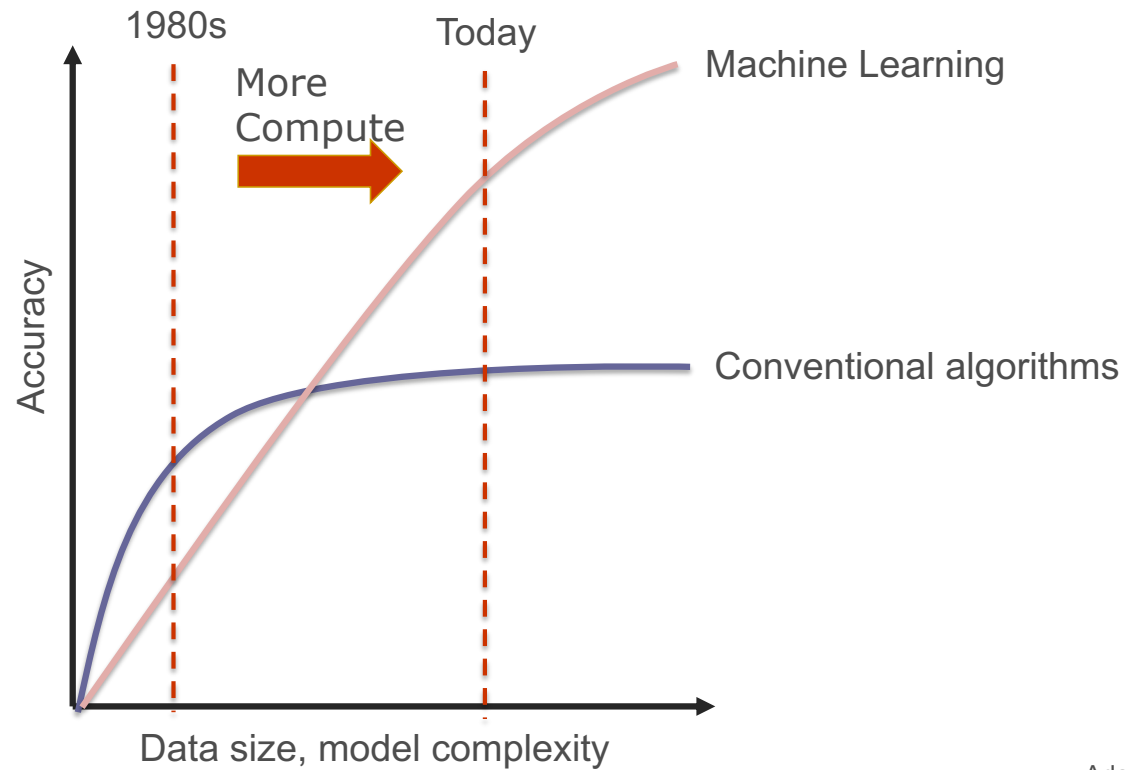
Cleanly separated



Images + patient data
outperform expert
pathologists at prognosis

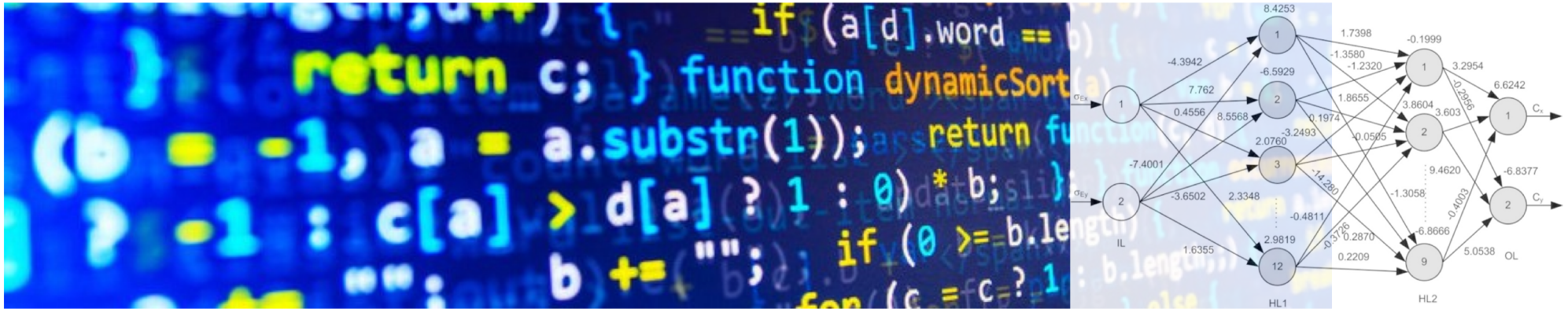
Tumor grade & stage classification from histopathology slides (Nature Comm., Hsing-Yu et. al.)

Machine Learning Today



Adapted from Jeff Dean
HotChips 2017

Software 1.0 vs Software 2.0



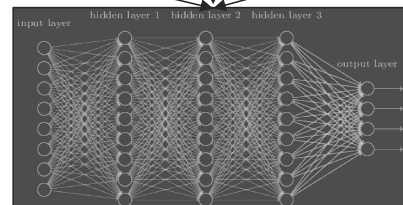
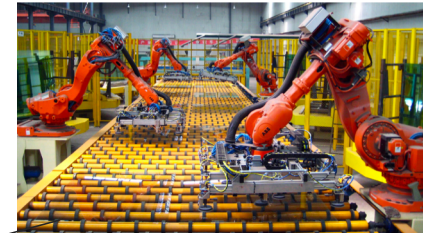
- Written in code (C++, ...)
- Requires domain expertise
 1. Decompose the problem
 2. Design algorithms
 3. Compose into a system
- Written in the weights of a neural network model by optimization

Andrej Karpathy
Scaled ML 2018 talk

Software 2.0 is Eating Software 1.0

Easier to build and deploy

- Build products faster
- Predictable runtimes and memory use: easier qualification



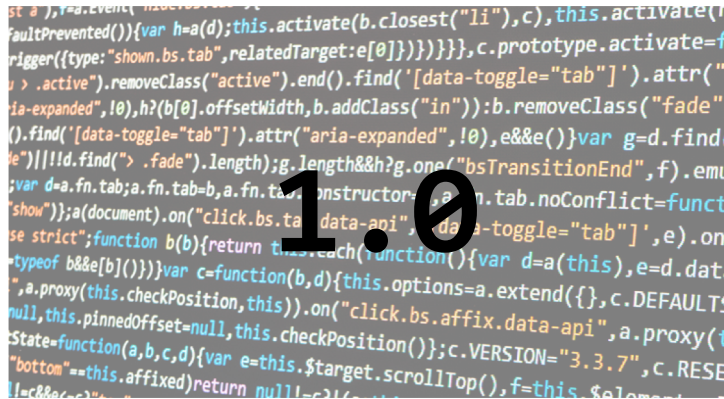
1000x Productivity: Google shrinks language translation code from 500k LoC to 500

Classical problems

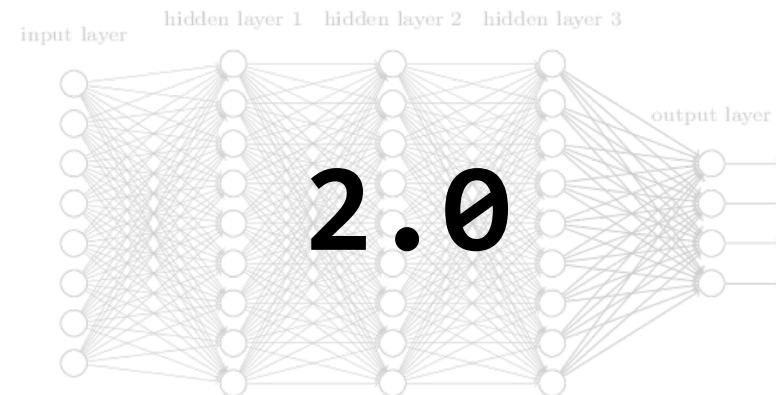
- Data cleaning (Holoclean.io)
- Self-driving DBMS (Peloton)
- Self-driving networks (Pensieve)

<https://jack-clark.net/2017/10/09/import-ai-63-google-shrinks-language-translation-code-from-500000-to-500-lines-with-ai-only-25-of-surveyed-people-believe-automation-better-jobs>

Training Data: The New Input to Software 2.0



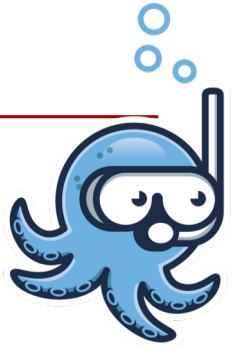
- Input: Algorithms in code
- Compiled to: Machine instructions



- Input: Training data
- Compiled to: Learned parameters

<https://medium.com/@karpathy/>

Better Training Data with Snorkel



- Training data is the critical interface to program Software 2.0

- Expensive & slow especially when domain expertise is needed

- Snorkel

- Get users to provide higher-level (but noisier) training data
 - weak supervision
 - Data programming
- Then model & de-noise it to train high-quality models

- Implications of Snorkel

- Two model training steps
- New training pipeline: data operations interleaved with training



Alex Ratner



Chris Ré

ML Training is Limited by Computation

From EE Times – September 27, 2016



“Today the job of training machine learning models is limited by compute, if we had faster processors we’d run bigger models...in practice we train on a reasonable subset of data that can finish in a matter of months. We could use improvements of several orders of magnitude – 100x or greater.”

Greg Diamos, Senior Researcher, SVAIL, Baidu

Power and Performance

$$\text{Power} = \frac{\text{Performance}}{\text{Energy efficiency}}$$

Power = $\frac{\text{Ops}}{\text{second}} \times \frac{\text{Joules}}{\text{Op}}$

FIXED  

Specialization \Rightarrow better energy efficiency

Key Questions

- How do we speed up machine learning by 100x?
 - Moore's law slow down and power wall
 - >100x improvement in performance/watt
 - Enable new ML applications and capabilities
 - Make ML easier to use (e.g. neural architecture search, Snorkel)

- How do we balance performance and programmability?
 - ASIC-like performance/Watt
 - Processor-like flexibility

- Need a “full-stack” solution
 1. ML Algorithms
 2. Domain Specific Languages and Compilers
 3. Hardware

ML Algorithms

Computational Models

- **Software 1.0 model**
 - Deterministic computations with algorithms
 - Computation must be correct for debugging
- **Software 2.0 model**
 - Probabilistic machine-learned models trained from data
 - Computation only has to be statistically correct
- **Creates many opportunities for improved performance**

Machine Learning Training

Optimization Problem:

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

Diagram annotations:

- Billions (points to N)
- Loss function (points to f)
- Data (points to y_i)
- Model (points to x)

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?

SGD: Two Kinds of Efficiency

- **Statistical efficiency:** how many iterations do we need to get the desired accuracy level?
 - Depends on **the problem** and implementation
- **Hardware efficiency:** how long it takes to run each iteration?
 - Depends on **the hardware** and implementation

trade off hardware and statistical efficiency
to maximize performance

Training Optimization Opportunities

- **Consistency** of algorithms can be relaxed to reduce overheads
- **Sparsity** to reduce communication and computation cost
- **Low precision** arithmetic to reduce computation cost

SGD On Shared Memory

SGD consists of **BILLIONS** of tiny threads that update a single data structure (x)!

Implemented with locking SGD actually gets *slower* with more cores

So what can we do?

Asynchronous Update Strategy (Hogwild!)

- 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 data races?
 - Races introduce errors we can model as noise
 - Below existing noise floor → negligible effect on statistical efficiency
 - Theorem (roughly, Niu et. al. NIPS11): If we do no locking, SGD converges to correct answer—at essentially the same rate!

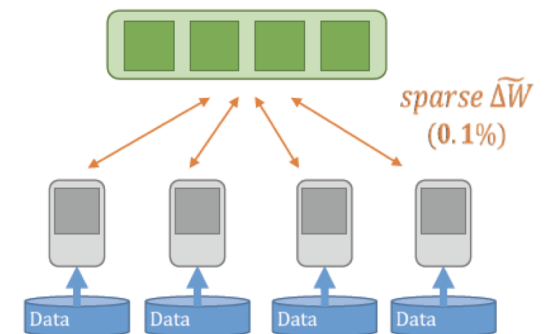
SGD Communication Reduction

■ Shared memory

- Obstinate cache: probabilistically drop 99% of invalidates
- No impact on statistical efficiency
- De Sa, Feldman, Ré, Olukotun: *ISCA 2017*

■ Distributed Memory

- Sparsity: 99.9% of the gradient exchange in distributed SGD is redundant
- Use momentum correction to maintain accuracy
- Lin, Han, Mao, Wang, Dally: *ICLR 18*



Low Precision: The Pros



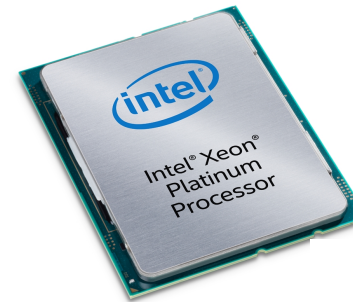
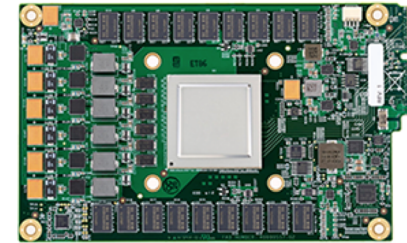
Energy



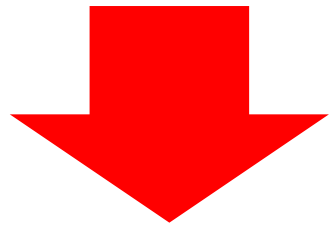
Memory



Throughput



Low Precision: The Con



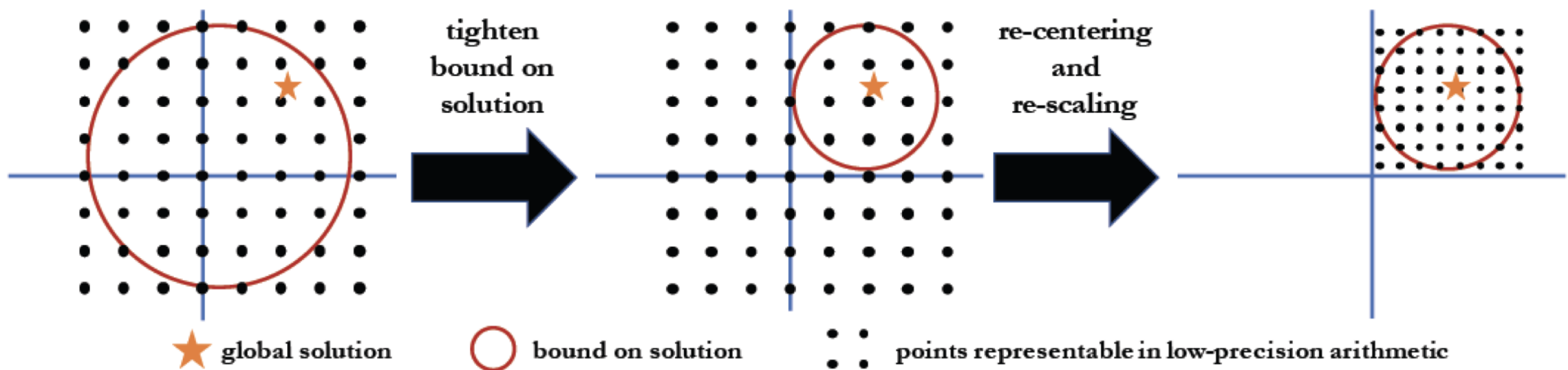
Accuracy

Low precision works for inference (e.g. TPU, Brainwave)

Training usually requires at least 16 bit floating point numbers

High Accuracy Low Precision (HALP) SGD

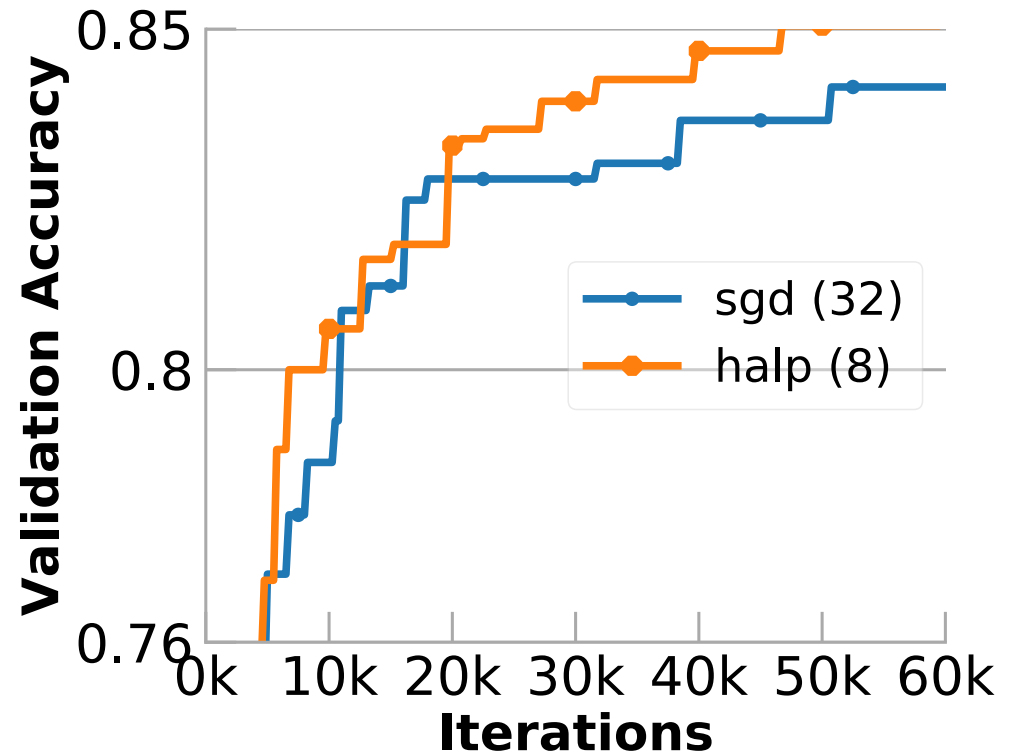
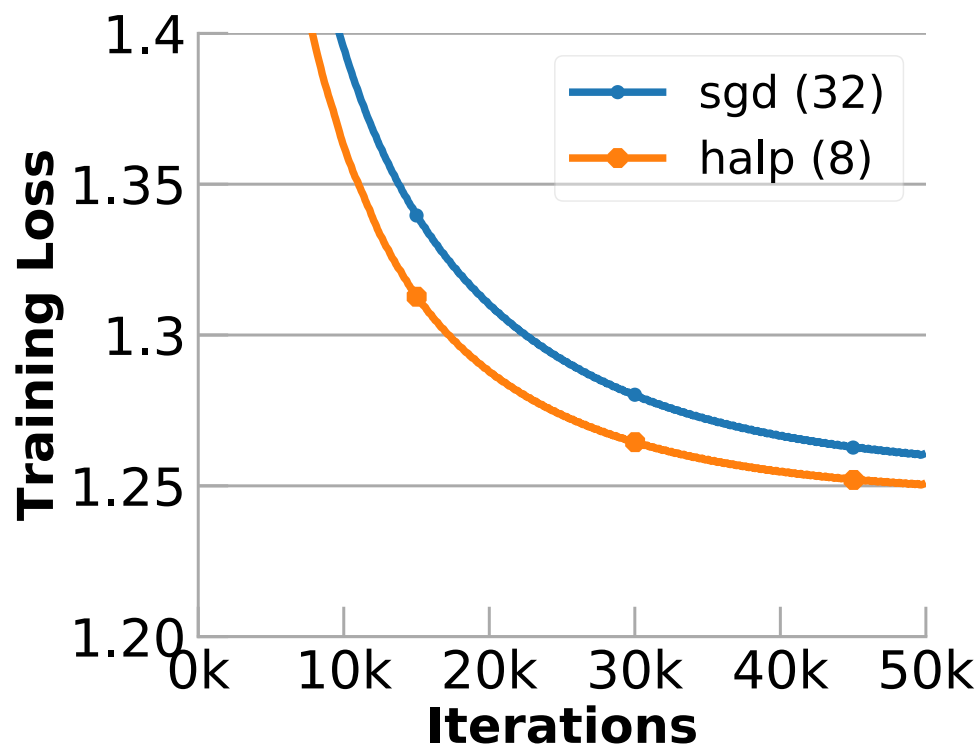
Bit Centering: bound, re-center, re-scale



- The gradients get smaller as we approach the optimum
- Dynamically rescale the fixed-point representation
- Get **less error** with the **same number of bits**

CNN: HALP versus Full-Precision Algorithms

14-layer ResNet on CIFAR10



■ HALP has better statistical efficiency than SGD!

Relax, It's Only Machine Learning

- Relax synchronization: data races are better
 - HogWild! [De Sa, Olukotun, Ré: *ICML 2016*, ICML Best Paper]
- Relax cache coherence: incoherence is better
 - [De Sa, Feldman, Ré, Olukotun: *ISCA 2017*]
- Relax communication: sparse communication is better
 - [Lin, Han et. al.: *ICLR 18*]
- Relax precision: small integers are better
 - HALP [De Sa, Aberger, et. al.]

Better hardware efficiency
with negligible impact on statistical efficiency



Chris De Sa



Song Han



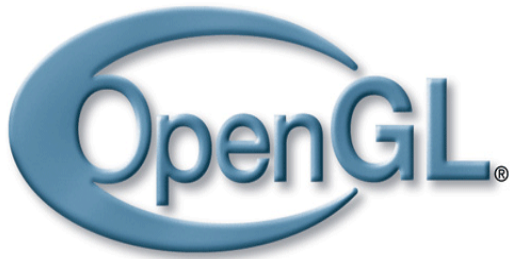
Chris Aberger

Domain Specific Languages

Domain Specific Languages

■ Domain Specific Languages (DSLs)

- Programming language with restricted expressiveness for a particular domain (operators and data types)
- High-level, usually declarative, and deterministic
- Focused on productivity not usually performance
- High-performance DSLs (e.g. OptiML) → performance and productivity



K-means Clustering in OptiML

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

assign each sample to the closest mean

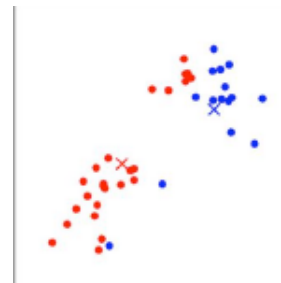
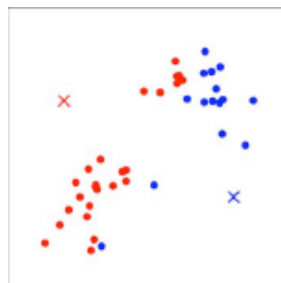
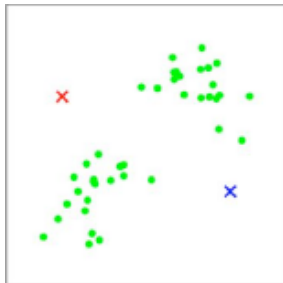


Arvind Sujeeth

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)
- Efficient multicore, cluster and GPU execution



A. Sujeeth et. al.,
“OptiML: An Implicitly
Parallel Domain-
Specific Language for
Machine Learning,”
ICML, 2011.

K-means Clustering in TensorFlow

```
points = tf.constant(np.random.uniform(0, 10, (points_n, 2)))
centroids = tf.Variable(tf.slice(tf.random_shuffle(points), [0, 0], [clusters_n, -1]))
```

```
points_expanded = tf.expand_dims(points, 0)
centroids_expanded = tf.expand_dims(centroids, 1)
```

calculate distances to
current means

```
distances = tf.reduce_sum(tf.square(tf.sub(points_expanded, centroids_expanded)), 2)
assignments = tf.argmin(distances, 0)
```

assign each sample to the closest mean

```
means = []
for c in xrange(clusters_n):
    means.append(tf.reduce_mean(
        tf.gather(points,
            tf.reshape(
                tf.where(
                    tf.equal(assignments, c)
                ), [1, -1])
            ), reduction_indices=[1]))
```

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

```
new_centroids = tf.concat(0, means)
```

```
update_centroids = tf.assign(centroids, new_centroids)
```

Open, standard software for
general machine learning

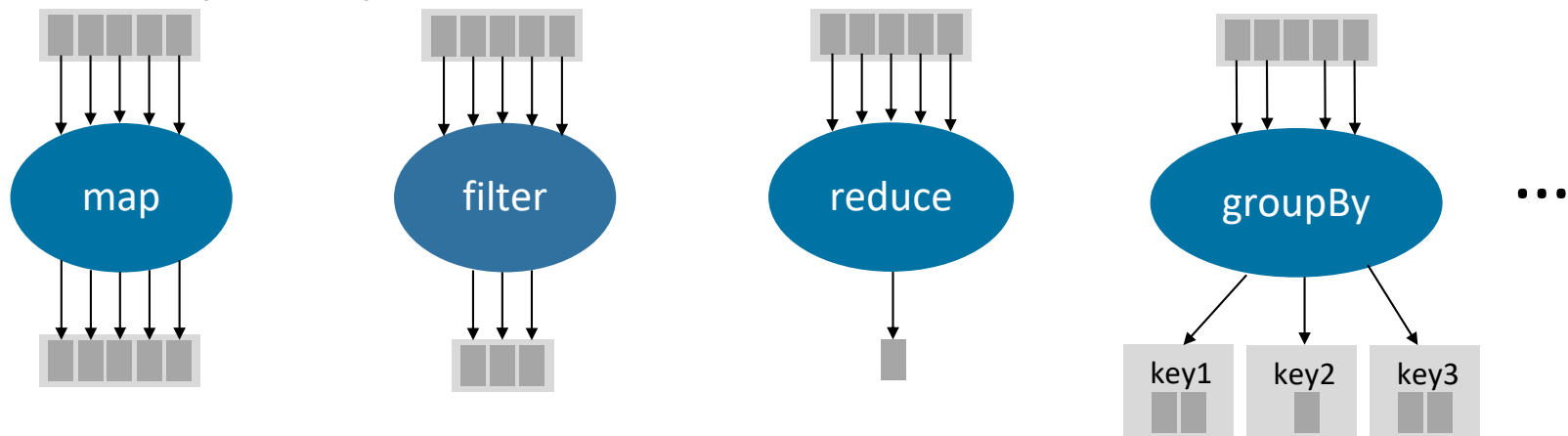
Created: Deep Learning in
particular

First released Nov 2015

DSL IR: Parallel Patterns

Most data analytic computations including ML can be expressed as functional data parallel patterns on collections (e.g. sets, arrays, tables, n-d matrices)

Nested parallel patterns



Map, Zip, Filter, FlatMap, Reduce,
GroupBy, Join, Sort, ...

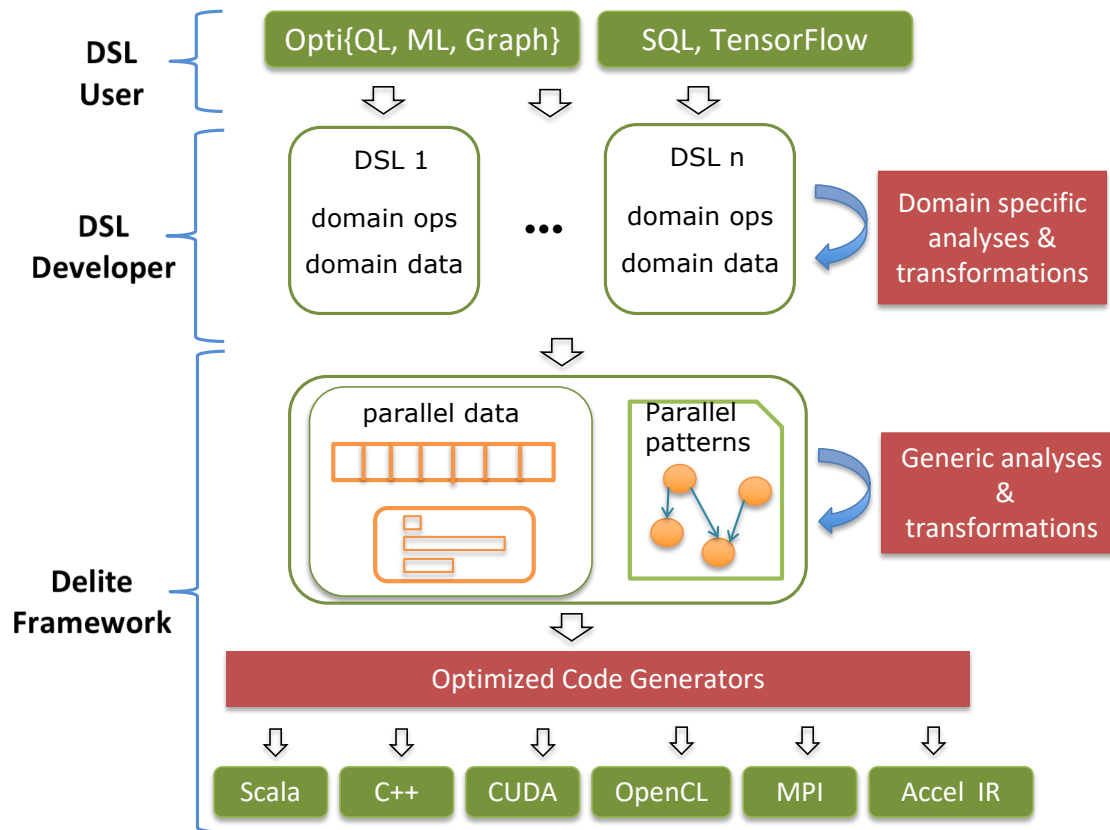
Parallel Pattern Language → High Level Parallel ISA

- A data-parallel language that supports nested parallel patterns $\{\{\{\}\}\}$
- Example application: *k*-means

```
val clusters = samples GroupBy { sample =>
  val dists = kMeans Map { mean =>
    mean.Zip(sample){ (a,b) => sq(a - b) } Reduce { (a,b) => a + b }
  }
  Range(0, dists.length) Reduce { (i,j) =>
    if (dists(i) < dists(j)) i else j
  }
}
val newKmeans = clusters Map { e =>
  val sum = e Reduce { (v1,v2) => v1.Zip(v2){ (a,b) => a + b } }
  val count = e Map { v => 1 } Reduce { (a,b) => a + b }

  sum Map { a => a / count }
}
```

Delite



Framework for building HP DSL compilers

Key elements

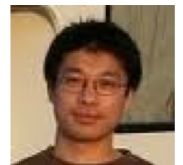
- IR embedded in Scala
- Domain specific optimization
- General parallelism and locality optimizations
 - Structured computation
 - Structured data
- Optimized mapping to HW targets



Hassan Chafi



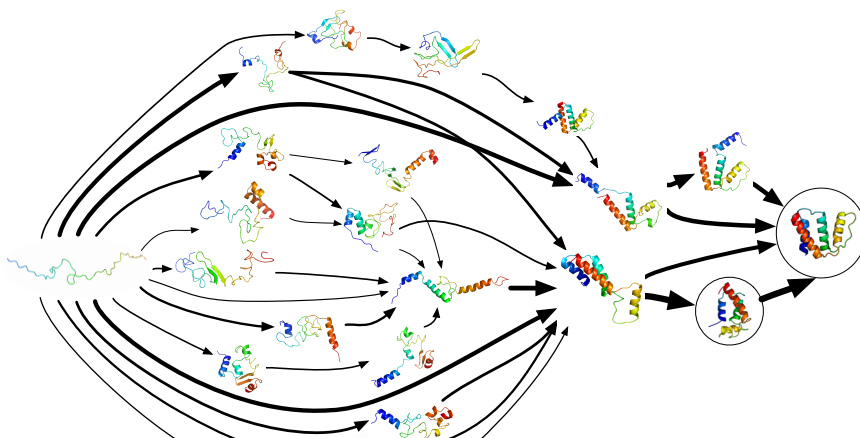
Kevin Brown



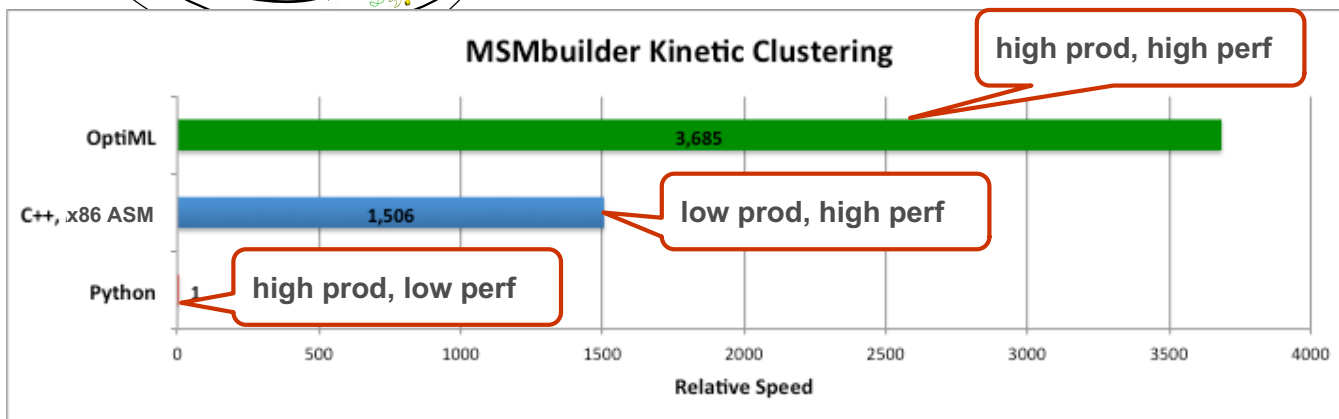
HyoukJoong Lee

MSM Builder Using OptiML

with Vijay Pande



Markov State Models (MSMs)
MSMs are a powerful means of modeling the structure and dynamics of molecular systems, like proteins



Hardware

Accelerators for ML



CPU

- Threads
- SIMD



GPU

- Massive threads
- SIMD
- HBM



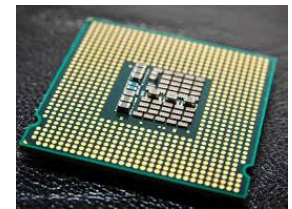
FPGA

- LUTs
- DSP
- BRAM



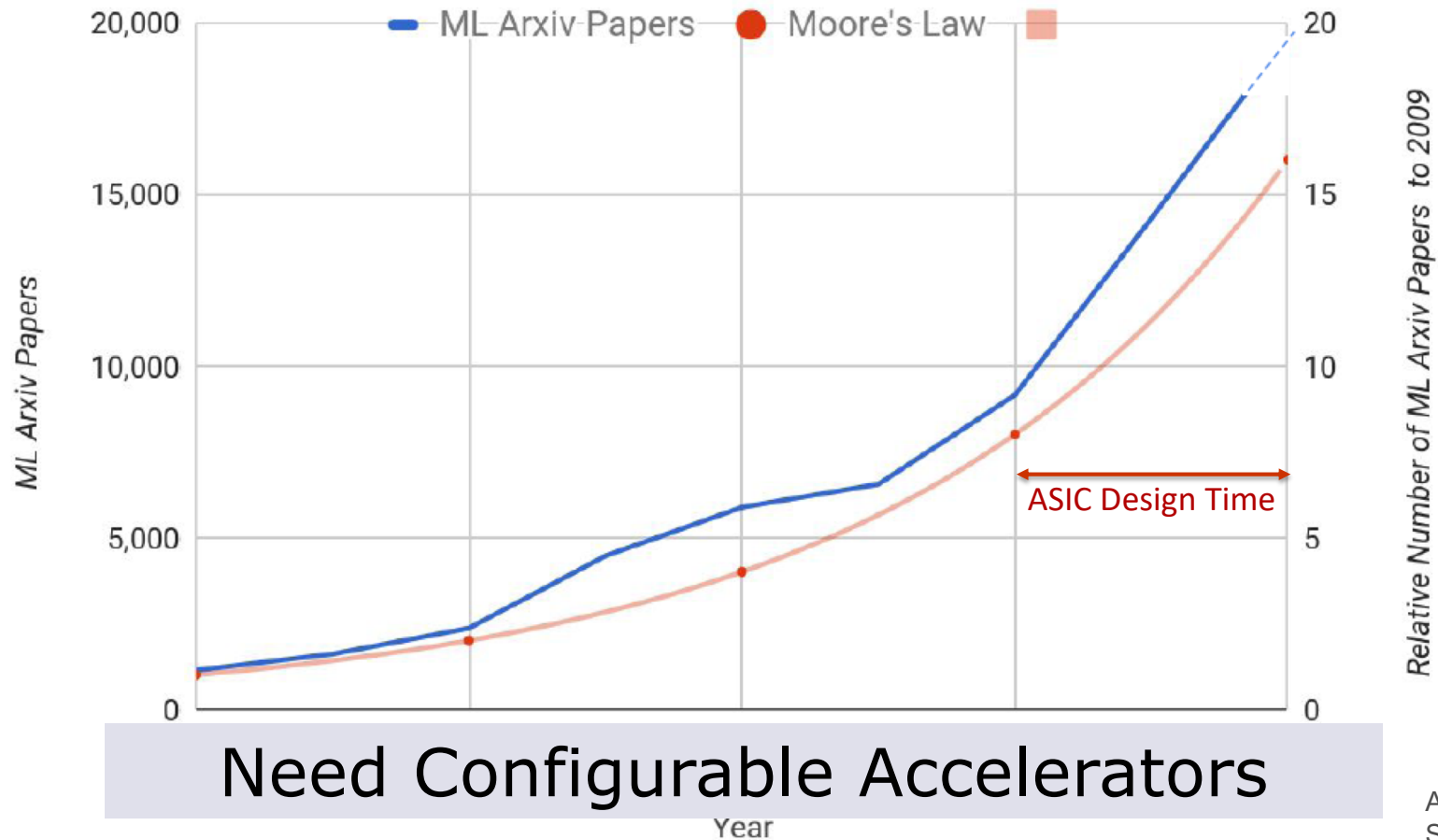
TPU

- MM unit
- BRAM



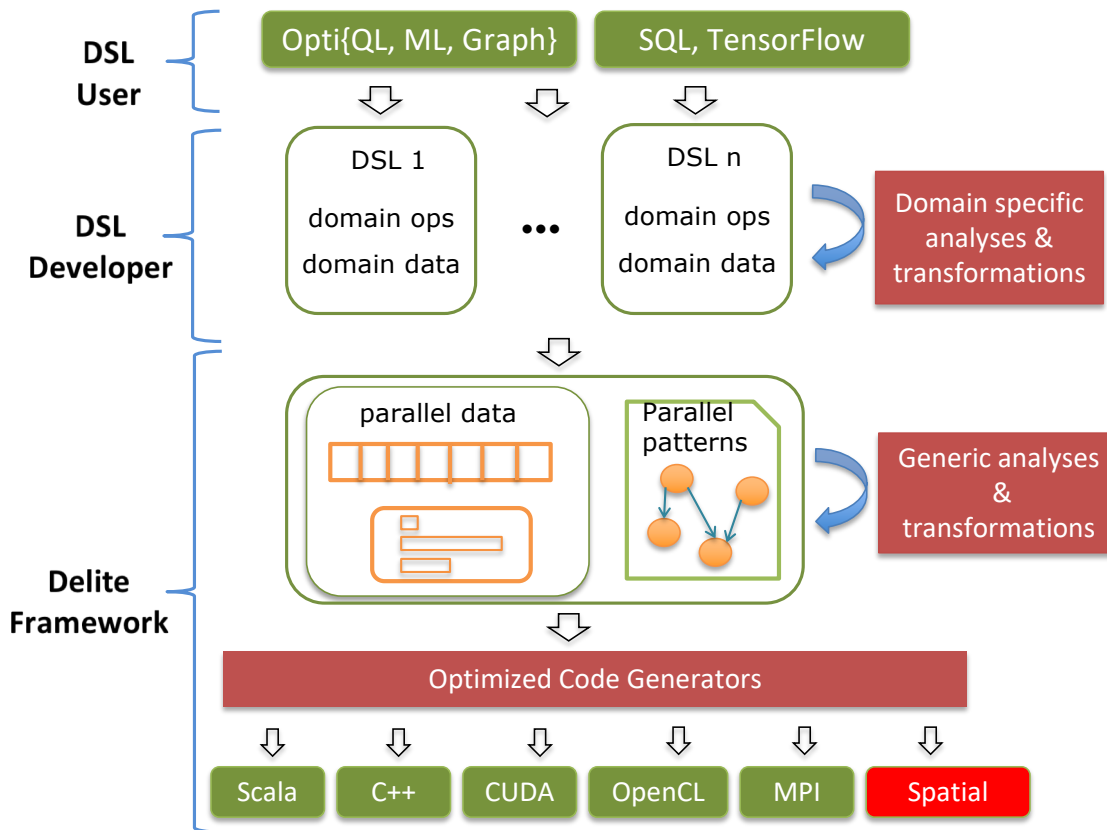
What next?

What to Accelerate? ML Arxiv Papers Per Year



Adapted from Jeff Dean
Scaled ML 2018

Parallel Patterns to Spatial



Accelerator IR: Spatial

- Interface to configurable accelerator

Generate Accelerator IR

- Tile parallel patterns
- Transform nested parallel patterns to hierarchical pipelines

Spatial: Accelerator IR/Language

■ Simplify configurable accelerator design

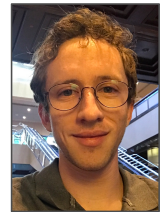
- IR that can be mapped to many hardware targets: FPGA, ASIC, ...
- Constructs to express:
 - Parallel patterns as parallel and pipelined datapaths
 - Hierarchical control
 - Explicit memory hierarchies
 - Explicit parameters
- Optimizes parameters for each target: parallelization, pipelining, memory size, memory banking



David Koeplinger

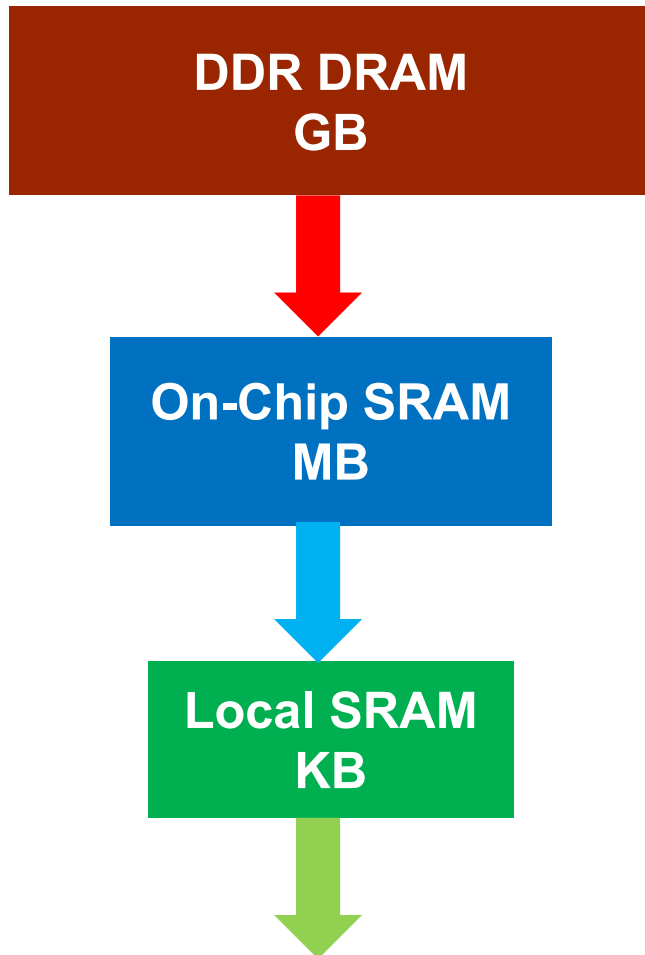
■ Allows programmers and high-level compilers to focus on specifying parallelism and locality

- Designed for performance oriented programmers
- Focus on dataflow instead of threads



Matt Feldman

Programing Locality: Memory Templates



```
val image = DRAM[UInt8](H,W)
```

```
buffer load  image(i, j::j+C) // dense  
buffer gather image(a)         // sparse
```

```
val buffer = SRAM[UInt8](C)
```

```
val accum = Reg[Double]  
val fifo  = FIFO[Float](D)  
val lbuf  = LineBuffer[Int](R,C)  
val pixels = RegFile[UInt8](R,C)
```

GDA in Spatial

```
1 type V = FixPt[TRUE, _9, _7]
2 val x_dram = DRAM[V](R, C)
3 val y_dram = DRAM[Bit](R)
4 val mu0    = SRAM[V](C)
5 val mu1    = SRAM[V](C)
6 val sigma  = SRAM[V](C, C)
7
8 MemReduce(sigma)(R by T){r =>
9   val x = SRAM[V](T, C)
10  val y = SRAM[Bit](T)
11  x load x_dram(r::r+T, 0::C)
12  y load y_dram(r::r+T)
13
14  MemReduce(SRAM[V](C, C))(T by 1){rr =>
15    val sub = SRAM[V](C)
16    val sigma_blk = SRAM[V](C, C)
17    Foreach(C by 1){c =>
18      sub(c) = x(c) - mux(y(c), mu1(c), mu0(c))
19    }
20    Foreach(C by 1, C by 1){(i, j) =>
21      sigma_blk(i, j) = sub(i) * sub(j)
22    }
23    sigma_blk
24  }{(a, b) => a + b }
25 }{(a, b) => a + b }
```

← Arbitrary precision custom types

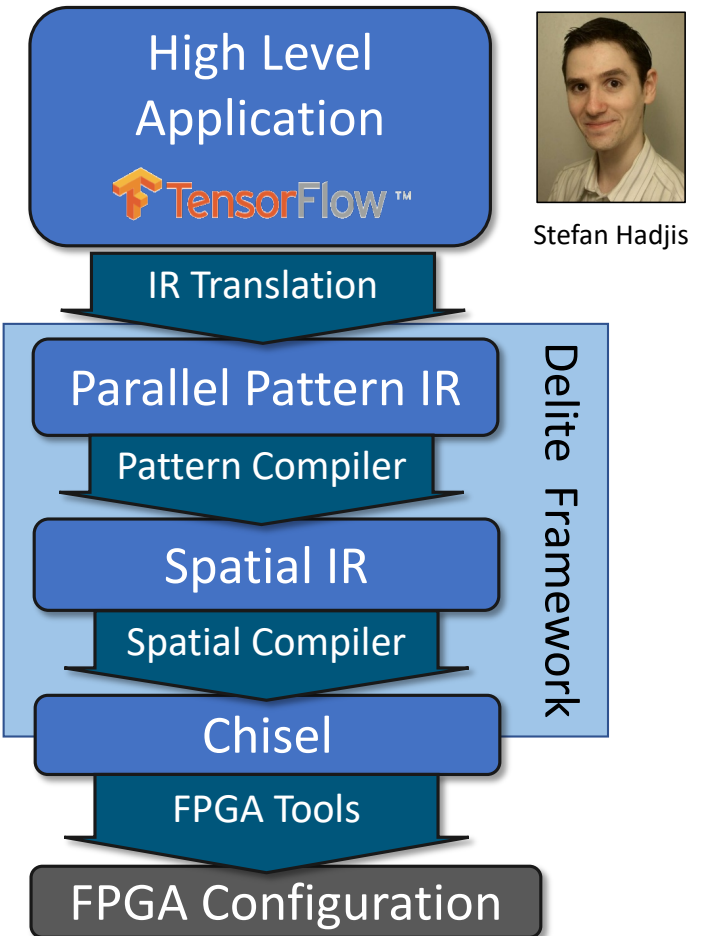
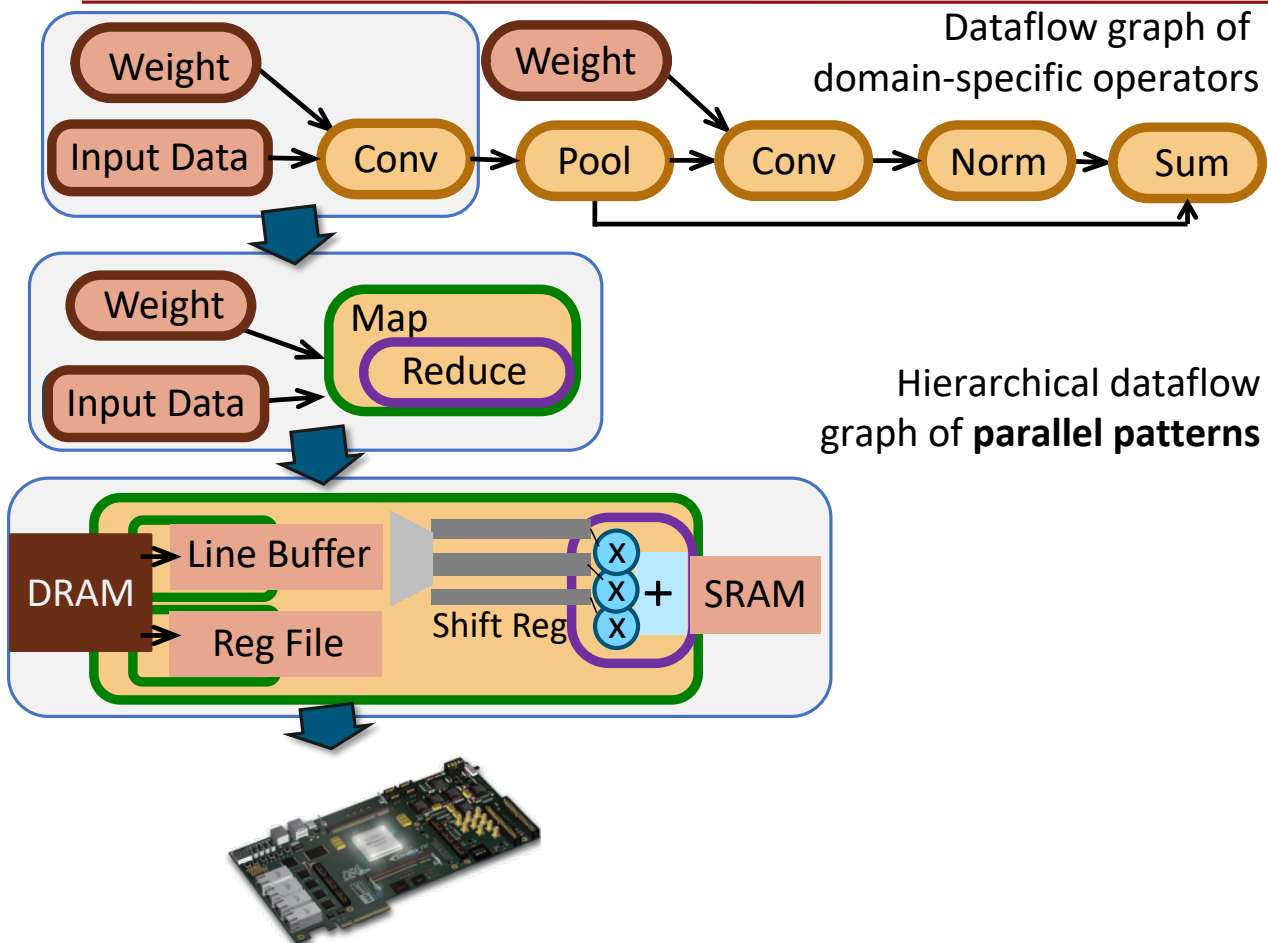
← Off-chip memory allocations

← On-chip memory allocations

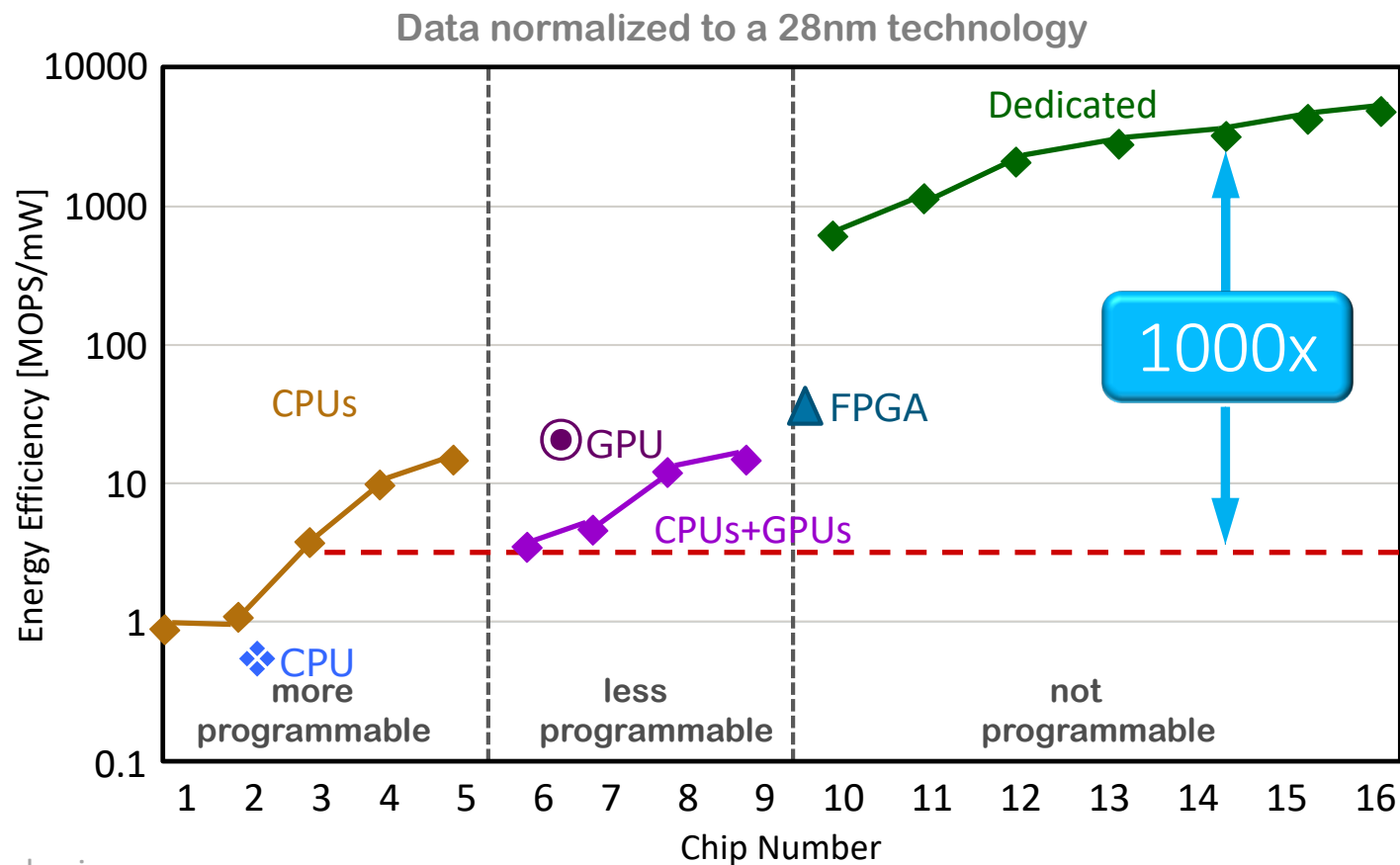
← Explicit memory transfers

← Nested pipelines

TensorFlow to FPGA



Programmability vs. Energy Efficiency

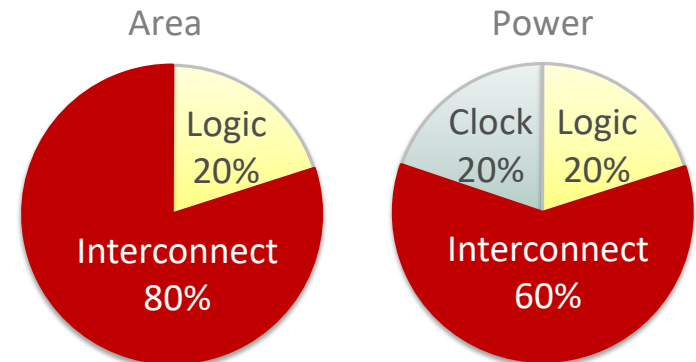


Source: Dejan Markovic

FPGA: Good, Bad and Ugly

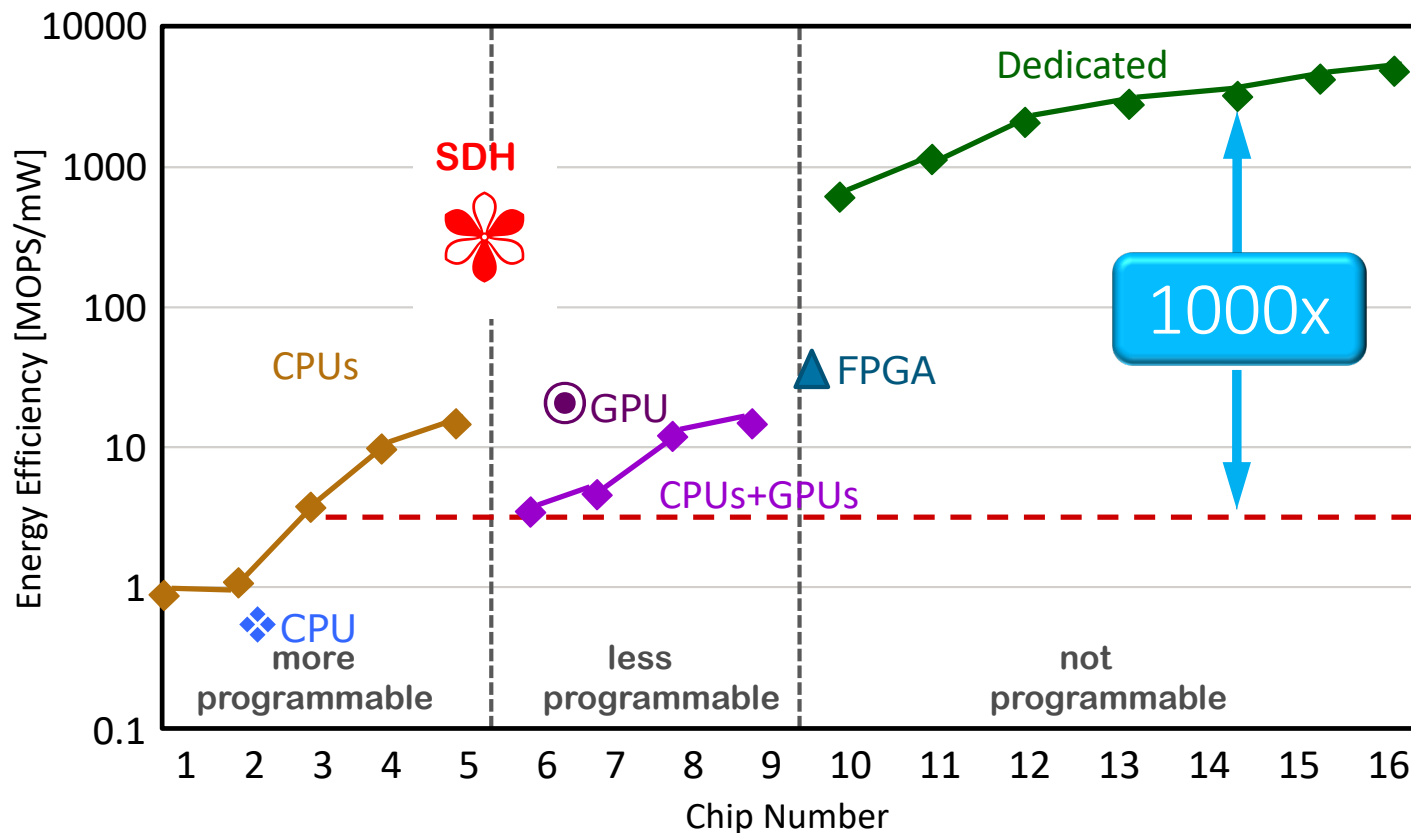
Bit-level reconfigurable logic elements + static interconnect

- Flexibility
 - No instruction overhead
 - Performance / Watt
-
- Fine-grained reconfigurability overheads:
 - >60% area and power spent on interconnect
 - Long compile times (days)



Design reconfigurable hardware with the right abstractions

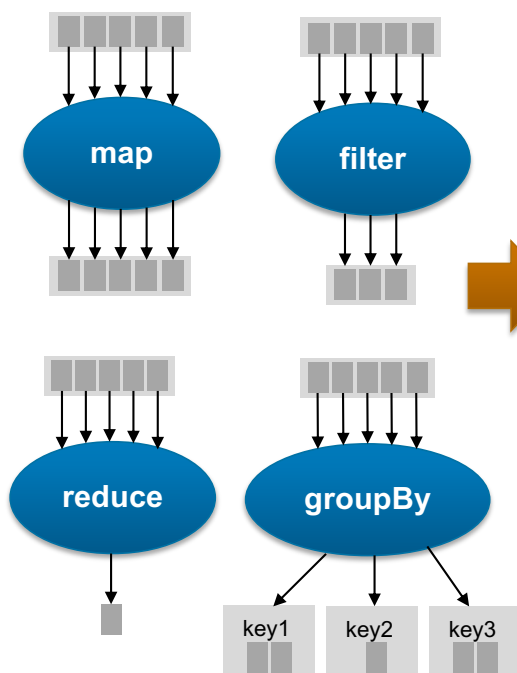
DARPA Software Defined Hardware (SDH) Program



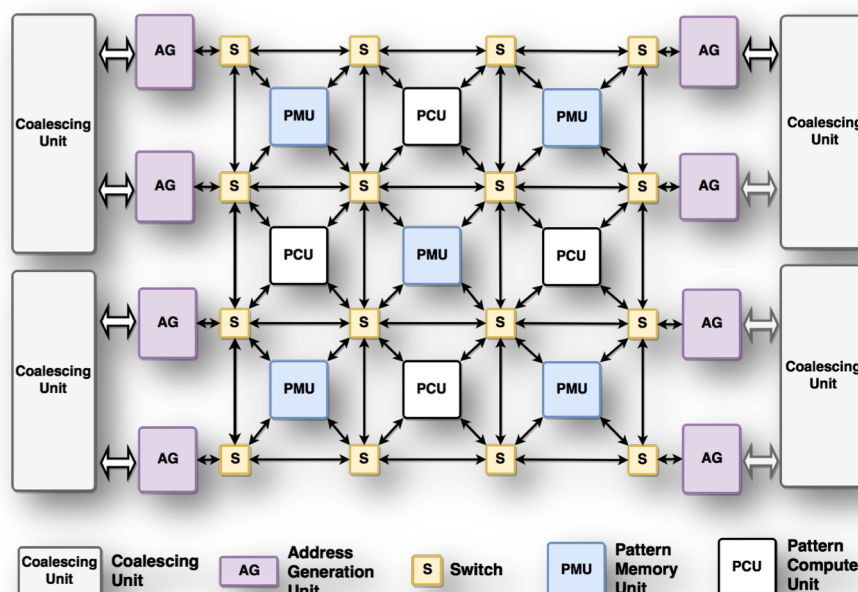
The goal of the SDH program is to build runtime-reconfigurable hardware and software that enables near ASIC performance without sacrificing programmability for data-intensive algorithms.

Plasticine: A Reconfigurable Architecture for Parallel Patterns

High-level Parallel Patterns (Spatial)



Plasticine Accelerator



Tiled architecture with reconfigurable SIMD pipelines, distributed scratchpads, and statically programmed switches

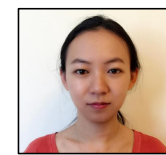
High Performance
Energy Efficiency

Up to **95x** Performance

Up to **77x** Perf/W
vs. Stratix V FPGA



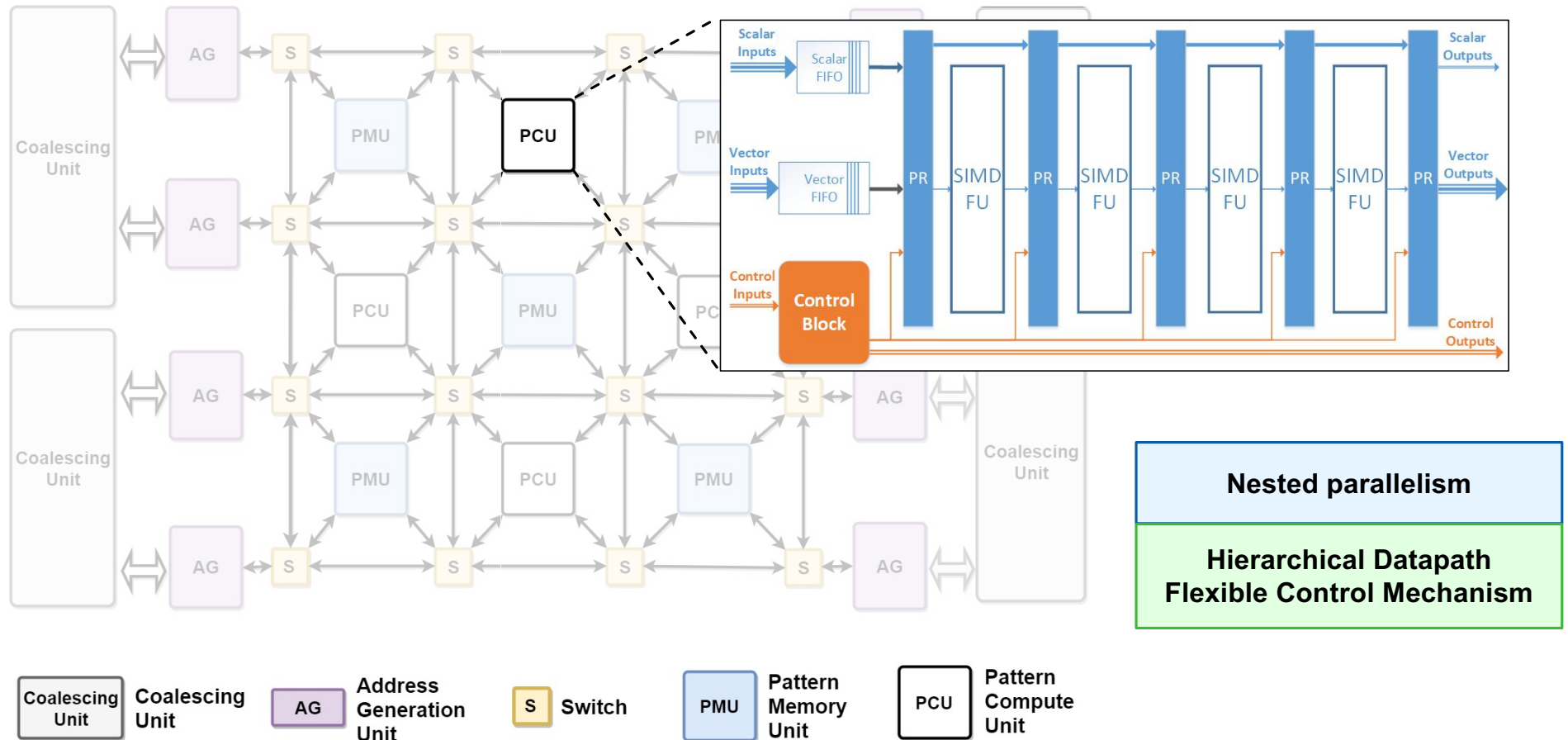
Raghu Prabhakar



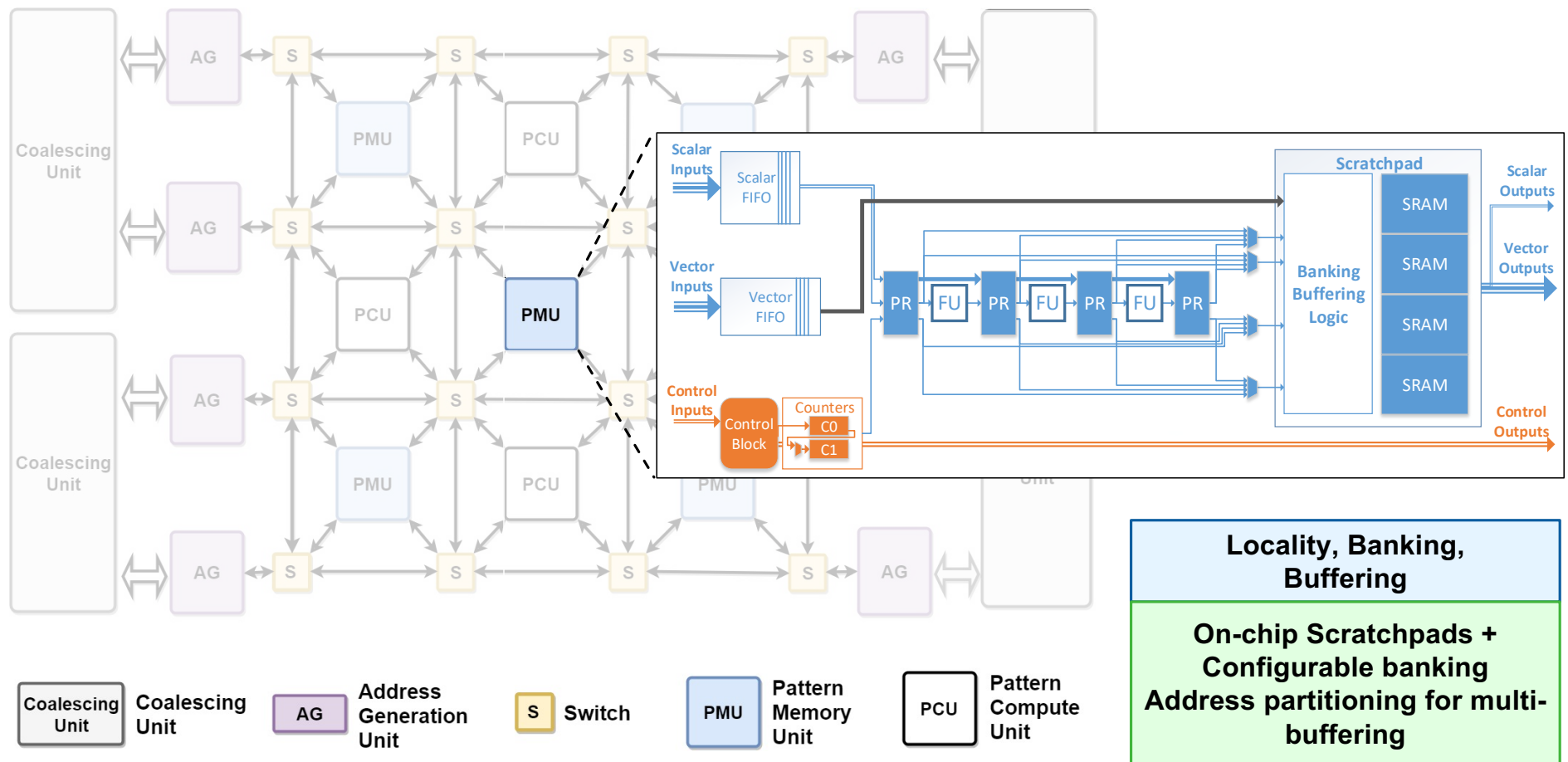
Yaqi Zhang

Prabhakar, Zhang, et. al. ISCA 2017

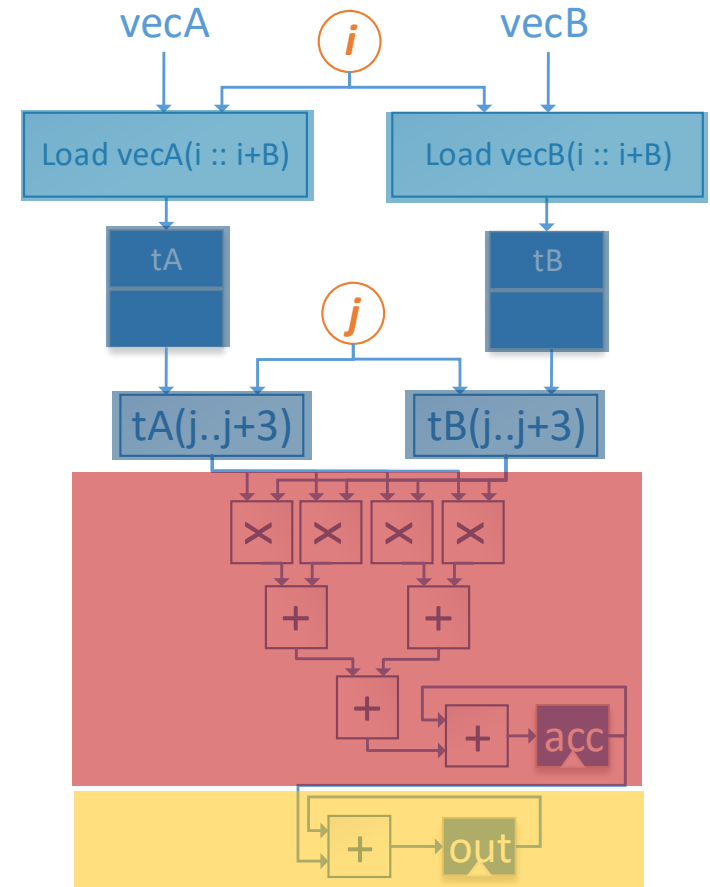
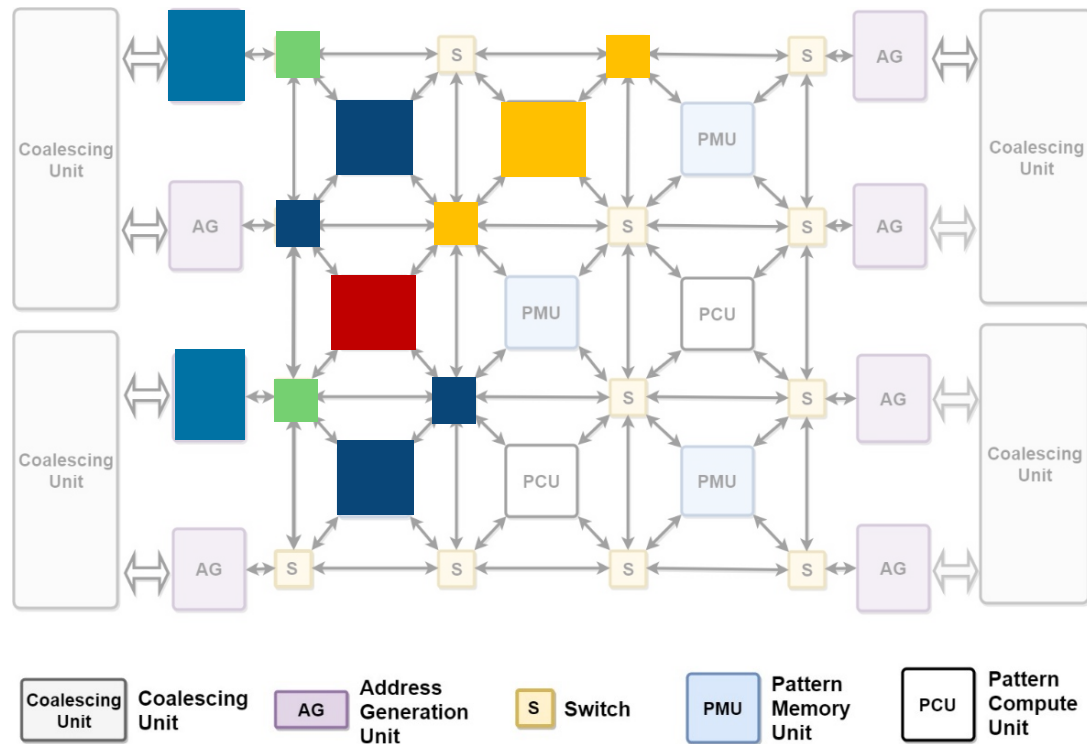
Plasticine: PCU



Plasticine: PMU

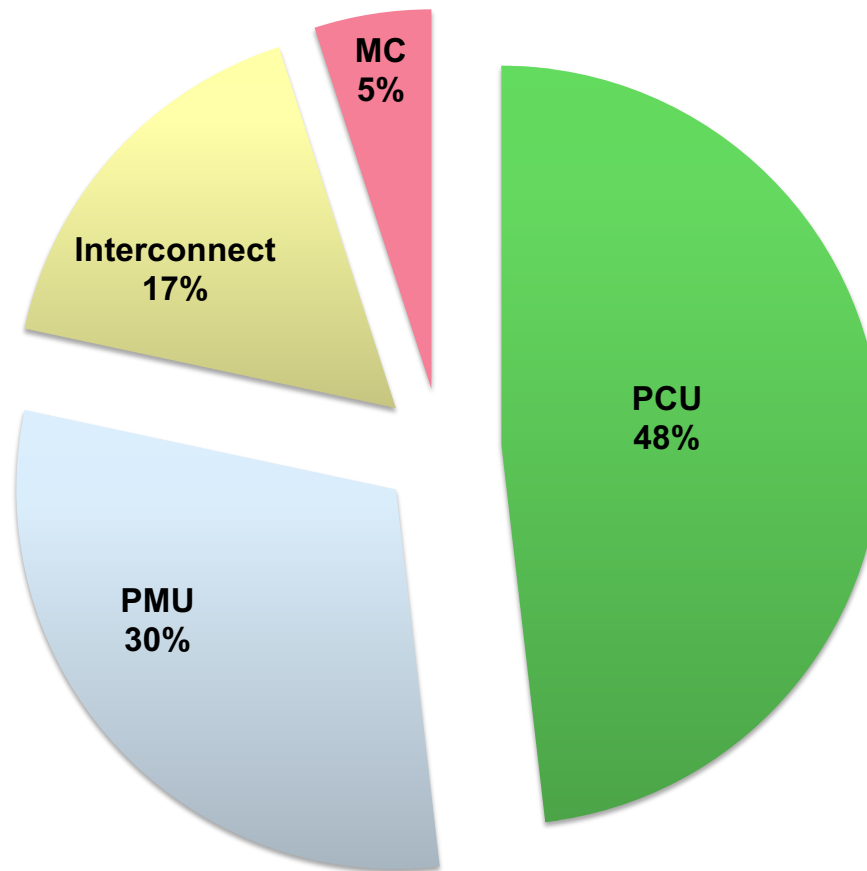


Mapping Spatial to Plasticine



Dot Product

Plasticine Area Breakdown



We Can Have It All with Software 2.0!

- Productivity
- Power
- Performance
- Programmability
- Portability

ML Algorithms (e.g. Hogwild!, HALP)



High Performance DSLs (e.g. OptiML, TensorFlow, PyTorch)



Accelerator IR (e.g. Spatial)



Architectures (e.g. TPU, SDH)

Thank You!

- Questions?