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

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)

Training Data: The New Input to Software 2.0

1.0

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

2.0

• 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
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

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

FIXED

Specialization ⇒ 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:

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

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

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

Chris De Sa | Chris Aberger | Megan Leszczynski | Jian Zhang | Alana Marzoev | Kunle Olukotun | Chris Ré
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
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

- OpenGL
- MATLAB
- SQL
K-means Clustering in OptiML

\[
\text{until converged}(\text{kMeans, tol}) \{ \\
\quad \text{kMeans} => \\
\quad \quad \text{val clusters} = \text{samples}.\text{groupRowsBy} \{ \text{sample} => \\
\quad \quad \quad \text{kMeans}.\text{mapRows}(\text{mean} => \text{dist}(\text{sample, mean})).\text{minIndex} \\
\quad \}
\quad \text{val newKmeans} = \text{clusters}.\text{map}(\text{e} => \text{e}.\text{sum} / \text{e}.\text{length}) \\
\quad \text{newKmeans}
\}
\]

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

K-means Clustering in TensorFlow

```python
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)

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

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]))

new_centroids = tf.concat(0, means)

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

Open, standard software for general machine learning

Deep Learning in particular

First released Nov 2015
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**
- **filter**
- **reduce**
- **groupBy**

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

```scala
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

Opti{QL, ML, Graph} SQL, TensorFlow

DSL 1 domain ops domain data

DSL n domain ops domain data

Domain specific analyses & transformations

Parallel data Parallel patterns

Generic analyses & transformations

Optimized Code Generators

Scala C++ CUDA OpenCL MPI Accel IR

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

MSMbuilder Kinetic Clustering

- OptiML
- C++, x86 ASM
- Python

Relative Speed

- high prod, high perf
- low prod, high perf
- high prod, low perf
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

Need Configurable Accelerators

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

- Allows programmers and high-level compilers to focus on specifying parallelism and locality
  - Designed for performance oriented programmers
  - Focus on dataflow instead of threads

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

```
GDA in Spatial

```

```
| type V = FixPt[TRUE,_9,_7]  |
| val x_dram = DRAM[V](R, C)  |
| val y_dram = DRAM[Bit](R)   |
| val mu0 = SRAM[V](C)        |
| val mu1 = SRAM[V](C)        |
| val sigma = SRAM[V](C,C)    |

MemReduce(sigma)(R by T){r =>
  val x = SRAM[V](T, C)
  val y = SRAM[Bit](T)
  x load x_dram(r::r+T, 0::C)
  y load y_dram(r::r+T)
}

MemReduce(SRAM[V](C,C))(T by 1){rr =>
  val sub = SRAM[V](C)
  val sigma_blk = SRAM[V](C,C)
  Foreach(C by 1){c =>
    sub(c) = x(c) - mux(y(c), mu1(c), mu0(c))
  }
  Foreach(C by 1, C by 1){(i,j) =>
    sigma_blk(i,j) = sub(i) * sub(j)
  }
  sigma_blk
}
```

**Arbitrary precision** custom types

**Off-chip** memory allocations

**On-chip** memory allocations

**Explicit** memory transfers

**Nested** pipelines
TensorFlow to FPGA

Dataflow graph of domain-specific operators

Hierarchical dataflow graph of parallel patterns

High Level Application

TensorFlow™

IR Translation

Parallel Pattern IR

Pattern Compiler

Spatial IR

Spatial Compiler

Chisel

FPGA Tools

FPGA Configuration

Delite Framework

Stefan Hadjis
Programmability vs. Energy Efficiency

Data normalized to a 28nm technology

- CPUs
- GPUs
- CPUs+GPUs
- FPGA
- Dedicated

Source: Dejan Markovic
FPGA: Good, Bad and Ugly

- 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
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)

- map
- filter
- reduce
- groupBy
  - key1
  - key2
  - key3

Plasticine Accelerator

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

Prabhakar, Zhang, et. al. ISCA 2017

High Performance Energy Efficiency

Up to $95x$ Performance
Up to $77x$ Perf/W
vs. Stratix V FPGA
Plasticine: PCU

Nested parallelism
Hierarchical Datapath
Flexible Control Mechanism
Plasticine: PMU

Locality, Banking, Buffering

On-chip Scratchpads + Configurable banking
Address partitioning for multi-buffering
Mapping Spatial to Plasticine

Load vecA($i : i+B$)

Load vecB($i : i+B$)

Dot Product
Plasticine Area Breakdown

- PCU: 48%
- PMU: 30%
- Interconnect: 17%
- MC: 5%
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)

**High-Level Compiler**

**Accelerator IR** (e.g. Spatial)

**Low-Level Compiler**

**Architectures** (e.g. TPU, SDH)
Thank You!

- Questions?