Making Parallelism Easy: My Stanford Odyssey

Kunle Olukotun
Stanford University

CS114, April 18, 2022
My Timeline

1960’s: London
1970’s: Nigeria
1980’s: Michigan
1990’s: CMP TLS
2000’s: Niagara Trans. Mem
2010’s: DSLs
Today: Plasticine + ML SambaNova
The Research Pendulum Swings

Hardware

Computer Architecture

Software
Stanford University
Choosing A Research Direction

- Solve a real problem

- Intellectually challenging ⇒ hardware and software

- Revisit and question conventional wisdom

- Potential to change the way people design and program computer systems

- Industry is not already doing it and many may think it’s a bad idea
Back to the Mid 90’s

- **Microprocessor performance boom**
  - Clock frequency increasing at 40% per year
  - Single processor performance increasing at 50% per year
  - Lots of computer architecture researchers trying to feed Intel new tricks for complex processor design
  - Free lunch for software developers (internet, GUIs, spreadsheets, multimedia)

- **Clouds on the horizon**
  - Architecture trend: single processor performance tricks running out of steam and complexity rising
  - IC Technology trend: Delay of the interconnect not scaling with smaller transistor sizes
Stanford Hydra Chip Multiprocessor (CMP)

- 4 simpler CPUs in same area as complex CPU
- Simpler to design and shorter wires
- Exploit explicit multi-thread parallelism
- Now need to write parallel programs!
  - Shared cache communication on chip makes parallelism easier
Hydra vs. Complex CPU

- ILP only
  - CCPU 30-50% better than single Hydra processor
- ILP & fine thread
  - CCPU and Hydra comparable
- ILP & coarse thread
  - Hydra 1.5–2× better
- “The Case for a CMP” ASPLOS ’96

**Graph:**
- Hydra 4 x simpler CPU
- Complex CPU

**Bar Chart:**
- Speedup across various applications:
  - Sequential
  - Parallel
  - Applications include: compress, eqntott, m88ksim, apsi, MPEG2, applu, swim, tomcatv, OLTP, pmake
Parallel Software Development

- Writing parallel software is difficult
  - Even with shared memory

- Software must be correct
  - Controlling access to shared data ⇒ synchronization (locks)
  - Races ⇒ incorrect program
  - Deadlock ⇒ program hangs

- Software must perform well
  - Find enough parallelism in algorithm
  - Not too much synchronization
  - Not too much communication

- The bottom line
  - Millions of people can write decent sequential programs
  - Few people can write correct parallel programs
  - Tiny minority can write efficient and correct parallel programs
Compiler Limitations and Speculative Threads

- While (True) {
  sentence = read();
  if (!sentence) break;
  err = parse (sentence);  // most of time spent here
  if (err) {
    print (sentence, err);
  }
}

- Could you parallelize this loop?
- Could a compiler parallelize this loop?
  - Compilers have to be conservative ⇒ “always”
- Hardware support for speculation
  - Safety net so compilers can be aggressive in finding parallelism ⇒ “sometimes” instead of “always”
Dynamic Java Parallelization (JRPM)

- A complete system for dynamically parallelizing sequential Java programs
  - JVM, compiler, runtime, architecture

- Easily exploit thread-level parallelism automatically without complex analysis
  - Find parallelism using dynamic profiling
  - Speculative threads execute in parallel safely

- Good performance
  - 3-4x speedup on floating-point
  - 2-3x speedup on multimedia
  - 1.5-2.5x speedup on integer
Recent Speculative Thread Results

Ten hard to parallelize C++ benchmarks from SPEC 2006

T4: Compiling Sequential Code for Effective Speculative Parallelization in Hardware, Victor A. Ying, Mark C. Jeffrey, Daniel Sanchez. ISCA 2020
Changing Industry Practice

- Write papers
- Develop prototypes and give them away
- Give talks (Intel, Sun, SGI, DEC, HP, IBM)

Not enough to change two multi-billion $ industries
  - Single thread performance still improving
  - No desire/ability for software industry to take on parallel programming
  - Industry is naturally conservative

- Crossing the academia/industry boundary
Afara WebSystems

- Founded in 1999
  - Height of internet boom
  - Large web sites running out of power and space
  - Goal: Revolutionize internet data centers (multi-B $ market)
  - Approach: 10x performance/watt with new microprocessor based on CMP

- Systems company
  - Top team: Intel, Sun, Cisco, HP, Brocade, C-Cube, SGI
  - Design silicon and build system
  - Sell as appliance with software
  - Higher margins than selling chips
  - $100M to market: “Big-boy project”
Making Hardware Threads Cheap: Niagara Approach

- Performance/watt and high throughput the design focus
  - Commercial server applications
  - Throughput more important than latency

- Many simple cores vs. few complex cores
  - No branch prediction or fancy pipeline techniques
  - Lower development cost, schedule risk with simple pipeline

- Microprocessor with 32 threads exploits TLP
  - Memory and pipeline stall time hidden by multiple threads
  - Shared cache allows efficient data sharing among threads

- Memory system designed for high throughput with cache misses
  - Banked and highly associative cache
  - High bandwidth interface to DRAM for cache misses
Afara WebSystems

- Founded in 1999
  - Height of internet boom
  - Large web sites running out of power and space
  - Goal: Revolutionize internet data centers (multi-B $ market)
  - Goal: Approach: 10x performance/watt with new microprocessor based on CMP

- Sold to Sun Microsystems in 2002
  - Dot com bomb
  - VCs wanted to cash-out
  - Sun processor design lagging
  - Most of the team moved to Sun to finish the design
Niagara 1 (UltraSPARC T1) Die

- **Features**
  - 8 64-bit 4-way Multithreaded
  - SPARC Cores
  - 16 KB, 4-way 32B line ICache per Core
  - 8 KB, 4-way 16B line write-through DCache per Core
  - Shared 3 MB, 12-way 64B line writeback L2 Cache
  - 4 144-bit DDR-2 channels
  - 3.2 GB/sec JBUS I/O

- **Technology**
  - TI's 90nm CMOS Process
  - 63 Watts @ 1.2GHz/1.2V
  - Die Size: 379mm²
  - 279M Transistors
  - Flip-chip ceramic LGA
e-business Applications

Web Server Tier
- Apache

Application Server Tier
- J2EE
- Java JVM

Database Server Tier
- Oracle 9i

• Web content
• Web2005

• Business logic
• JBB2005

• Persistent store
• TPC-C
Throughput Performance

Performance relative to Pentium D

- Power5+
- Opteron
- Niagara

SPECintRate  SPECFPRate  SPECJBB05  SPECWeb05  TPC-C
Performance/Watt

SPECintRate  SPECFPRate  SPECJBB05  SPECWeb05  TPC-C

Power5+  Opteron  Niagara

Performance/Watt relative to Pentium D

SPECintRate  SPECFPRate  SPECJBB05  SPECWeb05  TPC-C
Dawning of the Era of CMPs (Multicore)

- Industry and other academics not keen on CMPs
  - I got tenure for this work, but not everybody thought it was the right decision. Some thought industry would never pick up CMPs

- Uniprocessor performance scaling reaches limits
  - Power consumption increasing dramatically
  - Wire delays becoming a limiting factor
  - instruction-level parallelism (ILP) in single programs is mined out

**The Right Hand Turn:**
- Move away from frequency as performance
- Multi—everywhere; MT, CMP

Lesson: Innovation requires research courage
- Have to be willing to buck the conventional wisdom
- Good research requires risk taking

From Intel Developer Forum, September 2004
Microprocessor Trends

![Graph showing trends in microprocessor performance and efficiency over time.](image)

- **Moore’s Law** shows exponential growth in the number of transistors on a microprocessor chip.
- **Sequential performance plateau** indicates a slowdown in performance improvement.
- **Single-Thread Performance (SpecINT)** tracks the improvement in performance for single-thread tasks.
- **Frequency (MHz)** increases with technological advancements.
- **Typical Power (Watts)** shows a trend that power consumption has increased as well.
- **Number of Cores** indicates the trend towards multi-core processors.
- **Power Wall** represents the challenge of increasing heat generation with performance.

Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten.
UltraSPARC T2 and T3

T2
8 cores, 64 threads, 1.3GHz
65 nm, 2007

T3
16 cores, 128 threads, 1.6GHz
45 nm, 2009
SPARC @ Oracle

T3
16 x 2nd Gen cores
6MB L2 Cache
1.7 GHz

T4
8 x 3rd Gen Cores
4MB L3 Cache
3.0 GHz

T5
16 x 3rd Gen Cores
8MB L3 Cache
3.6 GHz

M5
6 x 3rd Gen Cores
48MB L3 Cache
3.6 GHz

M6
12 x 3rd Gen Cores
48MB L3 Cache
3.6 GHz

M7
32 x 4th Gen Cores
64MB L3 Cache
4.1 GHz
DAX1
My Timeline

- 1960’s: London
- 1970’s: Nigeria
- 1980’s: Michigan
- 1990’s: CMP (multicore)
- 2000’s: Niagara (Afara)
- 2010’s: Domain Specific Languages
- Today: RDA (SambaNova)
Era of Power Limited Computing

- **Mobile**
  - Battery operated
  - Passively cooled

- **Data center**
  - Energy costs
  - Infrastructure costs
Power and Performance

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

FIXED

Specialization ⇒ better energy efficiency
Heterogeneous Parallel Architectures Today

Only way to get high performance and performance/watt
Programmability Chasm

Applications
- Scientific Engineering
- Artificial Intelligence
- Personal Robotics
- Data informatics

Ideal Parallel Programming Language

Pthreads
OpenMP

Sun T2

CUDA
OpenCL

Nvidia Fermi

Verilog
VHDL

Altera FPGA

MPI
PGAS

Cray Jaguar
The ideal parallel programming language

Performance

Productivity

Generality
Successful Languages

Performance

Productivity

Generality

C/C++

Java

python

Ruby
Domain Specific Languages

- Domain Specific Languages (DSLs)
  - Programming language with restricted expressiveness for a particular domain
  - High-level, usually declarative, and deterministic
  - Focused on productivity

- OpenGL
- MATLAB
- SQL
- RAILS
Way Forward ⇒ Domain Specific Languages

- Domain Specific Languages
- Performance (Heterogeneous Parallelism)
- Productivity
- Generality

Languages: SQL, MATLAB, C/C++, Python, Ruby, Java
Benefits of High Performance DSLs

Productivity
- Shield average programmers from the difficulty of parallel programming
- Focus on developing algorithms and applications and not on low level implementation details

Performance
- Match high level domain abstraction to generic parallel execution patterns
- Restrict expressiveness to easily extract all available parallelism
- Use domain knowledge and semantics for static/dynamic optimizations

Portability and forward scalability
- DSL & Runtime can be evolved to take advantage of latest hardware features
- Applications remain unchanged
- Allows innovative HW without worrying about application portability
Heterogeneous Parallel Programming with DSLs

Applications
- Scientific Engineering
- Artificial Intelligence
- Personal Robotics
- Data informatics

Domain Specific Languages
- Statistics (R)
- Physics (Liszt)
- Data Analytics (OptiQL)
- Graph Alg. (Green Marl)
- Machine Learning (OptiML)

DSL Compiler

Heterogeneous Hardware

New Arch.
Scaling the DSL Approach

- Many potential high-performance DSLs
- Enable smart CS graduates to easily create new DSLs
  - Make optimization knowledge reusable
  - Simplify the compiler generation process
- A few DSL developers enable many more DSL users
Delite: Common DSL Infrastructure

Applications
- Scientific Engineering
- Artificial Intelligence
- Personal Robotics
- Data informatics

Domain Specific Languages
- Statistics (R)
- Physics (Liszt)
- Data Analytics (OptiQL)
- Graph Alg. (Green Marl)
- Machine Learning (OptiML)

Delite DSL Infrastructure
- Parallel data
- Parallel patterns
- Analyses & Transformations

Heterogeneous Hardware
- New Arch.
Delite: Common DSL Infrastructure

Key elements
- DSLs embedded in Scala
- Domain specific optimization
- Parallel patterns: functional data parallel operations on collections (e.g., set, tables, arrays)
- General parallelism and locality optimizations
- Optimized mapping to HW targets
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)

- Looping abstractions with extra information about parallelism and access patterns

Map

\[ y = \text{vector} + 4 \]
\[ y = \text{vector} \times 10 \]
\[ y = \text{sigmoid} (\text{vector}) \]

Zip

\[ y = \text{vecA} + \text{vecB} \]
\[ y = \text{vecA} \div \text{vecB} \]
\[ y = \max (\text{vecA}, \text{vecB}) \]

Reduce

\[ y = \text{vector.sum} \]
\[ y = \text{vector.product} \]
\[ y = \max (\text{vector}) \]

FlatMap

\[ y = \text{SELECT * FROM vector WHERE elem} < 5 \]
\[ \text{vector.groupBy}\{e \Rightarrow e \% 3\} \]
OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning, ICML 2011

- Designed for iterative statistical inference
  - e.g. SVMs, logistic regression, neural networks, etc.
  - Dense/sparse vectors and matrices, message-passing graphs, training/test sets

- Mostly functional
  - Data manipulation with classic functional operators (map, filter)
  - ML-specific ones (sum, vector constructor, untilconverged)
  - Math with MATLAB-like syntax (a*b, chol(..), exp(..))
  - Mutation is explicit (.mutable) and last resort

- Runs anywhere
  - Single source to multicore CPUs, GPUs, and clusters (via Delite)
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.
Today’s DSLs for ML

TensorFlow

PyTorch
My Timeline


London Nigeria Michigan CMP (multicore) Niagara Domain Specific Languages Plasticine + ML SambaNova
Two Big Trends in Computing

- Moore’s Law is slowing down
  - Dennard scaling is dead
  - Computation is limited by power
  - Conventional computer systems (CPU) stagnate

- Success of Machine Learning (ML)
  - Incredible advances in image recognition, natural language processing, and knowledge base creation
  - Society-scale impact: autonomous vehicles, scientific discovery, and personalized medicine
  - Insatiable computing demands for training and inference

Demands a new approach to designing computer systems for ML
ML Applications are Dataflow

1000x Productivity
Google shrinks language translation code from 500k imperative LoC to 500 lines of dataflow (TensorFlow)
ML Dataflow Graphs into Parallel Patterns

Dataflow Graph Analyzer

Hierarchical Parallel Patterns

Input Data

Output Data

Conv
Pool
Conv
Norm
Sum

Map
Reduce
GroupBy
Filter

Weight

Weight

Weight

Input Data

Output Data

PYTORCH

TensorFlow

SQL
Spatial: Software Defined Hardware IR

- IR for hierarchical pipeline dataflow
  - Constructs to express:
    - Parallel patterns as parallel and pipelined datapaths
    - Explicit memory hierarchies
    - Hierarchical control

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

spatial-lang.org

val output = vecA.Zip(vecB){(a,b) => a * b} Reduce{(a,b) => a + b}

val vecA = DRAM[Float](N)
val vecB = DRAM[Float](N)
val out = Reg[Float]

Reduce(N by B)(out) { i =>
  val tA = SRAM[Float](B)
  val tB = SRAM[Float](B)
  val acc = Reg[Float]

  tA load vecA(i :: i+B)
  tB load vecB(i :: i+B)

  Reduce(B by 1)(acc){ j =>
    tA(j) * tB(j)
  }
}{a, b => a + b}

}{a, b => a + b}
Tiled Dot Product

val vecA = DRAM[Float](N)
val vecB = DRAM[Float](N)
val out = Reg[Float]

Reduce(N by B)(out) { i =>
  val tA = SRAM[Float](B)
  val tB = SRAM[Float](B)
  val acc = Reg[Float]

  tA load vecA(i :: i+B)
  tB load vecB(i :: i+B)

  Reduce(B by 1)(acc){ j =>
    tA(j) * tB(j)
  }{a, b => a + b}
}{a, b => a + b}
Reconfigurable Dataflow Architecture (RDA)
Plasticine: A Reconfigurable Dataflow Architecture (RDA)

Parallel Patterns (Spatial)

- **map**
- **filter**
- **reduce**
- **groupBy**

Plasticine Architecture

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** Perf.
- Up to **77x** Perf/W

vs. Stratix V FPGA
Relax, It’s Only Machine Learning

- Stochastic Gradient Descent (SGD)
  - Key algorithm in ML training
  - Time to accuracy = # itters (statiscal eff.) x time per itter (hardware eff.)

- 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]

Better hardware efficiency with negligible impact on model accuracy
SambaNova Systems

2017
Founded the company

Palo Alto
California
USA

ML/AI
Reconfigurable Dataflow Architecture

350+
HW/SW AI Engineers

Kunle Olukotun
Professor EE/CS
Stanford University

Rodrigo Liang

Chris Ré
Professor CS
Stanford University
SambaNova Cardinal SN10

- First Reconfigurable Dataflow Unit (RDU)
- TSMC 7nm
- 40B transistors and 50 Km of wire
- 320 TFLOPS
- 320 MB on chip
- Direct interfaces to TBs off chip
RDU Attributes

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

- Specialized compute and memory
  - PCUs: SIMD pipeline
  - PMUs: large scratchpad banks
  - Efficient prefetching

- Wide interconnect
  - Vectorized datapath
  - Static and dynamic network

- Spatial unrolling to exploit all parallelism
  - Vectors
  - Pipelines
  - Spatial streams (Metapipelining)
Pattern Compute Unit (PCU)
Pattern Memory Unit (PMU)
One Kernel at a Time

Kernel-by-kernel
Bottlenecked by memory bandwidth and host overhead
Dataflow Exploits Locality and Parallelism

Spatial programming
Eliminates memory traffic and overhead

Metapipelining
Exploits more parallelism
A Fundamentally New Software Stack for Dataflow

\[
t_1 = \text{conv}(\text{in}) \\
t_2 = \text{pool}(t_1) \\
t_3 = \text{conv}(t_2) \\
t_4 = \text{norm}(t_3) \\
t_5 = \text{sum}(t_4)
\]

- Computation and memory access are coupled
- Traditional compilers map kernels to accelerator in time
- Communication through the memory hierarchy
- Kernel at at time optimization

- Computation and memory access are decoupled
- Dataflow compilers map kernels to accelerator in time and in space
- Program the communication between kernels
- Global model optimization
Figure 15: Performance breakdown of each component of FAST relative to a TPU-v3 single TensorCore baseline. Improvements are additive; for example, FAST fusion includes both datapath and scheduling improvements.
DataScale for Terabyte Sized Complex Models

High Compute Capability

Dataflow Efficiency

Large off-chip Memory Capacity

Up to 40x more memory than GPU systems
Train Large NLP Models

1T parameter NLP training with a small footprint and programming ease

<table>
<thead>
<tr>
<th>Model size</th>
<th>Attention heads</th>
<th>Hidden size</th>
<th>Number of layers</th>
<th>Number of parameters (bil)</th>
<th>Model parallel size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7B</td>
<td>24</td>
<td>2304</td>
<td>24</td>
<td>1.7</td>
<td>1</td>
</tr>
<tr>
<td>3.6B</td>
<td>32</td>
<td>3072</td>
<td>30</td>
<td>3.6</td>
<td>2</td>
</tr>
<tr>
<td>7.5B</td>
<td>32</td>
<td>4096</td>
<td>36</td>
<td>7.5</td>
<td>4</td>
</tr>
<tr>
<td>18B</td>
<td>48</td>
<td>6144</td>
<td>40</td>
<td>18.4</td>
<td>8</td>
</tr>
<tr>
<td>39B</td>
<td>64</td>
<td>8192</td>
<td>48</td>
<td>39.1</td>
<td>16</td>
</tr>
<tr>
<td>76B</td>
<td>80</td>
<td>10240</td>
<td>60</td>
<td>76.1</td>
<td>32</td>
</tr>
<tr>
<td>145B</td>
<td>96</td>
<td>12288</td>
<td>80</td>
<td>145.6</td>
<td>64</td>
</tr>
<tr>
<td>310B</td>
<td>128</td>
<td>16384</td>
<td>96</td>
<td>310.3</td>
<td>129</td>
</tr>
<tr>
<td>530B</td>
<td>128</td>
<td>20480</td>
<td>105</td>
<td>529.6</td>
<td>280</td>
</tr>
<tr>
<td>1T</td>
<td>160</td>
<td>25600</td>
<td>128</td>
<td>1008.0</td>
<td>512</td>
</tr>
</tbody>
</table>

Out of Box Models
- Huggingface Models
- Write yours in Pytorch

Developer Efficiency
- Focus on ML-problems instead of System Engineering

High Accuracy Models
- No Compromise on Model Architecture required to hide System Deficiencies

Complex System Engineering to Enable Model Architecture Exploration

cWAE on SambaNova Accelerates COVID-19 Research

Algorithmic and infrastructure optimizations together yielding human impact

- Improve the efficiency of the drug discovery process
- The **character-based Wasserstein autoencoder** (cWAE) model learns faster on RDU
  - Lower latency per mini-batch
  - Pipelined training algorithm maintains model quality
- Accelerated time to train to convergence

![WAE Training on 1.613 Billion Enamine Mpro_inhib dataset](image-url)
**True-Resolution Computer Vision**

**SAMBAFLOW**
Automatically tiles the input image for deep learning operations and handles overlaps between tiles.

Tiles are streamed through model pipeline on chip.

```
model = daas_client.get_model("unet", resolution)
segments = model.predict(image)
```

Model is exposed through a simple API that works at any resolution.
Record Accuracy High-Res Convolution Training

90.23% Accuracy

World Record CosmicTagger Training Accuracy

---

**DEPARTMENT: NOVEL ARCHITECTURES**

**Accelerating Scientific Applications With SambaNova Reconfigurable Dataflow Architecture**

Murat Ercan, Venkatram Vishwanath, Corey Adams, Michael E. Papka, and Rick Stevens, Argonne National Laboratory, Lemont, IL, 60439, USA
DLRM Inference Throughput and Latency

20x Better Throughput and Latency Than A100
Making Parallelism Easy: We Can Have It All!

- Power
- Performance
- Programmability
- Portability

App Developer

Algorithms (Hogwild!)

High Performance DSLs (OptiML, PyTorch, ...)

High Level Compiler

Accelerators (GPU, FPGA, RDA)