HIGH PERFORMANCE
BIG DATA ANALYTICS

Kunle Olukotun

Electrical Engineering and Computer Science
Stanford University

June 2, 2014
“DoD is swimming in sensors and drowning in data”

Challenge: enable discovery

- Deliver the capability to mine, search and analyze this data in near real time
Goal: Make high-performance data analytics easy to use

Manipulate big data sets in real time
  - Make better decisions, the right conclusions
  - Streaming data
  - Low latency computation
  - Interactive data exploration

Requires the full power of modern computing platforms
  - Heterogeneous parallelism
Benchmarks: fib, parse_int, quicksort, mandel, pi_sum, rand_mat_stat, and rand_mat_mul

Processing PERFORMANCE TODAY

Execution Time Relative to C++
PERFORMANCE VS. EASE OF USE

<table>
<thead>
<tr>
<th>Performance</th>
<th>Ease of use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab, R</td>
<td></td>
</tr>
<tr>
<td>Pig, Hive</td>
<td></td>
</tr>
<tr>
<td>MapReduce</td>
<td></td>
</tr>
<tr>
<td>Spark</td>
<td></td>
</tr>
<tr>
<td>GraphLab</td>
<td></td>
</tr>
</tbody>
</table>

Goal
Heterogeneous parallelism

Multicore CPU

Graphics Processing Unit (GPU)

Programmable Logic

Cluster
Expert PARALLEL programming

- Threads OpenMP
- Multicore CPU
- Verilog VHDL
- Programmable Logic
- CUDA
- OpenCL
- GPU
- Graphics Processing Unit (GPU)
- MPI
- MapReduce
- Cluster
- Programmable Logic
- Expert PARALLEL programming
THe programmability GAP

Applications
- Data Wrangling
- Data Transformation
- Graph Analysis
- Prediction Recommendation

Heterogeneous Hardware
- New Arch.
general-purpose languages

Applications

- Data Wrangling
- Data Transformation
- Graph Analysis
- Prediction Recommendation

Heterogeneous Hardware

Not enough semantic knowledge to compile automatically
No restrictions

Scala
Java
Python
C++
Ruby
Clojure

New Arch.
Domain Specific Languages (DSLs)

- Programming language with restricted expressiveness for a particular domain
- High-level, usually declarative, and deterministic
Scaling the DSL Approach

- Many potential DSLs

- How do we quickly create high-performance implementations for DSLs we care about?

- Enable expert parallel programmers to easily create new DSLs
  - Make optimization knowledge reusable
  - Simplify the compiler generation process

- A few DSL developers enable many more DSL users
  - Leave expert programming to experts!
Delite: dSL infrastructure

Applications
- Data Wrangling
- Data Transformation
- Graph Analysis
- Prediction Recommendation

Domain Specific Languages
- Data Transform OptiWrangle
- Data Query OptiQL
- Graph Alg. OptiGraph
- Machine Learning OptiML
- Convex Opt. OptiCVX

Heterogeneous Hardware

Delite Common DSL Infrastructure
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler

New Arch.
Delite: dSL infrastructure

**Applications**
- Data Transformation
- Graph Analysis
- Prediction Recommendation

**Domain Specific Languages**
- Data Transform: OptiWrangle
- Data Query: OptiQL
- Graph Alg.: OptiGraph
- Machine Learning: OptiML

**Delite DSL Framework**
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler

**Heterogeneous Hardware**
- Multicore
- GPU
- FPGA
- Cluster
Delite Overview

Key elements
- Scala libraries on steroids
- Intermediate representation
- Domain specific optimization
- General parallelism and locality optimizations
- Mapping to HW targets

Delite Framework

Opti{CVX, Graph, ML, QL, Wrangle}

Domain specific analyses & transformations

Generic analyses & transformations

Code generators
- Threads
- OpenMP
- CUDA
- OpenCL
- MPI
- Verilog
optiML: a DSL for machine learning

- 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) and ML-specific ones (sum, vector constructor, untilconverged)
  - Math with MATLAB-like syntax (a*b, chol(..), exp(..))

- Runs Anywhere
  - Single source to multicore CPUs, GPUs, and clusters
until converged(kMeans, tol){
  kMeans =>
  val clusters = samples.groupRowsBy { sample =>
    val allDistances = kMeans.mapRows { mean =>
      dist(sample, mean)
    }
    allDistances.minIndex
  }
  val newKmeans = clusters.map(e => e.sum / e.length)
  newKmeans
}
Markov State Models (MSMs)
MSMs are a powerful means of modeling the structure and dynamics of molecular systems, like proteins.
### OptiGraph

- A DSL for large-scale graph analysis based on Green-Marl
  - A DSL for Real-world Graph Analysis
  - Green-Marl: A DSL for Easy and Efficient Graph Analysis (Hong et. al.), ASPLOS ’12

- Functional DSL
  - No mutation, no explicit iteration

- Data structures
  - Graph (directed, undirected), node, edge,
  - Set of nodes, edges, neighbors, …

- Graph traversals
  - Summation, Breadth-first traversal, …

- Parallel reductions but no deferred assignment

- Example applications written in framework
  - Betweenness Centrality
  - Directed/Undirected Triangle Counting
  - PageRank
Example: Betweenness Centrality

- **Betweenness Centrality**
  - A measure that tells how ‘central’ a node is in the graph
  - Used in social network analysis
- **Definition**
  - How many shortest paths are there between any two nodes going through this node.

\[ C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \]
OptiGraph Betweenness Centrality

1. val bc = sum(g.nodes) { s =>
2.   val sigma = g.inBfOrder(s) { (v, prev_sigma) => if(v == s) 1
3.     else sum(v.upNbrs) { w => prev_sigma(w) }
4.   }
5.   val delta = g.inRevBfOrder(s) { (v, prev_delta) =>
6.     sum(v.downNbrs) { w =>
7.       sigma(v) / sigma(w) * (1 + prev_delta(w))
8.     }
9.   }
10.  delta
11. }
OptiGraph: Page Rank

1. `val pr = untilConverged(0, threshold) { oldPR =>
2.   g.mapNodes { n =>
3.     ((1.0 - damp)/g.numNodes) + damp*sum(n.inNbrs) { w =>
4.       oldPr(w)/w.outDegree
5.     }
6.   }
7. }{(curPR, oldPR) => sum(abs(curPr-oldPr))}
### OptiGraph vs. GPS (Pregel)

#### Higher Productivity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>OptiGraph (lines of code)</th>
<th>Native GPS (lines of code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Teenage Follower (AvgTeen)</td>
<td>13</td>
<td>130</td>
</tr>
<tr>
<td>PageRank</td>
<td>11</td>
<td>110</td>
</tr>
<tr>
<td>Conductance (Conduct)</td>
<td>12</td>
<td>149</td>
</tr>
<tr>
<td>Single Source Shortest Paths (SSSP)</td>
<td>29</td>
<td>105</td>
</tr>
<tr>
<td>Random Bipartite Matching (Bipartite)</td>
<td>47</td>
<td>225</td>
</tr>
<tr>
<td>Approximate Betweenness Centrality</td>
<td>25</td>
<td>Not Available</td>
</tr>
</tbody>
</table>

DSL compiler automatically converts OptiGraph to GPS.
(Pregel) Similar Performance

25 x 4 = 100 cores

![Bar chart showing OptiGraph execution time relative to native GPS](chart.png)
Microsoft Research, IBM, Logic Blox, MIT, and Oracle are trying to make this application run fast.

Simple yet practical example of multi-way join—a fundamental operation to any data analytics engine.

Serves as the building block to many graph mining applications such as identifying cliques, graph transitivity, and clustering coefficients.

1. val triangleCount = g.sumOverNodes{ n =>
2. sum(n.nbrs){nbr => n.commonNbrs(nbr).size } }
KeyS to TRIANGLE COUNTING PERFORMANCE

- Data layout
  - Hash
  - Compressed sparse row
  - Bit set
  - Compressed bit set
- Dynamically switch based on graph characteristics
  - E.g. mesh vs. scale free (power law)
  - Mixed layouts are best in some cases
- Dynamic scheduling for load balance
- Performance range: 2x–160x better than Graphlab