High Performance Big-Data Analytics

Kunle Olukotun
Pervasive Parallelism Laboratory
Stanford University
ppl.stanford.edu
Big Data Analytics Today

Disk-to-disk map-reduce data processing
Next Generation Big Data Analytics: Improved Decision Making

- Higher performance ⇒ faster decisions
  - Bigger data sizes ⇒ better decisions
  - Low latency big data processing ⇒ interactive decisions
  - Processing on live data streams ⇒ real time decisions

- Higher productivity ⇒ easier decisions
  - More intuitive than map-reduce with key-value pairs
  - Simple programming for complex tasks
    - Data transformation
    - Graph analysis
    - Predictive analysis using machine learning
Next Gen Big Data Analytics Must Embrace Heterogeneous Parallelism

Fine grained parallelism is the only way to get high performance and performance/watt
Heterogeneous Parallel Programming

- Pthreads
- OpenMP
- Multicore
- CUDA
- OpenCL
- GPU
- MPI
- PGAS
- Cluster
- Verilog
- VHDL
- FPGA
Huge Performance Variation: Image Filtering OpenMP Assignment

Optimizations:
- Precomputing twiddle
- Not computing what isn’t part of the filtering
- Transposing the matrix
- Using SSE

~3 orders of magnitude
Big-Data Analytics Programming Challenge

Data Analytics Application
- Data Prep
- Data Transform
- Network Analysis
- Prediction

Pthreads OpenMP
- Multicore

CUDA OpenCL
- GPU

MPI PGAS
- Cluster

Verilog VHDL
- FPGA
Big-Data Analytics Programming Challenge

Data Analytics Application

- Data Prep
- Data Transform
- Network Analysis
- Prediction

Domain Specific Languages

- Pthreads
- OpenMP
- Multicore
- CUDA
- OpenCL
- GPU
- MPI
- PGAS
- Cluster
- Verilog
- VHDL
- FPGA
Domain Specific Languages (DSLs)

- Definition: A language or library with restrictive expressiveness that exploits domain knowledge for productivity and efficiency
- High-level, usually declarative, and deterministic
Benefits of Using DSLs for High Performance

**Productivity**
- Shield most 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 more easily and fully extract available parallelism
- Use domain knowledge 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
Our Approach: Data Analytics DSLs

Applications
- Data Transform
- Data Wrangling
- Social network Analysis
- Predictions

Domain Specific Languages (Scala)
- Data Prep
  - OptiWrangle
  - DSL Compiler
- Data Query
  - OptiQL
  - DSL Compiler
- Graph Alg.
  - OptiGraph
  - DSL Compiler
- Machine Learning
  - OptiML
  - DSL Compiler
- Convex Opt.
  - OptiCVX
  - DSL Compiler

Heterogeneous Hardware

New Arch.
Delite: DSL Infrastructure

Applications
- Data Transform
- Data Wrangling
- Social network Analysis
- Predictions

Domain Specific Languages (Scala)
- Data Prep OptiWrangle
- Data Query OptiQL
- Graph Alg. OptiGraph
- Machine Learning OptiML
- Convex Opt. OptiCVX

DSL Infrastructure
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler

Heterogeneous Hardware

New Arch.
**Delite Overview**

**Key elements**
- DSLs embedded in Scala
- IR created using staging
- Domain specific optimization
- General parallelism and locality optimizations
- Mapping to HW targets
Delite: DSL Examples

Applications
- Data Transform
- Data Wrangling
- Social network Analysis
- Predictions

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

DSL Infrastructure
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler

Heterogeneous Hardware
New Arch.
Big Data Analytics Systems

Berkeley in memory framework for interactive queries and iterative computations

Hadoop
Spark
Delite

HDFS
Mesos

Processing
Storage management
Cluster resource management
OptiQL

// lineItems: Table[LineItem]
val q = lineItems
  Where(_.l_shipdate <=
    Date("1998-12-01"))
  GroupBy(l => 1.l_linestatus).
  Select(g => new Result {
    val linestatus = g.key
    val sumQty = g.Sum(_.l_quantity)
    val sumDiscountedPrice =
      g.Sum(l => 1.l_extendedPrice* (1.0-1.l_discount))
    val avgPrice =
      g.Average(_.l_extendedPrice)
    val countOrder = g.Count
  })
  OrderBy(_.returnFlag)
  ThenBy(_.lineStatus)

- In-memory data querying
- LINQ, SQL like
- Key operations are query operators on the Table data structure
  - User-defined schema
- Optimizations:
  - Fusion eliminates temporary allocations
  - Eliminate fields not used in query
TPC-H Query 1 on 20 x 4 cores

![Graph showing speedup over Hadoop for I/O and Compute tasks.](image)
OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning, ICML 2011

- 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
  - `Vector[T], Matrix[T], Stream[T]`
  - `val c = (0::100) { i => i*2 }` // vector constructor

- Implicitly parallel control structures
  - `sum{...}, (0::end) {...}, gradient { ... }, untilconverged { ... }
  - Allow anonymous functions with restricted semantics to be passed as arguments of the control structures
OptiML: $k$-means Clustering

until_converged(mu, tol) {
  mu =>
  // Find closest centroid to each sample

  // move each cluster centroid to the
  // mean of the samples assigned to it

  }

OptiML: \textit{k}-means Clustering

\texttt{untilconverged}(\texttt{mu}, \texttt{tol})\{ \\
\texttt{mu} \Rightarrow \\
// \text{Find closest centroid to each sample} \\
\texttt{val} \ c = (\emptyset::\texttt{m})\{ \texttt{i} \Rightarrow \\
\quad \texttt{val} \ \texttt{allD}


distances = \texttt{mu} \ \texttt{mapRows} \{ \texttt{centroid} \Rightarrow \\
\quad \texttt{dist}(\texttt{samples}(\texttt{i}), \ \texttt{centroid}) \\
\}
\}
\texttt{allDistances}.\texttt{minIndex}
\\
// move each cluster centroid to the
// mean of the samples assigned to it
\}
until converged (mu, tol) {
  mu =>
  // Find closest centroid to each sample
  val c = (0::m)(i =>
    val allDistances = mu mapRows { centroid =>
      dist(samples(i), centroid)
    }
    allDistances.minIndex
  }

  // move each cluster centroid to the mean of the samples assigned to it
  val newMu = (0::k,*) (cluster =>
    val weightedpoints =
      sumRowsIf(0,m)(i => c(i) == cluster)(i => samples(i))
    val d = c.count(i => i == cluster)
    weightedpoints / d
  )
  newMu
}
Machine Learning on 20 x 4 cores: Library vs. Compiler

For k-means:
- 1.7 GB: Spark 60, Delite 300
- 17G: Spark 70, Delite 150

For Logistic Regression:
- 3.4GB: Spark 100, Delite 500
- 17G: Spark 50, Delite 350
Machine Learning on 4 x 12 cores and 4 x GPU

- k-means
  - Spark: 1.0
  - Delite CPU: 4.0
  - Delite GPU: 7.0

- Logistic Regression
  - Spark: 1.0
  - Delite CPU: 3.0
  - Delite GPU: 8.0
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

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

- Graph iteration
  - Normal parallel iteration, Breadth-first iteration, Topological Order, ...

- Deferred assignment and parallel reductions (Bulk synchronous consistency)
OptiGraph: PageRank

Implicitly parallel iteration

for(t <- G.Nodes) {
  val rank = ((1.0 d)/ N) +
    d * Sum(t.InNbrs){w => PR(w) / w.OutDegree}
  PR <= (t,rank)
  diff += Math.abs(rank - PR(t))
}

Deferred assignment and scalar reduction

Writes become visible after the loop completes
Green-Marl vs. GPS (Pregel): Lines of Code

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Green-Marl</th>
<th>Native GPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Teenage Follower (AvgTeen)</td>
<td>13</td>
<td>130</td>
</tr>
<tr>
<td>PageRank</td>
<td>19</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 Betweeness Centrality</td>
<td>25</td>
<td>Not Available</td>
</tr>
</tbody>
</table>
Green-Marl vs. GPS (Pregel) on 20 x 4 cores
Conclusions

- DSLs are the key to next generation big data analytics
  - High Productivity: higher level abstractions
  - High performance: fine-grained parallelism

- Sophisticated compilers needed to make sense of high-level, domain-specific abstractions

- Performance advantage of compiling DSLs is substantial

- http://ppl.stanford.edu
DSLs: Barriers to High Performance

- Problem 1: abstraction penalty
  - Staging: remove abstraction programmatically using partial evaluation

- Problem 2: compiler lacks semantic knowledge
  - Extend compiler with high-level knowledge
    - E.g. Teach compiler linear algebra

- Problem 3: compiler lacks parallelism knowledge
  - Extend the compiler with parallelism and locality knowledge

- Solving any of the problems alone will not result in high performance
Markov State Models (MSMs)

MSMs are a powerful means of modeling the structure and dynamics of molecular systems, like proteins.

MSM Builder Using OptiML with Vijay Pande