Big Data Analytics with Delite

Kevin J. Brown, Arvind K. Sujeeth, HyoukJoong Lee, Tiark Rompf, Christopher De Sa, Martin Odersky, Kunle Olukotun

Stanford University, EPFL
Big data is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data.

IBM What is big data? - Bringing big data to the enterprise

Big data will become the foundation for innovation, underpinning new waves of productivity and fuelling the creation of consumer surplus—as long as the right policies...

Learn about big data challenges and opportunities, along with how to apply the latest strategies and technologies to extract maximum value from big data.
Heterogeneous Parallel Architectures Today

- High performance capability

Multicore CPU

GPU

Cluster
Heterogeneous Parallel Programming

- But high effort

- Threads
- OpenMP
- Multicore CPU
- CUDA
- OpenCL
- GPU
- Verilog
- VHDL
- MPI
- MapReduce
- Cluster
Programmability Chasm

Applications

Scientific Engineering
Virtual Worlds
Personal Robotics
Data Informatics

Too many different programming models
Bridging the Programmability Chasm

Applications
- Scientific Engineering
- Virtual Worlds
- Personal Robotics
- Data informatics

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

Heterogeneous Hardware
- New Arch.
Big Data Pipeline

Data Cleaning (OptiWrangler)

Data Querying (OptiQL)

- Machine Learning (OptiML)
- Graph Analysis (OptiGraph)

Visualization (OptiVis)
Domain Expertise

Gradient Descent
Images, Video, Audio
Convex Optimization
Probability
Linear Algebra

Message-passing graphs
Streaming training sets

Expressing the important problems
Language Expertise

Abstract Syntax Tree
Program Transformation

Control Flow Graph

Alias Analysis

Code Generation

Loop-invariant Code Motion

Elegant, natural and simple design
Performance Expertise

Thread
False Sharing
Locality
Mutex
SSE
Synchronization

Implementing efficiently and portably
Common DSL Infrastructure

Applications
- Scientific Engineering
- Virtual Worlds
- Personal Robotics
- Data Informatics

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

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

Heterogeneous Hardware
- New Arch.
Delite DSL Framework

Applications
- Scientific Engineering
- Virtual Worlds
- Personal Robotics
- Data informatics

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

Embedded Language (Scala) + DSL Framework (Delite)
- Polymorphic Embedding
- Staging
- Static Domain Specific Opt.

Parallel Runtime (Delite RT)
- Task & Data Parallelism
- Locality Aware Scheduling

Delite DSL Infrastructure

Heterogeneous Hardware

New Arch.
Outline

- Building high performance DSLs for heterogeneous cluster computing in Delite
- Case study: Tackling nested parallel patterns on clusters with existing Delite DSLs
- Performance results
Delite Programming Model

- **Parallel Patterns**
  - Map, Zip, Filter, FlatMap, Reduce, GroupBy, ...
  - Composable
    - e.g., Filter-GroupBy
    - e.g., Map{Reduce}

- **Restricted Data**
  - Records of primitives and arrays
    - Per-target implementations

Diagram:
- Application
  - Multiple DSLs
    - Domain data
      - Domain ops
    - Parallel data
      - Parallel ops
    - Generic analyses & transformations
    - Code generators
Delite Example

- **DSL author writes:**

```scala
trait VectorOps {
  trait Vector[A] //user-facing types (abstract)

  //DSL methods on abstract types create domain-specific IR nodes
  def infix_max[A:Manifest:Ordering](v: Rep[Vector[A]]) = VectorMax(v)

  //DSL ops implemented using Delite parallel patterns; Delite handles codegen
    def func = (a,b) => if (a > b) a else b
  }

  //DSL data structures implemented using Delite structs
  case class VectorNew[A:Manifest](length: Rep[Int]) extends DeliteStruct[Vector[A]] {
    val elems = ("_data" -> DeliteArray[A](length), "_length" -> length)
  }
}
```
DSL Optimizations

```scala
trait MatrixOpsOpt extends MatrixOps {
  override def matrix_plus[A:Manifest:Arith](x: Rep[Matrix[A]], y: Rep[Matrix[A]]) = (x, y) match {
    // (AB + AD) == A(B + D)
    case (Def(MatrixTimes(a, b)), Def(MatrixTimes(c, d))) if (a == c) =>
      matrix_times(a, matrix_plus(b, d)) //return optimized version

    //case ... (other rewrites)
    case _ => super.matrix_plus(x, y)
  }
}

trait MyDSL extends VectorOpsOpt with MatrixOpsOpt
trait MyApp extends MyDSL {
  def main() {
    val v = Vector(1,2,3,4,5)
    v.max //can run on CPU, GPU, across a cluster, ...
  }
}
```
Delite Advantages

- *Parallel pattern IR* makes it easy for DSL authors to expose parallelism in their domain

- Provides code generators for Scala, C++, OpenCL, CUDA, and clusters thereof

- Uses staging (LMS) to build the IR (POPL ‘13)
  - Systematically removes abstraction
  - Simple transformation interface based on rewrites
    - Each DSL can add new domain-specific optimizations

- Provides reusable, composable optimizations
  - High-level operator fusion, AoS to SoA, code motion, dead field elimination, CSE
Outline

- Building high performance DSLs for heterogeneous cluster computing with Delite

- Case study: Tackling nested parallel patterns on clusters with existing Delite DSLs

- Performance results
Nested Parallel Patterns

- Parallelization decisions are much more difficult
  - What if a big loop nested inside a small loop?
  - Unroll the outer loop, flatten the loops, interchange the loops, ...

- Nesting order determines the data access stencil
  - Affects freedom to physically partition data across the cluster
  - Want to distribute over the “big” dataset

- Optimal traversal order is architecture dependent
  - No one “correct” way of writing the program
  - Often reversed for CPU vs. GPU (row-oriented vs. column-oriented)

- Compiler needs to be able to transform between both versions to maintain architecture-agnostic source code
OptiQL

- Data querying of in-memory collections
  - inspired by LINQ to Objects

- SQL-like declarative language

- Use high-level semantic knowledge to implement query optimizer
OptiQL: TPCH-Q1

// lineItems: Table[LineItem]
// Similar to Q1 of the TPC-H benchmark
val q = lineItems Where(_.l_shipdate <= Date('19981201')).
  GroupBy(l => l.l_linenumber).
  Select(g => new Record {
    val lineStatus = g.key
    val sumQty = g.Sum(_.l_quantity)
    val sumDiscountedPrice =
      g.Sum(r => r.l_extendedprice*(1.0-r.l_discount))
    val avgPrice = g.Average(_.l_extendedprice)
    val countOrder = g.Count
  }) OrderBy(_.returnFlag) ThenBy(_.lineStatus)

hoisted
fused
TPC-H LineItem

- User-defined struct

```haskell
type LineItem = Record {
  val l_orderkey: Int;
  val l_suppkey: Int;
  val l_quantity: Double;
  val l_discount: Double;
  val l_returnflag: Char;
  val l_shipdate: Date;
  val l_receiptdate: Date
  val l_linenum: Int
  val l_linenumber: Int
  val l_partkey: Int
  val l_extendedprice: Double
  val l_tax: Double
  val l_linestatus: Char
  val l_commitdate: Date
  val l_shipinstruct: String
  val l_shipmode: String
  val l_comment: String
}
```
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
  - Subtypes: `TrainingSet`, `IndexVector`, `Image`, ...

- Implicitly parallel control structures
  - `sum{...}`, `(0:end) {...}`, `gradient { ... }`, `untilconverged { ... }
  - Arguments to control structures are anonymous functions with restricted semantics
until converged (\( \mu, \text{tol} \))
{
  \( \mu \Rightarrow \)
  // assign each sample to the closest centroid
  val clusters = x.groupRowsBy { row =>
    // calculate distances to current centroids
    val allDistances = mu.mapRows { centroid =>
      dist(row, centroid)
    }
    allDistances.minIndex
  }

  // move each cluster centroid to the
  // mean of the points assigned to it
  val newMu = clusters.map(e => e.sum / e.length)
  newMu
}
OptiML: \( k \)-means Clustering (2)

```scala
untilConverged(mu, tol){ mu =>
  // calculate distances to current centroids
  val c = (0::m){i =>
    val allDistances = mu mapRows { centroid =>
      dist(x(i), centroid)
    }
    allDistances.minIndex
  }

  // move each cluster centroid to the
  // mean of the points assigned to it
}
```
OptiML: $k$-means Clustering (2)

```scala
until_converged(mu, tol) { 
  mu =>
  // calculate distances to current centroids
  val c = (0::m)(i => 
    val all_distances = mu map { 
      centroid =>
        dist(x(i), centroid)
    }
    all_distances.minIndex
  }

  // move each cluster centroid to the
  // mean of the points assigned to it
  val new_mu = (0::k,*) { cluster =>
    val weighted_points =
      sum_rows_if(0, m)(i => c(i) == cluster)(i => x(i))
    val points = c.count(i => i == cluster)
    weighted_points / points
  }
  new_mu
}
```
OptiML: \( k \)-means Clustering (3)

\[
\text{untilconverged}(\mu, \text{tol})\{ \mu \mapsto \\
// \text{calculate distances to current centroids} \\
\text{val } c = (\emptyset::m)\{ i \mapsto \\
\text{val } \text{allDistances} = \mu \map{\text{mapRows}} \{ \text{centroid } \mapsto \\
\text{dist}(x(i), \text{centroid}) \\
\}
\text{allDistances.minIndex} \\
\}
\text{val } \text{allWP} = \text{bucketReduce}(\emptyset::m)(i \mapsto c(i), i \mapsto x(i), _ + _) \\
\text{val } \text{allP} = \text{bucketReduce}(\emptyset::m)(i \mapsto c(i), i \mapsto 1, _ + _)
\]

\[
\text{val } \text{newMu} = (\emptyset::k,*)\{ \text{cluster } \mapsto \\
\text{val } \text{weightedpoints} = \text{allWP(cluster)} \\
\text{val } \text{points} = \text{allP(cluster)} \\
\text{weightedpoints} / \text{points} \\
\}
\text{newMu}
\}
\]
OptiML: Logistic Regression

```scala
untilconverged(theta, tol){ theta =>
  (0::x.numFeatures){ j =>   //vector of sums
    val gradient = sum(0, x.numSamples){ i =>
      x(i)(j)*(y(i) - hyp(theta,x(i)))
    }
    theta(j) + alpha*gradient
  }
}

untilconverged(theta, tol){ theta =>
  val gradientVec = sum(0, x.numSamples){ i => //sum of vectors
    (0::x.numFeatures){ j =>
      x(i)(j)*(y(i) - hyp(theta,x(i)))
    }
  }
  (0::x.numFeatures){ j =>
    val gradient = gradientVec(j)
    theta(j) + alpha*gradient
  }
}
```
Logistic Regression on the GPU

- For GPU execution the original traversal order is actually superior!
  - Better to compute a vector of multiple scalar sums than a sum of vectors
- But we need the transformation to optimally partition the app across a cluster

Solution: Apply the inverse transformation within the CUDA kernel implementation and transpose each chunk of the input matrix when shipped to the GPU
- Distribute matrix by samples (rows) across the cluster, iterate and sum by features (columns) within each GPU
Stencil Analysis and Data Partitioning

- Delite analyzes the access pattern for each input of a parallel op
  - Possible Stencils: One, Interval (distribute); All (broadcast); Unknown (runtime message passing)

- Stencil for each op is joined conservatively to determine a partition for each data structure / schedule for each op
  - Consider constraints on input locations & constraints on output locations
  - Attempt to create a schedule that requires no data re-shuffling
Runtime Management

- Runtime uses master/slave model for cluster
  - Master runs all effectful / sequential ops
  - Pure parallel ops can be executed across multiple slaves using RPC calls
  - Efficient serialization using Protocol Buffers
  - Each slave can utilize multi-core and GPU

- Each slave keeps its portion of distributed data structures in memory for future ops
  - Garbage collection handled using DEG information (liveness analysis)
  - GC the GPU’s memory in a similar way
Outline

- Building high performance DSLs for heterogeneous cluster computing with Delite
- Case study: Tackling nested parallel patterns on clusters with existing Delite DSLs
- Performance results
Experimental Setup

- Amazon EC2 Cluster: 20 nodes
  - m1.xlarge instances
    - 4 virtual cores, 15GB RAM, 1Gb Ethernet

- Local Cluster: 4 nodes
  - 12 Intel Xeon X5680 cores (2 sockets)
  - 48 GB RAM
  - NVIDIA Tesla C2050 GPU
  - 1Gb Ethernet
3 versions of every app

1) Deliteful languages (OptiML, OptiQL)

2) Apache Hadoop

3) Spark
   - Scala library for cluster parallelization
   - Solves many of the inefficiencies related to Hadoop by keeping data in memory
   - Provides data-parallel operators on distributed datastructures
   - Stay for the next talk!
OptiQL: TPC-H Query 1 & OptiML: GDA

20 node Amazon cluster
Q1: TPC-H 5GB dataset; GDA: 17GB dataset
OptiML: $k$-means & logistic regression

20 node Amazon cluster
OptiML: $k$-means and logistic regression

4 node local cluster: 3.4 GB dataset
Strong Scaling Results on Amazon

$k$-means 50MB dataset
(execution time in seconds above bars)
Thank You!

- Delite repository

- Stanford Pervasive Parallelism Lab
  - Links to publications and related projects
  - http://ppl.stanford.edu