OPTIML

Arvind K. Sujeeth, HyoukJoong Lee, Kevin J. Brown, Hassan Chafi,
Tiark Rompf, Anand Atreya, Michael Wu, Kunle Olukotun

Stanford University
Pervasive Parallelism Laboratory (PPL)

Outline

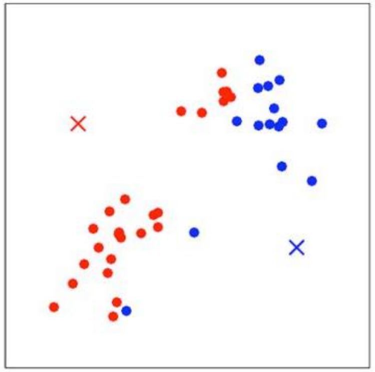
- OptiML as an example of DSL design
  - How should we approach designing an embedded DSL?
- OptiML as an example of DSL implementation
  - How does OptiML actually work with LMS and Delite?
  - You will probably use OptiML as an example for building your own DSL, so we'll walk through the code and make sure its familiar
- You don't have to do things exactly the way I did
  - Try stuff; if you come up with better organizations, let me know!

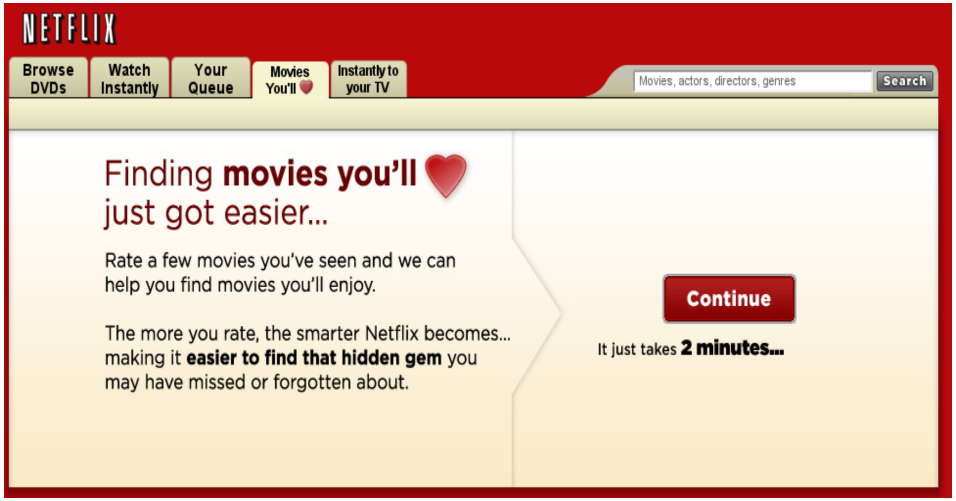
OptiML: A DSL for ML

- Machine Learning domain
  - Learning patterns from data
  - Applying the learned models to tasks
    - Regression, classification, clustering, estimation
  - Computationally expensive
  - Regular and irregular parallelism

- Characteristics of ML applications
  - Iterative algorithms on fixed structures
  - Large datasets with potential redundancy
  - Trade off between accuracy for performance
  - Large amount of data parallelism with varying granularity
  - Low arithmetic intensity

Machine Learning Examples





Finding **movies you'll** ♥ just got easier...

Rate a few movies you've seen and we can help you find movies you'll enjoy.

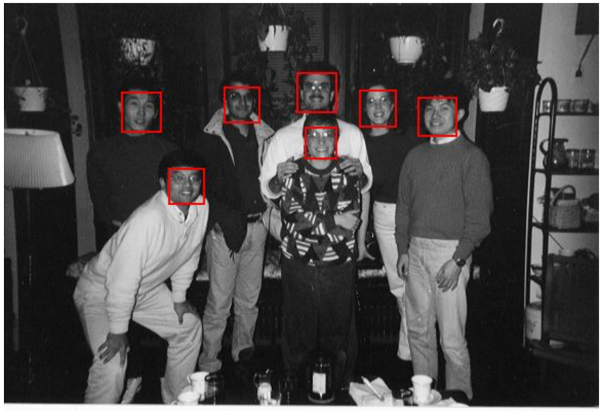
The more you rate, the smarter Netflix becomes... making it **easier to find that hidden gem** you may have missed or forgotten about.

It just takes **2 minutes...**

reCAPTCHA™

Google translate

Report Spam



OptiML: Motivation

- Raise the level of abstraction
  - Focus on algorithmic description, get parallel performance
- Use domain knowledge to identify coarse-grained parallelism
  - Identify parallel and sequential operations in the domain (e.g. ‘batch gradient descent’)
- Single source => Multiple heterogeneous targets
  - Not possible with today’s MATLAB support
- Domain specific optimizations
  - Optimize data layout and operations using domain-specific semantics
- A driving example
  - Flesh out issues with the common framework, embedding etc.

Domain model

- What specific problems are we trying to solve?
  - Statistical inference problems, especially those that can be expressed in summation form (called the statistical query model)
    - Represented by vectors / matrices
    - Represented by graphical models
- Why is this interesting?
  - Expresses a large slice of machine learning algorithms
  - Well known parallel patterns (we can abstract parallel details)

What is the domain knowledge we have?

- Algorithm characteristics
  - Bandwidth more important than latency
  - Low arithmetic intensity on average (tend to be memory-bound)
  - Usually more data parallelism than task parallelism

- Kernel structure
  - Large training sets (GBs or larger)
    - Often a 2-class problem (2-d vector of labels)
    - Streaming model
    - Multiple passes through training set sometimes required
  - Most kernels use dense and/or sparse matrix/vector operations
  - Iterative algorithms (run the same kernel repeatedly many times)
    - Especially with parameter updates/constrained optimization (e.g. gradient descent)

Domain knowledge, continued

- Redundancy in training data
  - Many large datasets (e.g. network traffic) contain significant redundancy and a smaller set of distinct patterns
    - Best-effort computing

- Fuzzy correctness
  - Probabilistic
  - Trade-off of computation for accuracy
    - Relaxed dependencies
    - Approximate convergence

- Common types of input data with fixed structure and characteristics
  - Images (RGB, gray-scale, etc.)
  - Video (format, codec, framerate, etc.)
  - Audio (sampling rate)
  - Text

How do we encode/exploit it?

- In implementation of high level data and control structures
  - TrainingSet, PackedBooleanVector, SparseMatrix, FactorGraph, DecisionTree, MarkovChain, GrayScaleImage, RGBImage, AudioSample, LowResVideo...
  - Sum, filter, foreach edge, …

- In deciding what data to transfer and when to transfer to GPU (or other heterogeneous nodes)
  - Dynamic compression (especially for sparse data)
  - Hints to the runtime
    - Arithmetic intensity of an op
    - Lifetime of data (e.g. transient, permanent, etc.)

- In domain-specific optimizations
  - Best Effort vectors or ops
  - Fusing (For common op-sequences, reduces working set size and data communication costs)
  - Locality (e.g. Localizing operations on the same data across op instances, such as in two dot products: <A,X>, <A,Y>)

What are the semantics and syntax that can improve productivity?

- MATLAB like syntax
  - Array indexing, slicing using [a:b:end]
  - .*, *, ^, `
- Math like syntax
  - Reduction/accumulator/sum: sum from i to m { .. }
- Higher level abstraction
  - optimize / batch gradient descent / until converged {…}
  - train / classify
- Anonymous and Higher order functions
  - Kernels as first class objects
  - Map, filter, etc.
- Side-effects
  - Need to retain the ability to mutate state, e.g. A[5] = x

OptiML: Overview

- 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
  - General data types : Vector[T], Matrix[T]
    - Independent from the underlying implementation
  - Special data types : TrainingSet, TestSet, IndexVector, Image, Video ..
    - Encode semantic information

- Implicitly parallel control structures
  - sum{…}, (0::end) {…}, gradient { … }, untilconverged { … }
  - Allow anonymous functions with restricted semantics to be passed as arguments of the control structures

Example OptiML / MATLAB code (Gaussian Discriminant Analysis)

ML-specific data types

```
// x : TrainingSet[Double]
// mu0, mu1 : Vector[Double]

val sigma = sum(0,x.numSamples) {
    if (x.labels(_) == false) {
        (x(_)-mu0).trans.outer(x(_)-mu0)
    }
    else {
        (x(_)-mu1).trans.outer(x(_)-mu1)
    }
}
```

Implicitly parallel control structures

Restricted index semantics

```
% x : Matrix, y: Vector
% mu0, mu1: Vector

n = size(x,2);
sigma = zeros(n,n);

parfor i=1:length(y)
    if (y(i) == 0)
        sigma = sigma + (x(i,:)-mu0)'*(x(i,:)-mu0);
    else
        sigma = sigma + (x(i,:)-mu1)'*(x(i,:)-mu1);
    end
end
```

OptiML code

(parallel) MATLAB code

MATLAB implementation

- `parfor` is nice, but not always best
  - MATLAB uses heavy-weight MPI processes under the hood
  - Precludes vectorization, a common practice for best performance
  - GPU code requires different constructs

- The application developer must choose an implementation, and these details are all over the code

```
ind = sort(randsample(1:size(data,2),length(min_dist)));
data_tmp = data(:,ind);
all_dist = zeros(length(ind),size(data,2));
parfor i=1:size(data,2)
    all_dist(:,i) = sum(abs(repmat(data(:,i),1,size(data_tmp,2)) -
data_tmp),1)';
end
all_dist(all_dist==0)=max(max(all_dist));
```

OptiML vs. MATLAB

- OptiML
  - Statically typed
  - No explicit parallelization
  - Automatic GPU data management via run-time support
  - Inherits Scala features and tool-chain
  - Machine learning specific abstractions

- MATLAB
  - Dynamically typed
  - Applications must explicitly choose between vectorization or parallelization
  - Explicit GPU data management
  - Widely used, numerous libraries and toolboxes

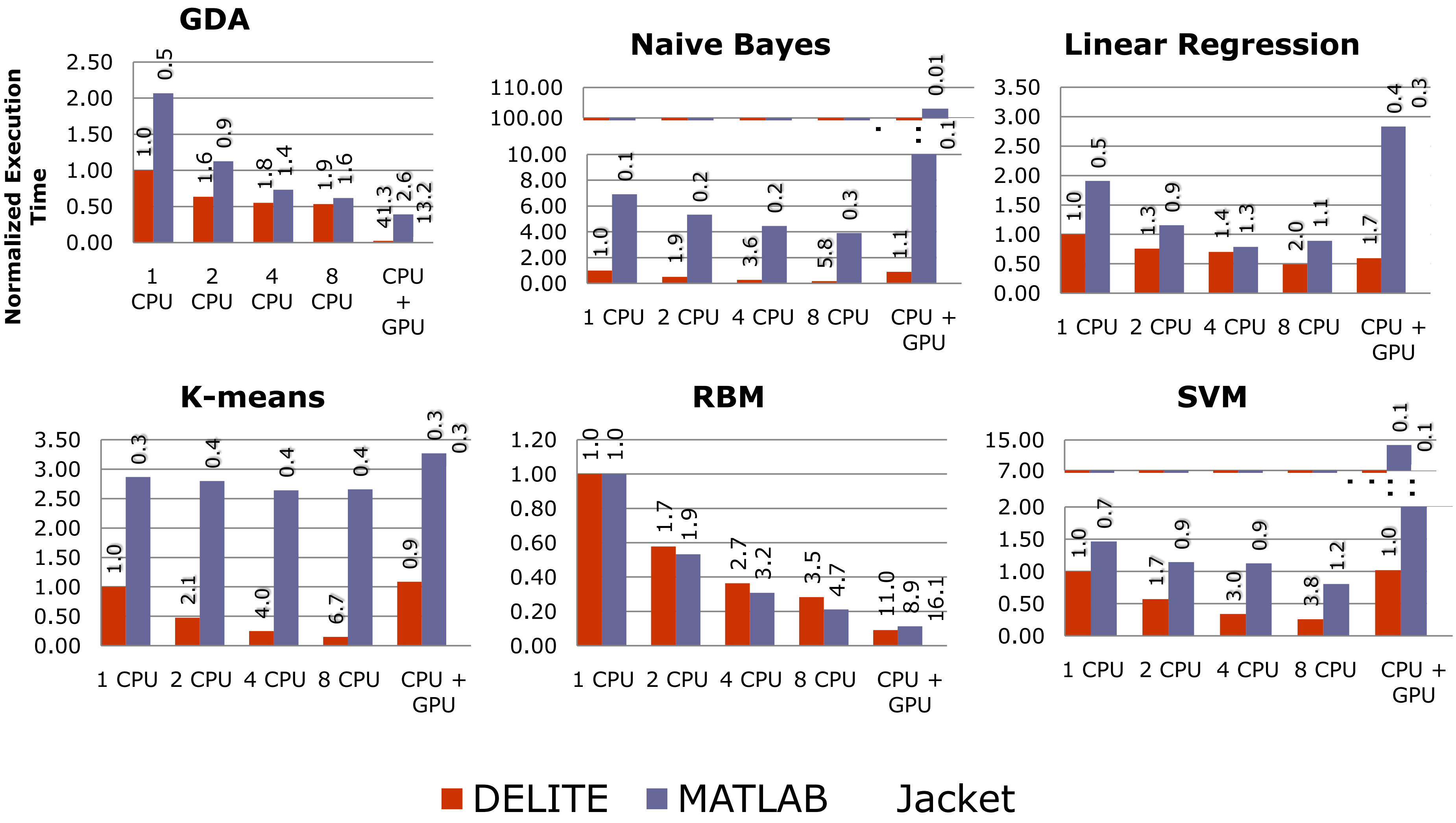
Static Optimizations

- All the general ones we talked about for free from LMS
  - Code motion, CSE, DCE
- Pattern rewritings
  - Linear algebra simplifications
  - Shortcuts to help fusing
- Op fusing
  - can be especially useful in ML due to fine-grained operations and low arithmetic intensity
- Remember: coarse-grained
  - Optimizations happen on vectors and matrices

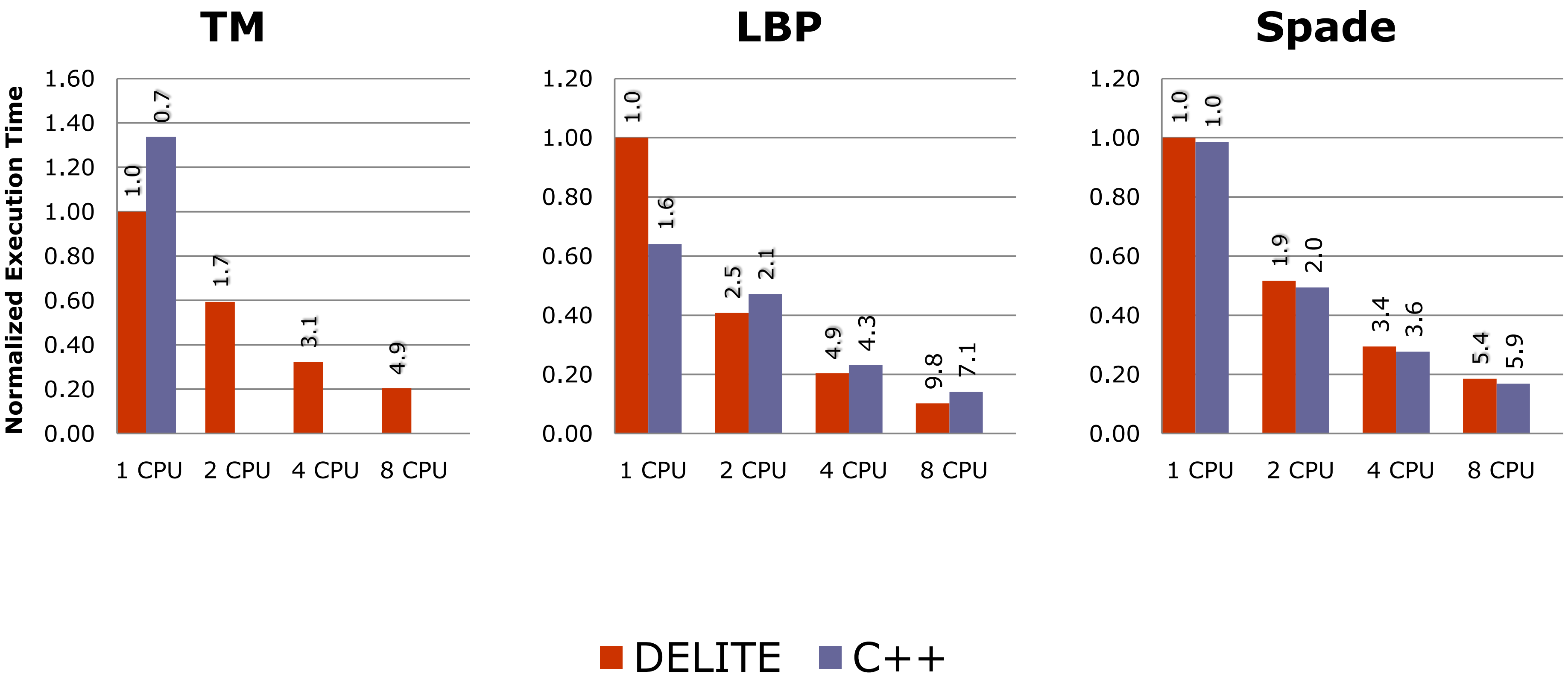
Dynamic Optimizations

- Relaxed dependencies
  - Iterative algorithms with inter-loop dependencies prohibit task parallelism
  - Dependencies can be relaxed at the cost of a marginal loss in accuracy
  - Relaxation percentage is run-time configurable
- Best effort computations
  - Some computations can be dropped and still generate acceptable results
  - Provide data structures with “best effort” semantics, along with policies that can be chosen by DSL users

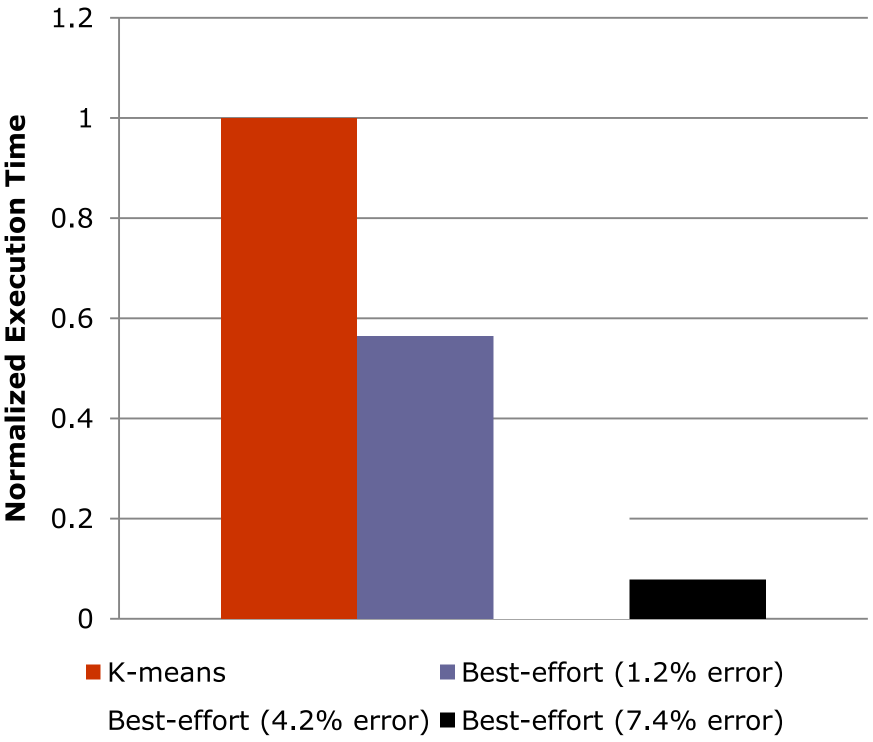
Experiments relative to MATLAB



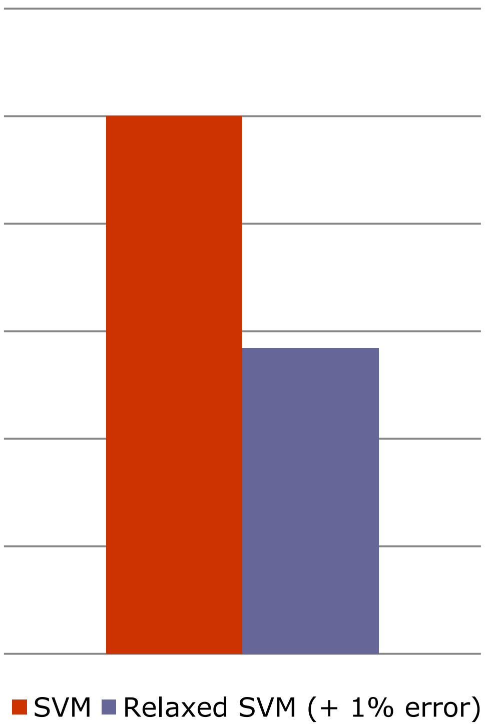
Experiments relative to C++



Domain specific optimizations (PPoPP)



K-means Best Effort



SVM Relaxed Dependencies

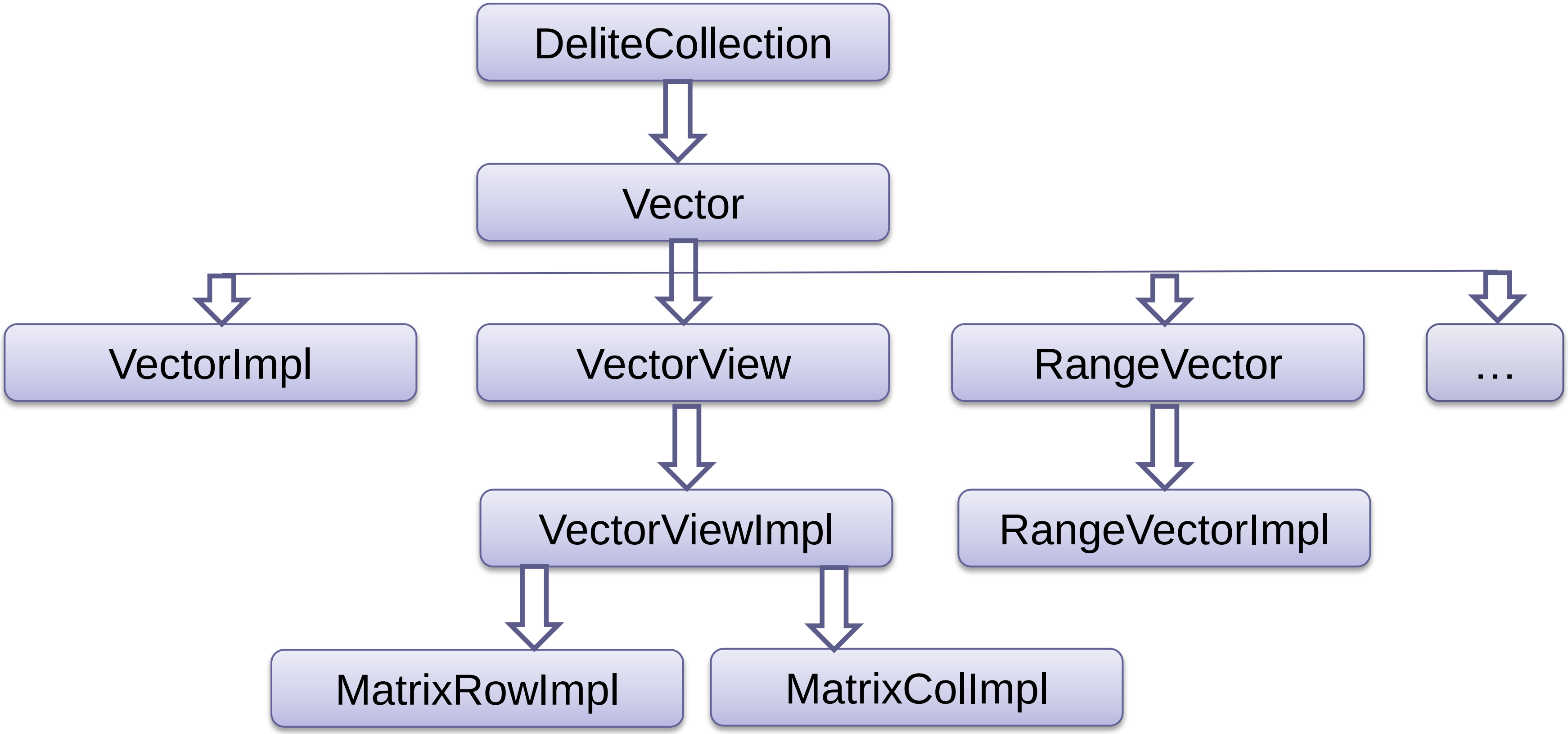
Fusing (not pictured): eliminated 17 out of 28 total loops in SVM

OPTIML IMPLEMENTATION

Data types

- Vector
  - IndexVector
  - RangeVector
  - VectorView
    - MatrixRow, MatrixCol
  - Labels
  - Edges/Vertices
- Matrix
  - Image
    - GrayscaleImage
  - TrainingSet
    - Unsupervised/Supervised (TBD)
- Stream
- Graph
  - MessageGraph

Class diagram for OptiML Vector



VectorOps.scala

```
trait VectorOps extends DSLType with Variables {
  this: OptiML =>
```

Self-type, like an import statement

```
  object Vector {
    def apply[A:Manifest](len: Int, isRow: Boolean) = vector_obj_new(unit(len), unit(isRow))
      …
  }

implicit def repVecToVecOps[A:Manifest](x: Rep[Vector[A]]) = new vecOpsCls(x)
implicit def varToVecOps[A:Manifest](x: Var[Vector[A]]) = new vecOpsCls(readVar(x))
```

Alternative to infix

```
class vecOpsCls[A:Manifest](x: Rep[Vector[A]]) {
     def length = vector_length(x)
     def isRow = vector_isRow(x)
     def apply(n: Rep[Int]) = vector_apply(x, n)
     def isEmpty = length == 0
     def first = apply(0)
      …
}
```

From scala-virtualized

```
def __equal[A](a: Rep[Vector[A]], b: Rep[Vector[A]])(implicit o: Overloaded1, mA: Manifest[A]):
      Rep[Boolean] = vector_equals(a,b)

def vector_obj_new[A:Manifest](len: Rep[Int], isRow: Rep[Boolean]): Rep[Vector[A]]
def vector_length[A:Manifest](x: Rep[Vector[A]]): Rep[Int]
…
```

Abstract impl methods

Kernels

- 3 main types:
    - Methods on underlying real datastructure
    - Sequential method implementation
        - DeliteOpSingleTask
    - Parallel method implementation
        - DeliteOpMap, DeliteOpZipWith, etc.

Methods on underlying datastructure

```
trait VectorOpsExp extends VectorOps with VariablesExp with BaseFatExp {

  this: VectorImplOps with OptiMLExp =>

  case class VectorApply[A:Manifest](x: Exp[Vector[A]], n: Exp[Int]) extends
     Def[A]
  case class VectorLength[A:Manifest](x: Exp[Vector[A]]) extends Def[Int]
  case class VectorIsRow[A:Manifest](x: Exp[Vector[A]]) extends Def[Boolean]
}

trait ScalaGenVectorOps extends BaseGenVectorOps with ScalaGenFat {
  val IR: VectorOpsExp
  import IR._

  override def emitNode(sym: Sym[Any], rhs: Def[Any])(implicit stream:
     PrintWriter) = rhs match {
      case VectorApply(x,n) => emitValDef(sym, quote(x) + "(" + quote(n) + ")")
      case VectorLength(x)    => emitValDef(sym, quote(x) + ".length")
      case VectorIsRow(x)     => emitValDef(sym, quote(x) + ".isRow")
    }
}
```

Sequential kernels

```
case class VectorConcatenate[A:Manifest](x: Exp[Vector[A]], y: Exp[Vector[A]])
  extends DeliteOpSingleTask(reifyEffectsHere(vector_concatenate_impl(x,y)))
```

- Where is vector_concatenate_impl?
  - VectorImplOps.scala

Sequential kernels (2)

- VectorImplOps.scala

```
trait VectorImplOps { this: OptiML =>
  def vector_concatenate_impl[A:Manifest](v1: Rep[Vector[A]], v2: Rep[Vector[A]]): Rep[Vector[A]]
}

trait VectorImplOpsStandard extends VectorImplOps {
  this: OptiMLCompiler with OptiMLLift =>

def vector_concatenate_impl[A:Manifest](v1: Rep[Vector[A]], v2: Rep[Vector[A]]) = {
    if (v1.isInstanceOfL[NilVector[A]]) v2
    else if (v2.isInstanceOfL[NilVector[A]]) v1
    else {
      val out = Vector[A](v1.length+v2.length, v1.isRow)
      for (i <- 0 until v1.length){
        out(i) = v1(i)
      }
      for (i <- 0 until v2.length){
        out(i+v1.length) = v2(i)
      }
      out
    }
  }

   …
}
```

Notice the use of Rep and not Exp

Parallel kernels

- Probably the easiest kernel type to add

```
case class VectorPlusScalar[A:Manifest:Arith](in: Exp[Vector[A]],
    y: Exp[A]) extends DeliteOpMap[A,A,Vector] {

    val alloc = reifyEffects(Vector[A](in.length, in.isRow))
    val v = fresh[A]
    val func = reifyEffects(v + y)
  }
```

Parallel kernels with fusing

- Use multiloop instead of map
- We want to abstract this sooner than later (so that DeliteOpMap internally extends Multiloop)

```
abstract case class VectorTimesScalar[A:Manifest:Arith](in:
    Exp[Vector[A]], y: Exp[A]) extends DeliteOpVectorLoopMA[A]

  class VectorTimesScalarFresh[A:Manifest:Arith](in:
    Exp[Vector[A]], y: Exp[A]) extends VectorTimesScalar[A](in,y) {
    val size = in.length
    val isRow = in.isRow
    val v = fresh[Int]
    val body: Def[Vector[A]] = DeliteCollectElem[A,Vector[A]](
      alloc = reifyEffects(Vector[A](size, isRow)),
      func = reifyEffects(in(v) * y)
    )
  }
```

Vertices foreach

- Overrides vector foreach to provide constrained synchronization
- More semantic information => more structured parallelism

```
trait VerticesOpsExp extends VerticesOps with VariablesExp {
  this: OptiMLExp =>

  case class VerticesForeach[V <:Vertex:Manifest](in: Exp[Vertices[V]], v:
     Sym[V], func: Exp[Unit])
    extends DeliteOpForeachBounded[Vertex,V,Vertices] {

    val i = fresh[Int]
    val sync = reifyEffects(in(i).neighbors.toList)
  }

  def vertices_foreach[V <: Vertex:Manifest](x: Exp[Vertices[V]], block: Exp[V]
     => Exp[Unit]) = {
    val v = fresh[V]
    val func = reifyEffects(block(v))
    reflectEffect(VerticesForeach( (x), v, func))
  }
}
```

Using BLAS

- Currently has to be a DeliteOpSingleTask
- Eventually want to be able to express a node as BLAS, parallel, or sequential (variants)

```
case class MatrixMultiply[A:Manifest:Arith](x: Exp[Matrix[A]], y:
    Exp[Matrix[A]]) extends
    DeliteOpSingleTask(reifyEffectsHere(matrix_multiply_impl(x,y)) {

    val mM = manifest[MatrixImpl[A]]
  }

case m@MatrixMultiply(x,y) if (Config.useBlas) =>
      emitValDef(sym, "new " + remap(m.mM) + "(" + quote(x) +
    ".numRows," + quote(y) + ".numCols)")

    stream.println("scalaBLAS.matMult(%s.data,%s.data,%s.data,%s
    .numRows,%s.numCols,%s.numCols)".format(quote(x),quote(y),
    quote(sym),quote(x),quote(x),quote(y)))
```

scalaBLAS?

- Interface provided by Delite framework in ExternLibrary.scala
- Ad-hoc, only contains BLAS calls used by OptiML right now
- Intend to generalize this to a more robust external library call
  - DeliteOpExternalLib?

The datastruct folder

- Contains actual data structures that will be used inside generated kernels
- Entire folder is copied over to generated directory when the program starts generating
- Runtime code cache will only recompile if there are changes
- Contents
  - VectorImpl.scala, MatrixImpl.scala, …
  - Interfaces.scala

VectorImpl.scala

```
class VectorImpl[@specialized T: ClassManifest](__length: Int,
    __isRow: Boolean) extends Vector[T] {
  import VectorImpl._

  protected var _length = __length
  protected var _isRow = __isRow
  protected var _data: Array[T] = new Array[T](_length)

  def length = _length
  def isRow = _isRow
  def data = _data

  def this(__data: Array[T], __isRow: Boolean){
    this(0, __isRow)
    _data = __data
    _length = _data.length
  }

  …
```

Ordinary Scala code!
Uses vanilla Scala compiler

Interfaces.scala

```
trait Vector[@specialized(Boolean, Int, Long, Float, Double)
  T] extends ppl.delite.framework.DeliteCollection[T] {
  // methods required on real underlying data structure impl
  // we need these for:
  //   1) accessors to data fields
  //   2) setters to data fields (alternatively, methods that can
    mutate data fields)
  //   3) methods that the runtime expects (dcApply,
    dcUpdate)
  def length : Int
  def isRow : Boolean
  def apply(n: Int) : T
```

Ordinary Scala code!
Uses vanilla Scala compiler

Control structures (LanguageOps.scala)

- sum
- untilconverged
- aggregate
- gradient

- other built-ins:
  - random, min, max, distance, nearest neighbor, sample

sum

```
case class Sum[A:Manifest:Arith](start: Exp[Int], end:
    Exp[Int], mV: Sym[Int], map: Exp[A])
    extends DeliteOpMapReduce[Int,A,Vector] {

    val in = Vector.range(start, end)
    val rV = (fresh[A],fresh[A])
    val reduce = reifyEffects(rV._1 += rV._2)
}

  def optiml_sum[A:Manifest:Arith](start: Exp[Int], end:
    Exp[Int], block: Exp[Int] => Exp[A]) = {

    val mV = fresh[Int]
    val map = reifyEffects(block(mV))
    Sum(start, end, mV, map)
}
```

OptiML type classes

- Arith
  - Used instead of Numeric
  - Allows primitives and OptiML types (vector, matrix) to share the same arithmetic type class
- Cloneable

Arith

```
trait ArithInternal[Rep[X],T] {
  def +=(a: Rep[T], b: Rep[T]) : Rep[T]
  def +(a: Rep[T], b: Rep[T]) : Rep[T]
  def -(a: Rep[T], b: Rep[T]) : Rep[T]
  def *(a: Rep[T], b: Rep[T]) : Rep[T]
  def /(a: Rep[T], b: Rep[T]) : Rep[T]
  def abs(a: Rep[T]) : Rep[T]
  def exp(a: Rep[T]) : Rep[T]
}
```

- Why ArithInternal?
  - Need to agree on Rep[T]
  - May be a better way.. Find it and let me know!

Arith (2)

```
trait ArithOps extends Variables with OverloadHack {
  this: OptiML =>

  type Arith[X] = ArithInternal[Rep,X]

  implicit def arithToArithOps[T:Arith:Manifest](n: T) = new ArithOpsCls(unit(n))
  implicit def repArithToArithOps[T:Arith:Manifest](n: Rep[T]) = new ArithOpsCls(n)
  implicit def varArithToArithOps[T:Arith:Manifest](n: Var[T]) = new ArithOpsCls(readVar(n))

  implicit def chainRepArithToArithOps[A,B](a: Rep[A])
    (implicit mA: Manifest[A], aA: Arith[A], mB: Manifest[B], aB: Arith[B], c: Rep[A] => Rep[B]) =
      new ArithOpsCls(c(a))

  class ArithOpsCls[T](lhs: Rep[T])(implicit mT: Manifest[T], arith: Arith[T]){
    def +=(rhs: Rep[T]): Rep[T] = arith.+=(lhs,rhs)
    def +(rhs: Rep[T]): Rep[T] = arith.+(lhs,rhs)
    def -(rhs: Rep[T]): Rep[T] = arith.-(lhs,rhs)
    def *(rhs: Rep[T]): Rep[T] = arith.*(lhs,rhs)
    def /(rhs: Rep[T]): Rep[T] = arith./(lhs,rhs)
    def abs: Rep[T] = arith.abs(lhs)
    def exp: Rep[T] = arith.exp(lhs)
  }

  …
}
```

OptiML pattern matching optimizers

```
override def vector_apply[A:Manifest](x: Exp[Vector[A]], n:
    Exp[Int]) = x match {
    case Def(VectorObjectZeros(l)) =>
    unit(0).asInstanceOf[Exp[A]]
    case Def(VectorObjectOnes(l)) =>
    unit(1).asInstanceOf[Exp[A]]
    case Def(VectorObjectRange(s,e,d,r)) => (s +
    n*d).asInstanceOf[Exp[A]]
    case Def(VectorTrans(x)) => vector_apply(x,n)
    case Def(MatrixGetRow(x, i)) => matrix_apply(x,i,n)
    super.vector_apply(x,n)
}
```

- If a VectorObjectZeros instance is only used for apply, it never gets created!
- Key component of automatically removing allocations after we use – field reads don’t require intermediate instance

Recap

- Showed how to define:
  - Vector ops (VectorOps.scala)
    - Datastructure, sequential, parallel, BLAS
  - Vector sequential kernel implementations (VectorImplOps.scala)
  - Vector data structure (VectorImpl.scala and Interfaces.scala)
  - OptiML control structures (LanguageOps.scala)
  - An OptiML type class (Arith.scala and ArithOps.scala)
  - Vector pattern matching optimization (VectorOps.scala)

OptiML.scala

- OptiML package definitions
- remap
- specialization

OptiML package definitions

- Goals:
  - As modular and flexible as possible
  - Separate lifting of constants from their IR definitions
    - So you don’t accidentally lift something inside the compiler you didn’t intend to (like a variable)
  - Separate which ops DSL applications can use and which the compiler can use
    - Just because I want to use ListOps doesn’t mean I want my DSL applications to use Lists..

OptiML package definitions (2)

```
/**
 * These separate OptiML applications from the Exp world.
 */

// ex. object GDARunner extends OptiMLApplicationRunner with
    GDA
trait OptiMLApplicationRunner extends OptiMLApplication with
    DeliteApplication with OptiMLExp

// ex. trait GDA extends OptiMLApplication
trait OptiMLApplication extends OptiML with OptiMLLift with
    OptiMLLibrary {
  var args: Rep[Array[String]]
  def main(): Unit
}
```

OptiML package definitions (3)

```
/**
 * These are the portions of Scala imported into OptiML's scope.
 */
trait OptiMLLift extends LiftVariables with LiftEquals with LiftString with LiftBoolean with
      LiftNumeric {
  this: OptiML =>
}

trait OptiMLScalaOpsPkg extends Base
  with Equal with IfThenElse with Variables with While with Functions
  with ImplicitOps with OrderingOps with StringOps
  with BooleanOps with PrimitiveOps with MiscOps with TupleOps
  with MathOps with CastingOps with ObjectOps
  // only included because of args. TODO: investigate passing args as a vector
  with ArrayOps

trait OptiMLScalaOpsPkgExp extends OptiMLScalaOpsPkg with DSLOpsExp
  with EqualExp with IfThenElseExp with VariablesExp with WhileExp with FunctionsExp
  with ImplicitOpsExp with OrderingOpsExp with StringOpsExp with RangeOpsExp with IOOpsExp
  with ArrayOpsExp with BooleanOpsExp with PrimitiveOpsExp with MiscOpsExp with TupleOpsExp
  with ListOpsExp with SeqOpsExp with MathOpsExp with CastingOpsExp with SetOpsExp with
      ObjectOpsExp
  with ArrayBufferOpsExp
```

OptiML package definitions (4)

```
trait OptiMLScalaCodeGenPkg extends ScalaGenDSLOps
  with ScalaGenEqual with ScalaGenIfThenElse with
    ScalaGenVariables with ScalaGenWhile with
    ScalaGenFunctions
  with ScalaGenImplicitOps with ScalaGenOrderingOps with
    ScalaGenStringOps with ScalaGenRangeOps with
    ScalaGenIOOps
  with ScalaGenArrayOps with ScalaGenBooleanOps with
    ScalaGenPrimitiveOps with ScalaGenMiscOps with
    ScalaGenTupleOps
  with ScalaGenListOps with ScalaGenSeqOps with
    ScalaGenMathOps with ScalaGenCastingOps with
    ScalaGenSetOps with ScalaGenObjectOps
  with ScalaGenArrayBufferOps
  { val IR: OptiMLScalaOpsPkgExp  }
```

OptiML package definitions (5)

```
/**
 * This the trait that every OptiML application must extend.
 */
trait OptiML extends OptiMLScalaOpsPkg with LanguageOps with ApplicationOps with ArithOps with CloneableOps
  with VectorOps with MatrixOps with MLInputReaderOps with MLOutputWriterOps with VectorViewOps
  with IndexVectorOps with IndexVector2Ops with MatrixRowOps with MatrixColOps
  with StreamOps with StreamRowOps
  with GraphOps with VerticesOps with EdgeOps with VertexOps with MessageEdgeOps with MessageVertexOps
  with LabelsOps with TrainingSetOps with ImageOps with GrayscaleImageOps {

  this: OptiMLApplication =>
}

// these ops are only available to the compiler (they are restricted from application use)
trait OptiMLCompiler extends OptiML with RangeOps with IOOps with SeqOps with SetOps with ListOps {
  this: OptiMLApplication with OptiMLExp =>
}


/**
 * These are the corresponding IR nodes for OptiML.
 */
trait OptiMLExp extends OptiMLCompiler with OptiMLScalaOpsPkgExp with LanguageOpsExp with ApplicationOpsExp with
        ArithOpsExpOpt
  with VectorOpsExpOpt with MatrixOpsExpOpt with MLInputReaderOpsExp with MLOutputWriterOpsExp with
        VectorViewOpsExp
  with IndexVectorOpsExp with IndexVector2OpsExp with MatrixRowOpsExpOpt with MatrixColOpsExpOpt
  …
}
```

OptiML package definitions summary

- OptiML: collection of all Rep[T] Ops available to OptiML applications
- OptiMLCompiler: collection of all Rep[T] ops available to OptiML compiler
- OptiMLExp: collection of all Exp[T] Ops in OptiML
- OptiMLLift: collection of traits that perform implicit lifting of constants (available to applications and embedded kernels)
- OptiML{Scala,Cuda}OpsPkg: collection of traits that implement code generators for ops

remap

- A function that defines the mapping between a Rep[T], and its actual type for a particular generator (Scala, Cuda)
- Defined inside the code gen object
- Ex.

```
override def remap[A](m: Manifest[A]) : String = {
    m.toString match {
      case "ppl.dsl.optiml.datastruct.scala.Matrix[Int]" =>
   "Matrix<int>"
      case "ppl.dsl.optiml.datastruct.scala.Matrix[Long]" =>
   "Matrix<long>“
    …
}
```

specialization

- When OptiML copies datastructures to the generated folder, it also specializes them (and takes care of the type mapping inside ‘remap’)

- Similar to @specialized in Scala, but more explicit (easier to verify/debug)

- Simple string manipulation

```
def specmap(line: String, t: String) : String = {
    var res = line.replaceAll("object ", "object " + t)
    res = res.replaceAll("import ", "import " + t)
    res = res.replaceAll("@specialized T: ClassManifest", t)
    res = res.replaceAll("T:Manifest", t)
    res = res.replaceAll("\\bT\\b", t)
    parmap(res)
  }
```

OptiML unit tests

- DeliteSuite
  - A ScalaTest suite
  - Stages (generates) and then invokes runtime (executes) for a set of test “modules”
  - Allows using a constrained form of ScalaTest in our metaprogramming environment
    - Only Boolean assertions

OptiML unit tests example

```
class LinearAlgebraSuite extends DeliteSuite {
  def testSimpleVector() { compileAndTest(SimpleVectorArithmeticRunner)
  }
}

object SimpleVectorArithmeticRunner extends DeliteTestRunner with
      OptiMLApplicationRunner with SimpleVectorArithmetic
trait SimpleVectorArithmetic extends DeliteTestModule with LinearAlgebraTestsCommon
      {
  def main() = {
    implicit val collector = ArrayBuffer[Boolean]()
    val rowA = Vector(11., 22., 33.)
    val rowB = Vector(-5.3, -17.2, -131.)
    val rowD = Vector(-1.1, -6.2)
    val colC = Vector(7., 3.2, 13.3).mt

    // A*B piecewise
    val ansVec = rowA*rowB
    collect(check(ansVec, Vector(-58.3, -378.4, -4323.)))

    mkReport
  }
}
```

Application datastructures

- What happens when you want to support application datastructures in this world?
- How do field reads and methods get lifted?
- Good question
  - For now, we have a simple approach for handling user-defined struct-like datastructures
  - Like everything else, (but even a little moreso), this is very, very alpha

Application datastructures (2)

- Tell users to put all of their real struct-like datastructures in a single folder
- Must follow a format like:

```
class Foo(
    val x: Int,
    val y: Bar,
    val z: Double,
    …
)
```

- We have a python script, lift_user_class.py, that will process this directory, and output a single ApplicationOps.scala file
- This file should be in a known location and statically mixed in to your language

Putting it all together: SPADE

- Application code:

Datastructure op, constructs StreamImpl

```
val distances = Stream[Double](data.numRows,
data.numRows){ (i,j) => dist(data(i),data(j)) }
val densities = Vector[Int](data.numRows, true)

for (row <- distances.rows) {
    if(densities(row.index) == 0) {
        val neighbors = row find { _ < apprxWidth }
        densities(neighbors) = row count { _ < kernelWidth }
    }
}
densities
```

Parallel op,
StreamForeach

Multiloop op, fused
Pattern matching ensures entire
allocation can be removed