



OptiML: An Implicitly Parallel Domain-Specific Language for ML

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

Stanford University
Pervasive Parallelism Laboratory (PPL)

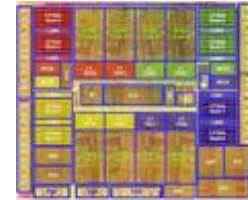
Tiark Rumpf, Martin Odersky
Ecole Polytechnique Federale de Lausanne (EPFL),
Programming Methods Laboratory

Background

- We are researchers in programming languages, parallel programming, and computer architecture
- Working with machine learning and bioinformatics groups at Stanford and elsewhere
- Would love to work with you and get your feedback, suggestions, and criticism

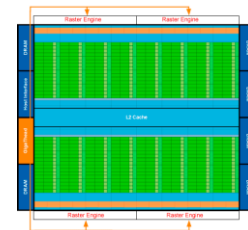
Heterogeneous Parallel Programming

Pthreads
OpenMP



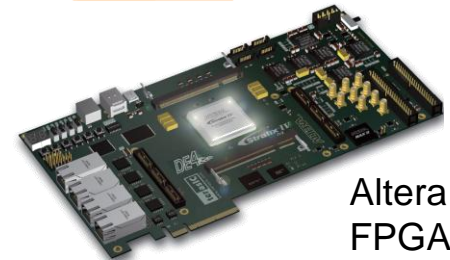
Sun
T2

CUDA
OpenCL



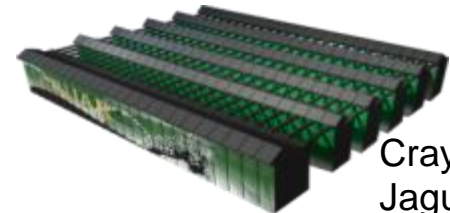
Nvidia
Fermi

Verilog
VHDL



Altera
FPGA

MPI



Cray
Jaguar

Programmability Chasm

Applications

Scientific
Engineering

Virtual
Worlds

Personal
Robotics

Data
informatics



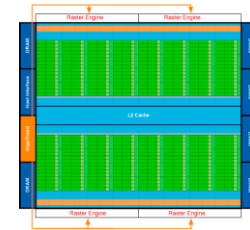
Too many different programming models

Pthreads
OpenMP



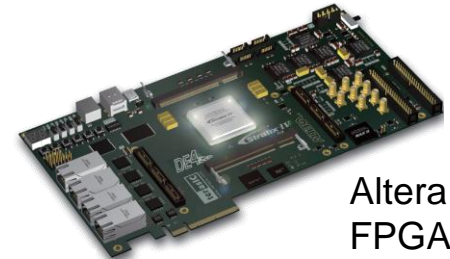
Sun
T2

CUDA
OpenCL



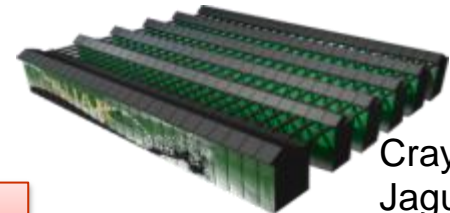
Nvidia
Fermi

Verilog
VHDL



Altera
FPGA

MPI



Cray
Jaguar

**IS IT POSSIBLE TO WRITE
ONE PROGRAM**

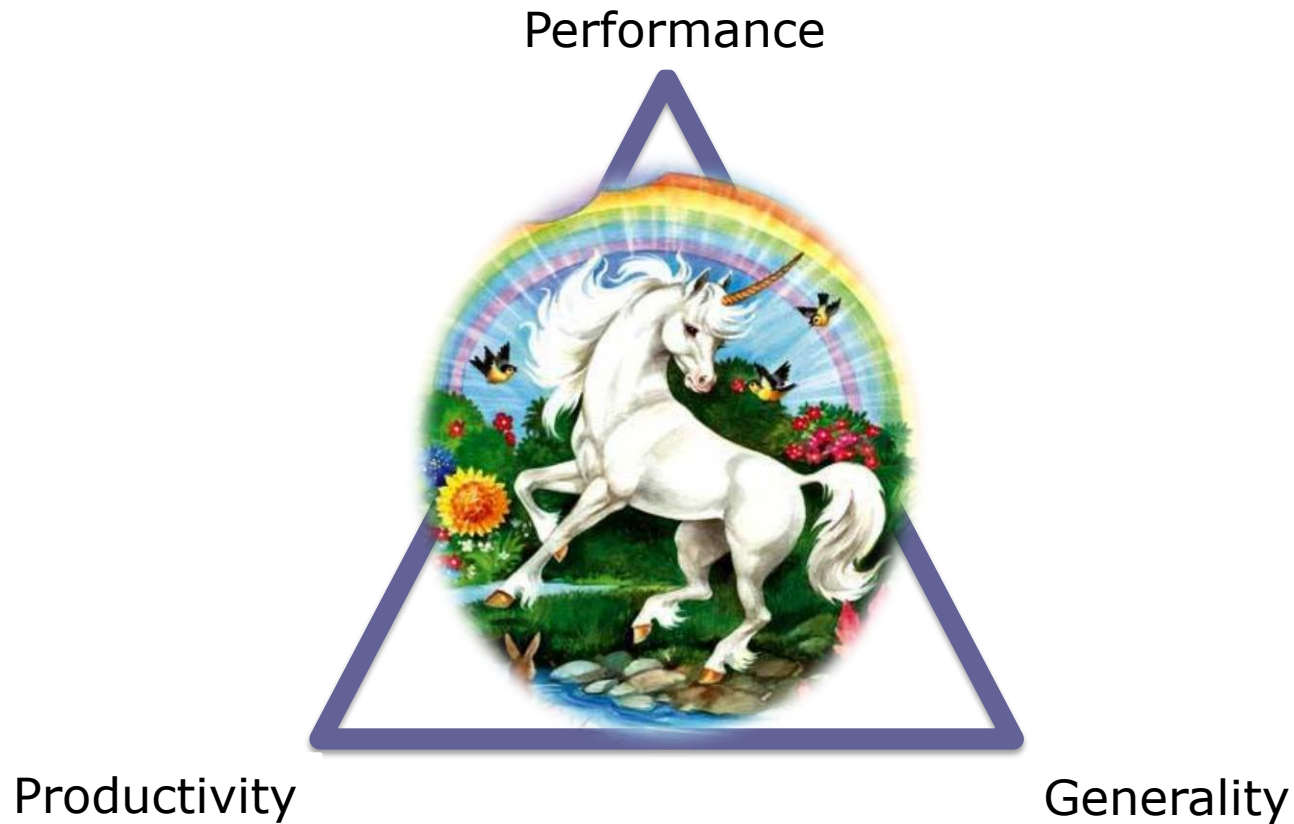
AND

RUN IT ON ALL THESE TARGETS?

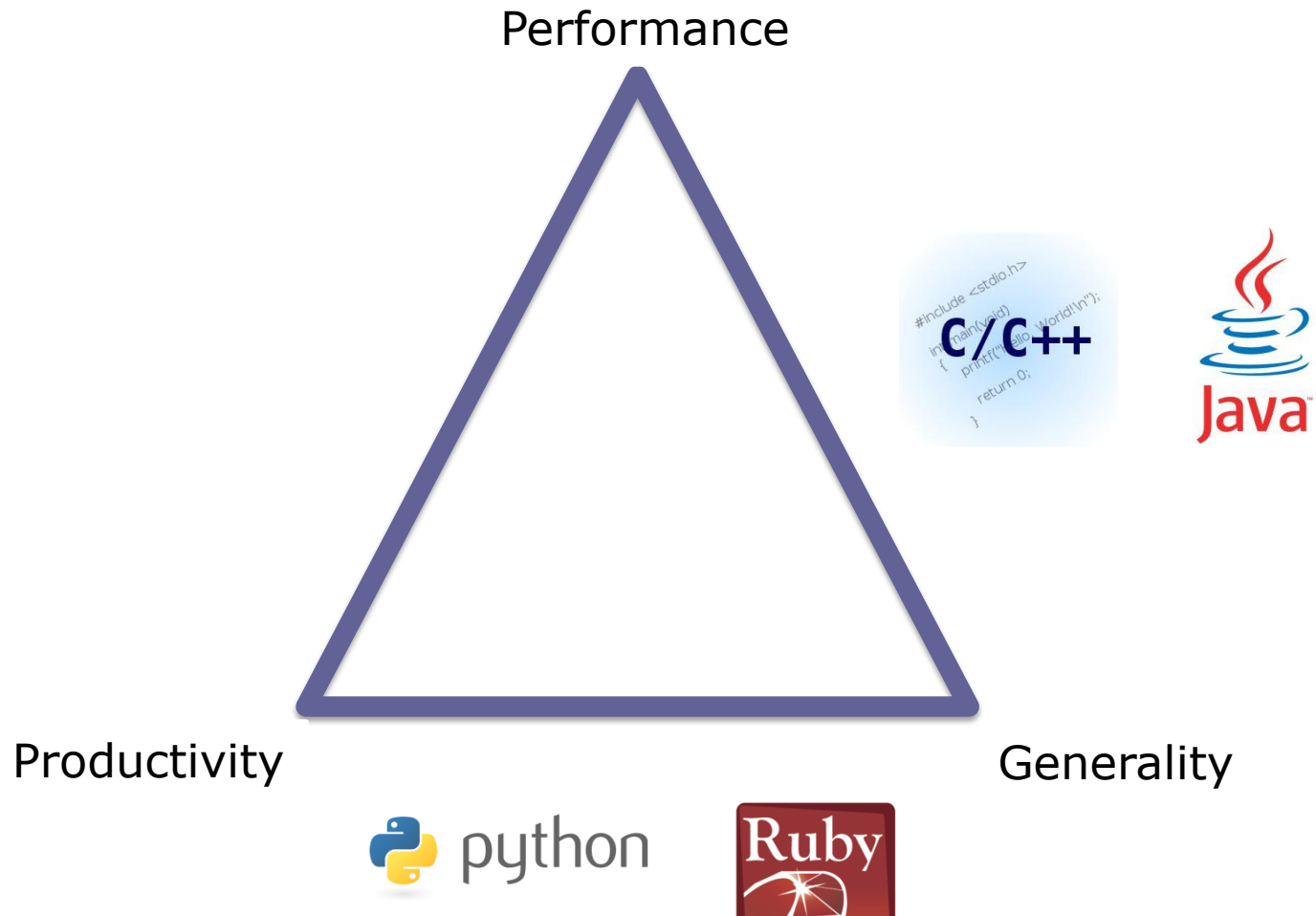
HYPOTHESIS: YES, BUT NEED

DOMAIN-SPECIFIC LANGUAGES

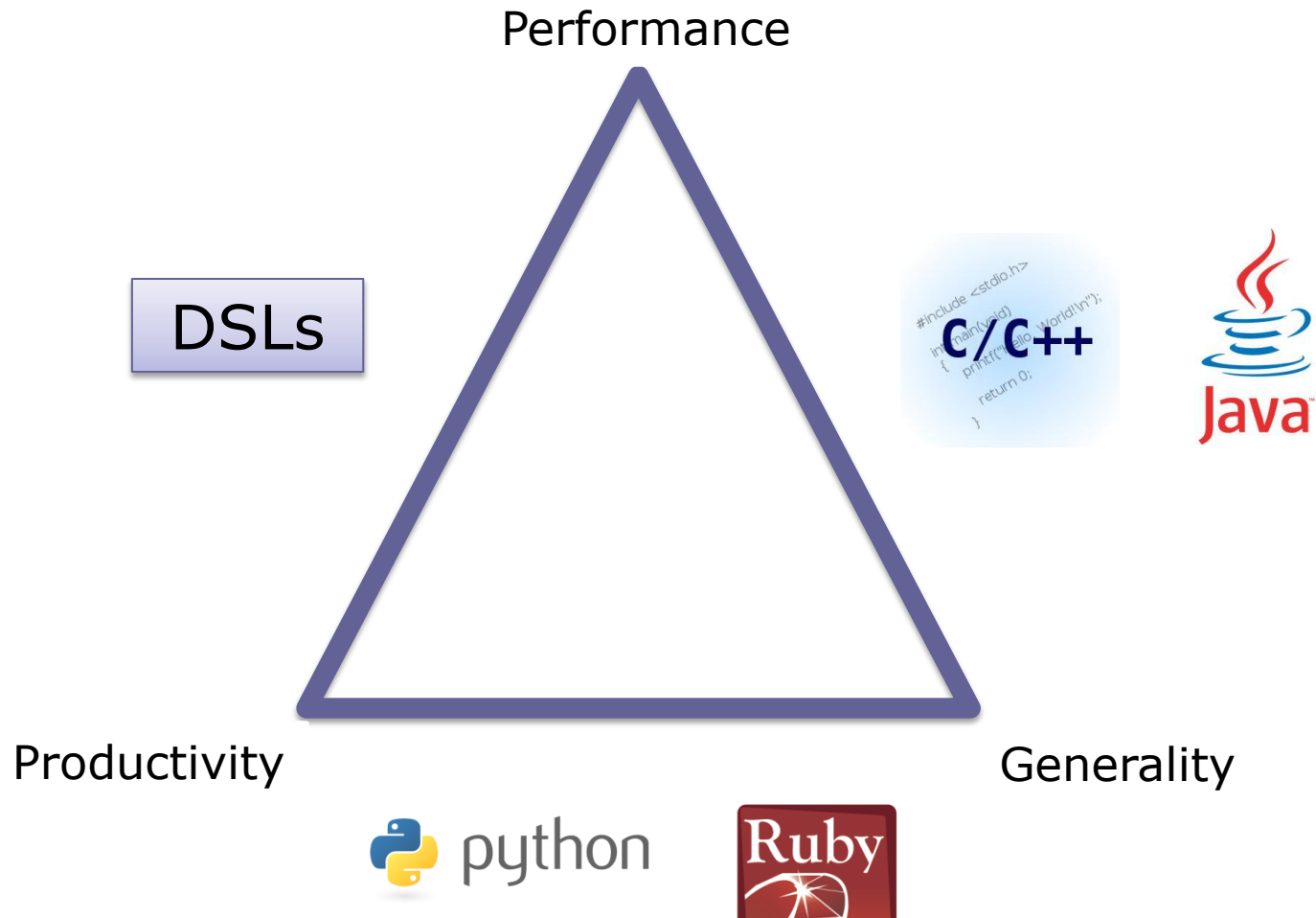
The Ideal Parallel Programming Language



Successful Languages



Successful Languages



OptiML: A DSL For ML

- **Productive**
 - Operate at a higher level of abstraction
 - Focus on algorithmic description, get parallel performance
- **Portable**
 - Single source => Multiple heterogeneous targets
 - Not possible with today's MATLAB support
- **High Performance**
 - Builds and optimizes an intermediate representation (IR) of programs
 - Generates efficient code specialized to each target

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]`, `Graph[V,E]`
 - Independent from the underlying implementation
 - Specialized data types: `Stream`, `TrainingSet`, `TestSet`, `IndexVector`, `Image`, `Video` ..
 - Encode semantic information & structured, synchronized communication
- **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 example

```
untilconverged(mu, tol){ mu =>
  // calculate distances to current centroids
  val c = (0::m){i =>
    val allDistances = mu mapRows { centroid =>
      // distance from sample x(i) to centroid
      ((x(i)-centroid)*(x(i)-centroid)).sum
    }
    // choose the closest centroid
    allDistances.minIndex
  }
  // update cluster centroid to the
  // mean of points assigned to it
  val newMu = (0::k,*) { i =>
    val (weightedpoints, points) = sum(0,m) { j =>
      if (c(i) == j){
        (x(i),1)
      }
    }
    if (points == 0) Vector.zeros(n)
    else weightedpoints / points
  }
  newMu
}
```

normal
matrix/vector
arithmetic syntax

Multiple granularities of
parallelism

control structure can only
access indices i and j
(disjoint)

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

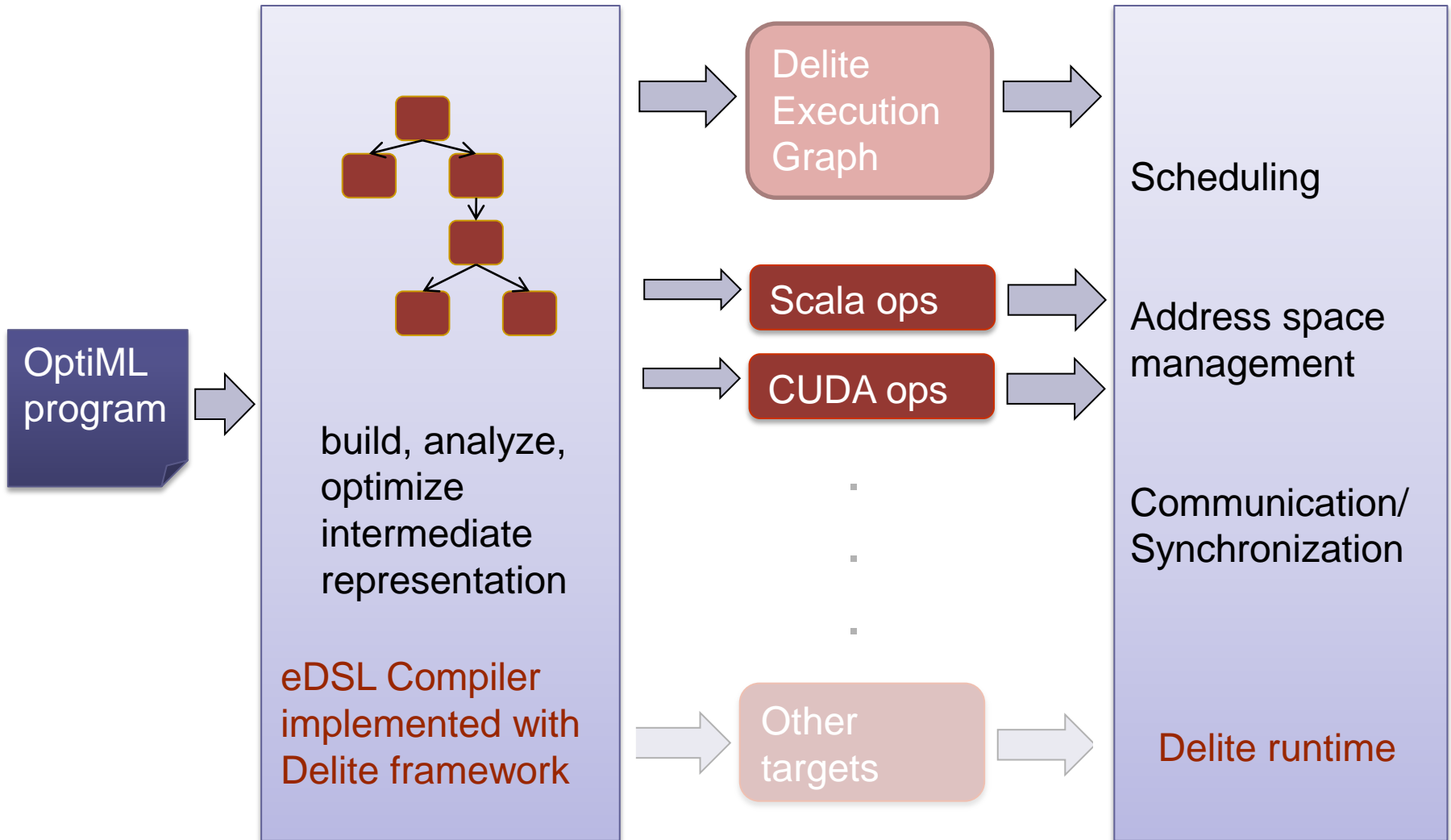
- Dynamically typed
- Applications must explicitly choose between vectorization or parallelization
- Explicit GPU data management
- Widely used, numerous libraries and toolboxes

MATLAB parallelism

- `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 Implementation



Optimizations

- Common subexpression elimination (CSE), Dead code elimination (DCE), Code motion
- 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

Coarse-grained: optimizations happen on vectors and matrices

OptiML Linear Algebra Rewrite Example

- A straightforward translation of the Gaussian Discriminant Analysis (GDA) algorithm from the mathematical description produces the following code:

```
val sigma = sum(0,m) { i =>  
  if (x.labels(i) == false) {  
    ((x(i) - mu0).t) ** (x(i) - mu0)  
  }  
  else  
    ((x(i) - mu1).t) ** (x(i) - mu1)  
}  
}
```

- A much more efficient implementation recognizes that

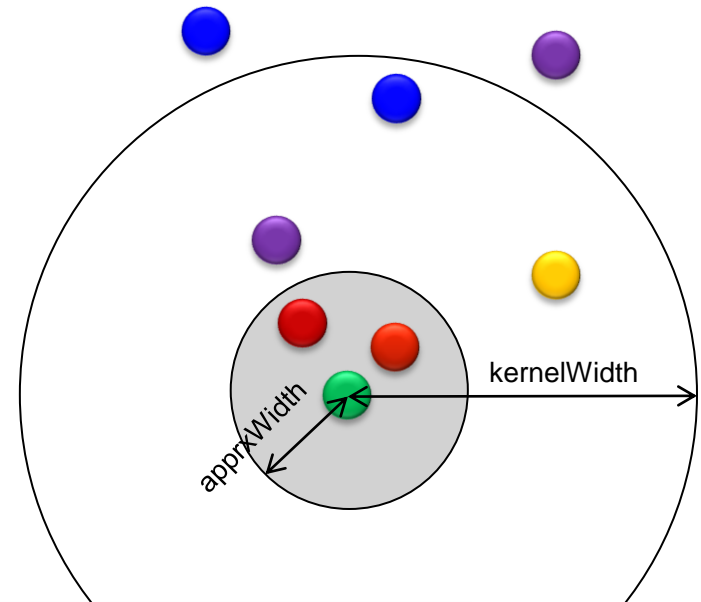
$$\sum_{i=0}^n \vec{x}_i * \vec{y}_i \rightarrow \sum_{i=0}^n X(:, i) * Y(i, :) = X * Y$$

- Transformed code was **20.4x** faster with 1 thread and **48.3x** faster with 8 threads.

Putting it all together: SPADE

Downsample:

L1 distances
between all 10^6
events in 13D
space... reduce to
50,000 events



```
val distances = Stream[Double](data.numRows, data.numRows){  
  (i,j) => dist(data(i),data(j))  
}  
  
for (row <- distances.rows) {  
  if(densities(row.index) == 0) {  
    val neighbors = row find { _ < aprxWidth }  
    densities(neighbors) = row count { _ < kernelWidth }  
  }  
}
```

SPADE transformations

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

for (row <- distances.rows) {
  row.init // expensive! part of the stream foreach operation
  if(densities(row.index) == 0) {
    val neighbors = row find { _ < apprxWidth }
    densities(neighbors) = row count { _ < kernelWidth }
  }
}
```

row is 235,000 elements
in one typical dataset –
fusing is a big win!

SPADE generated code

```
// FOR EACH ELEMENT IN ROW
while (x155 < x61) {
  val x168 = x155 * x64
  var x180 = 0

  // INITIALIZE STREAM VALUE (dist(i,j))
  while (x180 < x64) {
    val x248 = x164 + x180
    // ...
  }

  // VECTOR FIND
  if (x245) x201.insert(x201.length, x155)

  // VECTOR COUNT
  if (x246) {
    val x207 = x208 + 1
    x208 = x207
  }
  x155 += 1
}
```

From a ~5 line
algorithm
description in
OptiML

...to an efficient,
fused, imperative
version that closely
resembles a hand-
optimized C++
baseline!

Performance Results

■ Machine

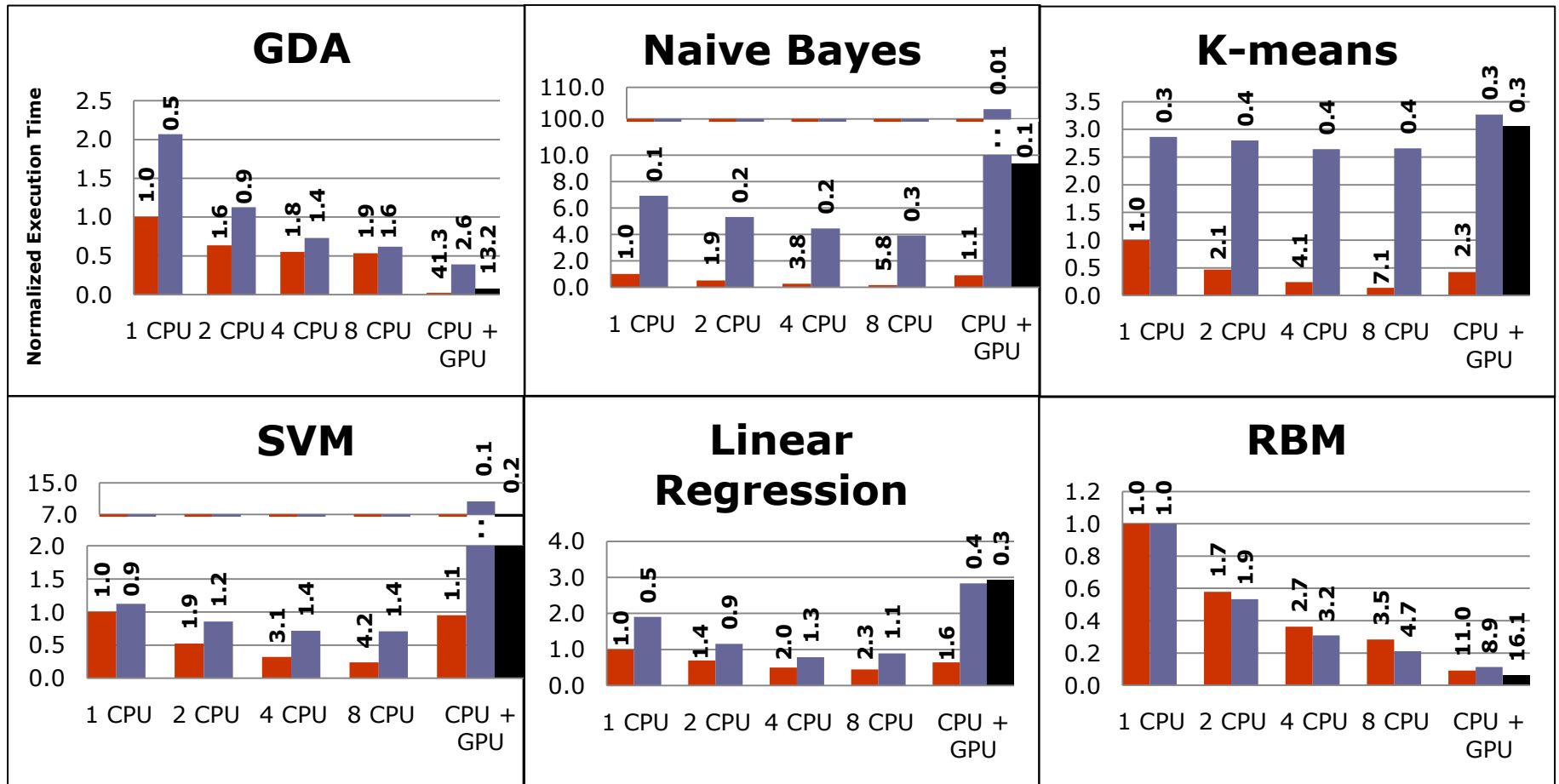
- Two quad-core Nehalem 2.67 GHz processors
- NVidia Tesla C2050 GPU

■ Application Versions

- OptiML + Delite
- MATLAB
 - version 1: multi-core (parallelization using "parfor" construct and BLAS)
 - version 2: MATLAB GPU support
 - version 3: Accelereyes Jacket GPU support
- C++
 - Optimized reference baselines for larger applications

Experiments on ML kernels

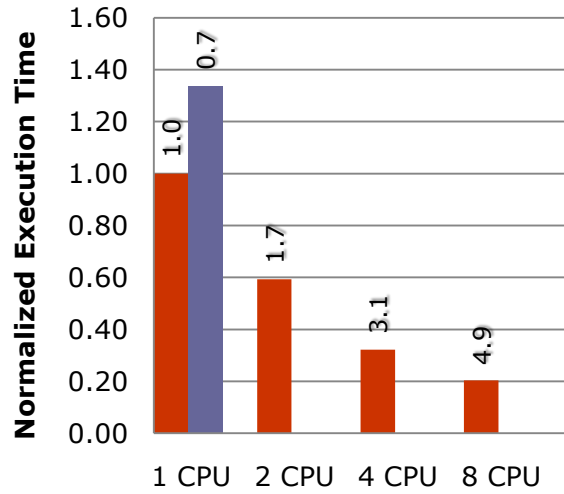
■ OptiML ■ Parallelized MATLAB ■ MATLAB + Jacket



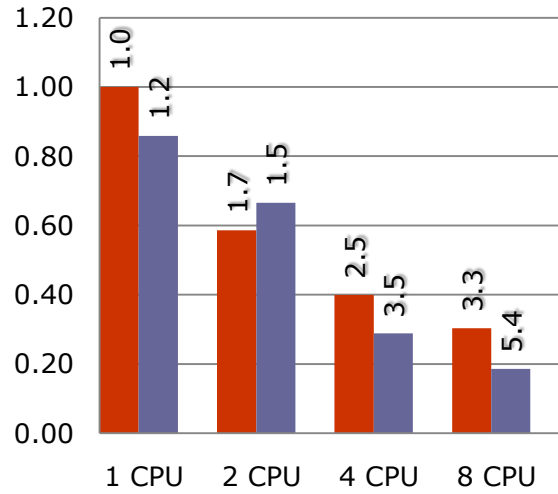
Experiments on larger apps

■ OptiML ■ C++

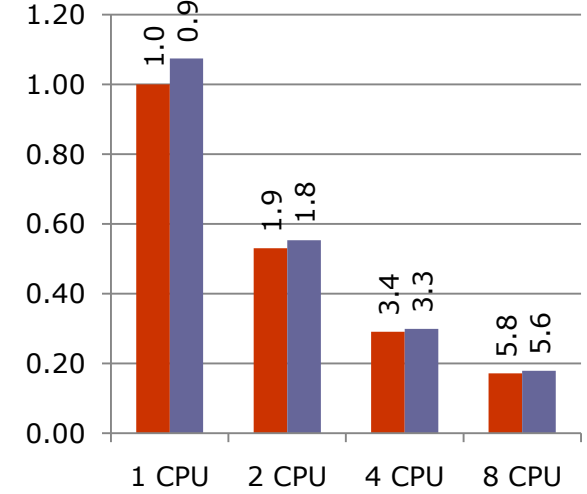
TM



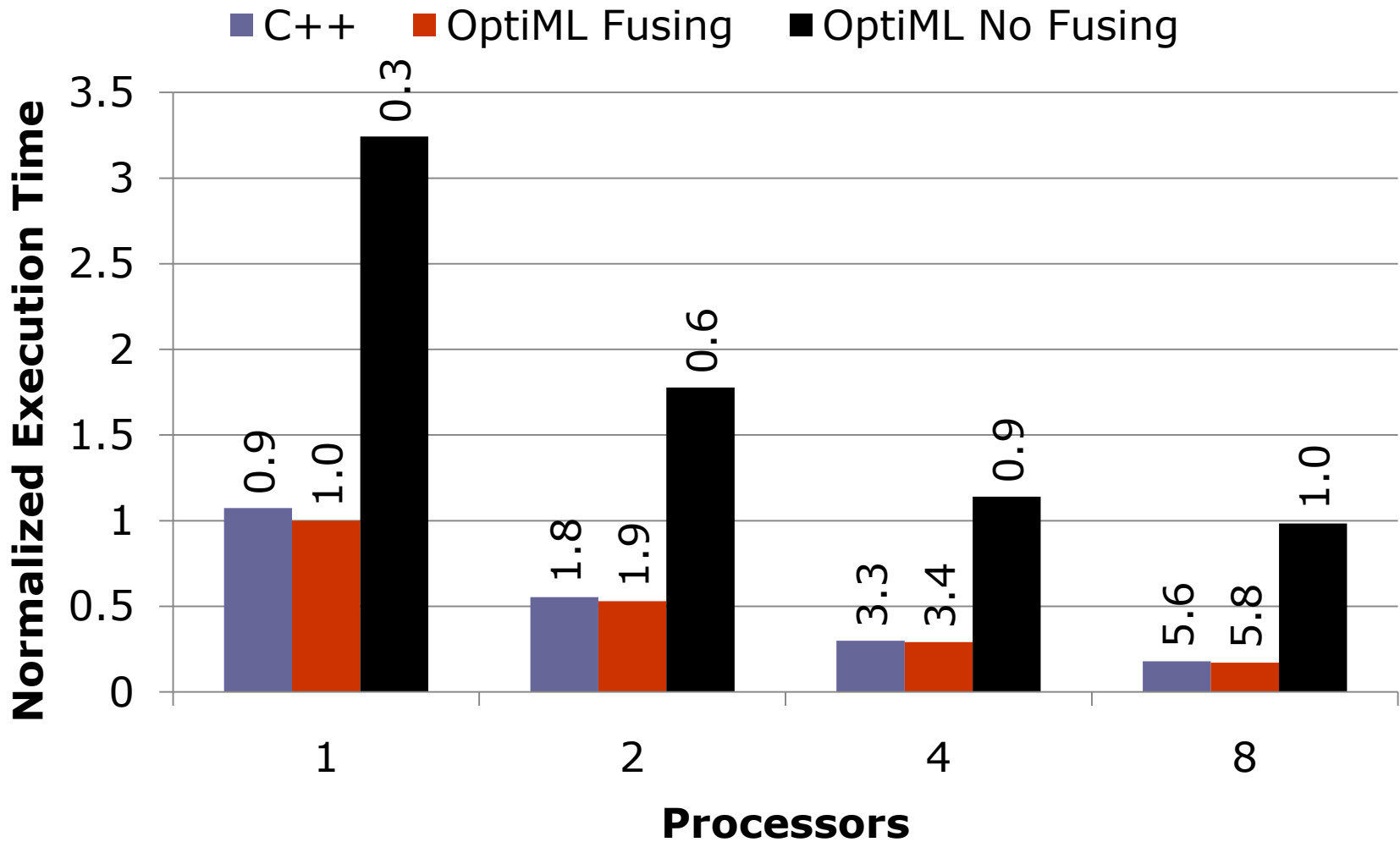
LBP



SPADE



Impact of Op Fusion



Summary

- DSLs are a promising parallel programming platform
 - Capable of achieving portability, productivity, and high performance
- OptiML is a proof-of-concept DSL for ML embedded in Scala, using the Lightweight Modular Staging (LMS) framework and Delite
- OptiML translates simple, declarative machine learning operations to optimized code for multiple platforms
- Outperforms MATLAB and C++ on a set of well-known machine learning applications

Thank you!

- For the brave, find us on Github:
 - <https://github.com/stanford-ppl/Delite>
 - (very alpha)
- Comments and criticism very welcome
- Questions?

backup

OptiML: Approach

- Encourage a functional, parallelizable style through restricted semantics
 - Fine-grained, composable map-reduce operators
- **OptiML does not have to be conservative**
- **Guarantees major properties (e.g. parallelizable) by construction**
- Automatically synchronize parallel iteration over domain-specific data structures
 - Exploit structured communication patterns (nodes in a graph may only access neighbors, etc.)
- Defer as many implementation-specific details to compiler and runtime as possible

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

OptiML code

```
% 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
```

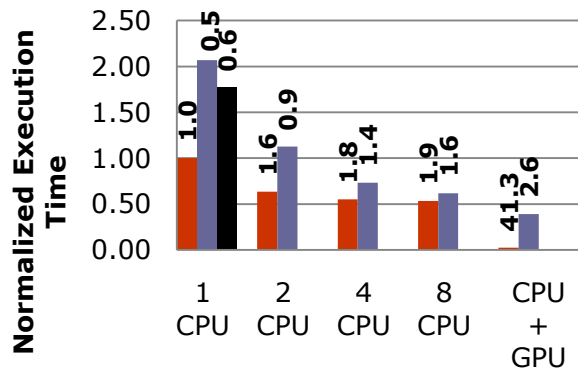
```
end
```

(parallel) MATLAB code

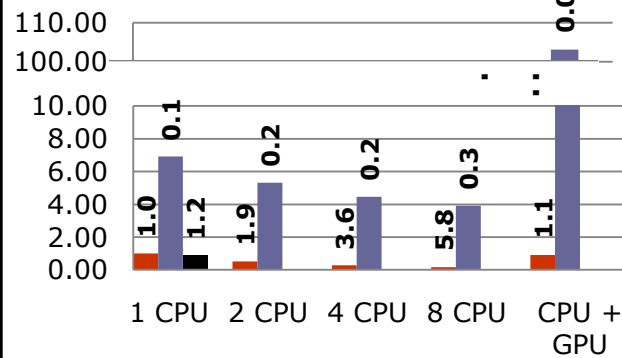
Experiments on ML kernels (C++)

■ OptiML ■ Parallelized MATLAB ■ C++

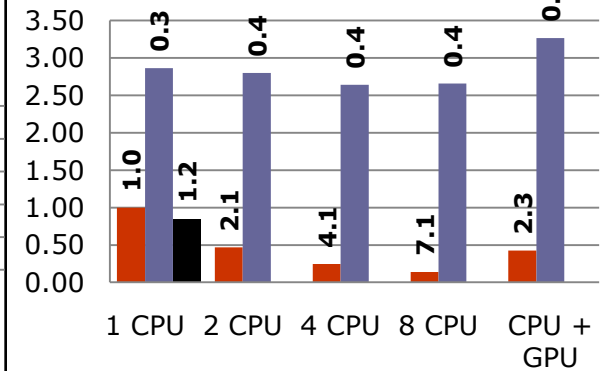
GDA



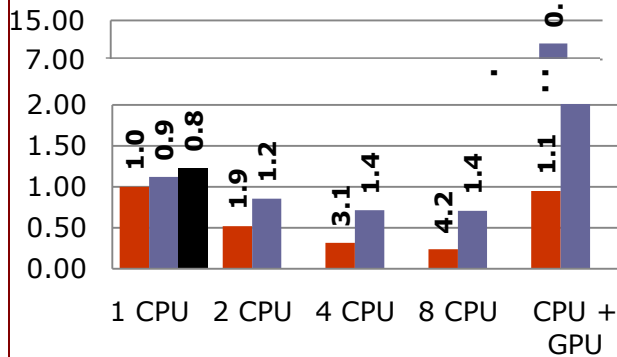
Naive Bayes



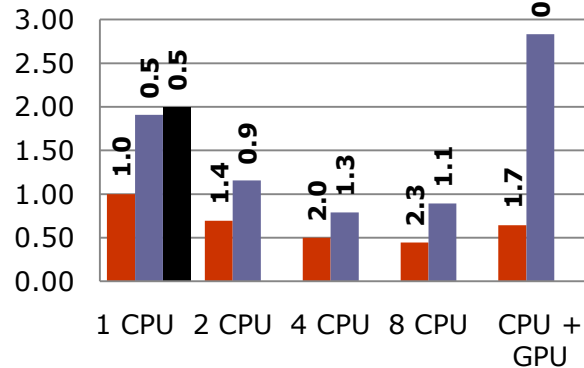
K-means



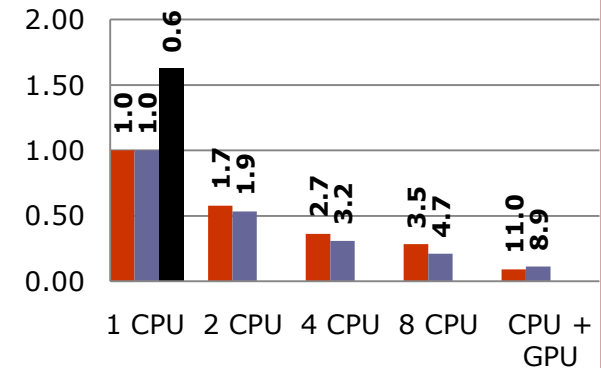
SVM



Linear Regression



RBM



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

Dynamic optimizations

K-means Best Effort

SVM Relaxed Dependencies

