CS 744: WELD

Shivaram Venkataraman

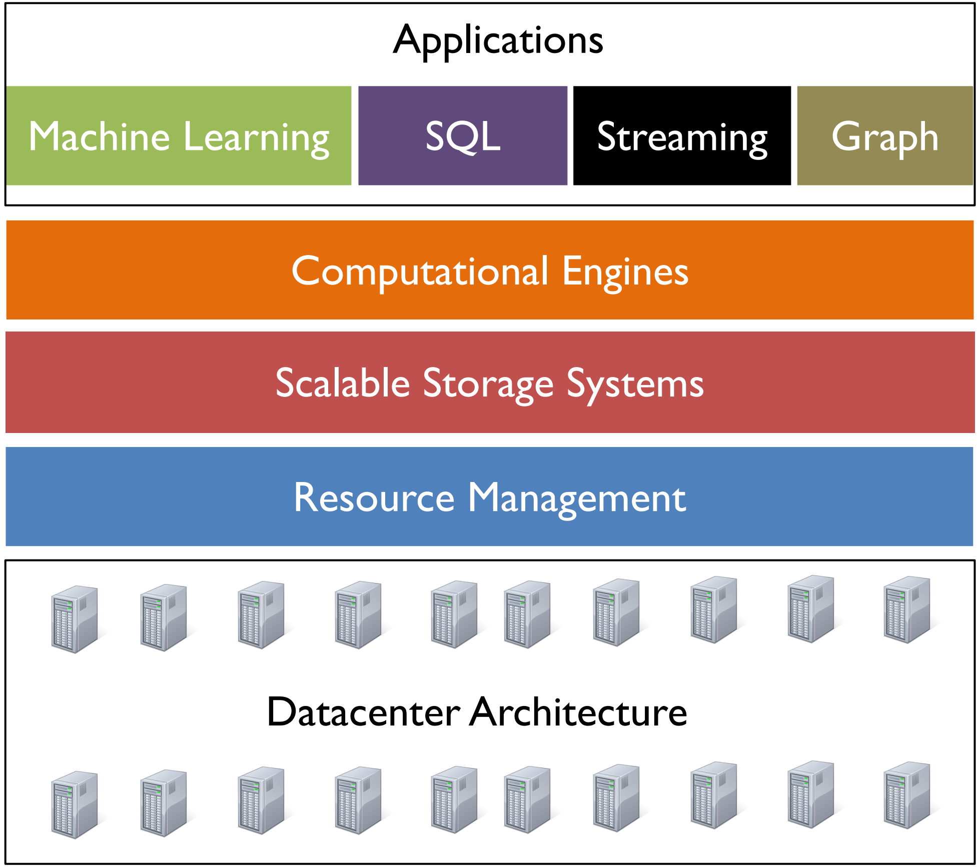
Fall 2019

ADMINISTRIVIA

Course Project: Check in meetings Thu, Mon

Preparation for the meeting

- what have you done so far
- a timeline for things you want to do next
- what are some specific things we can help you with



Applications

Machine Learning

SQL

Streaming

Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture

SETTING

Multi-core machines

Multiple functions and libraries

Data movement vs. compute

Alternate approaches?

```
// From Black Scholes
// all inputs are vectors
d1 = price * strike
d1 = np.log2(d1) + strike
```

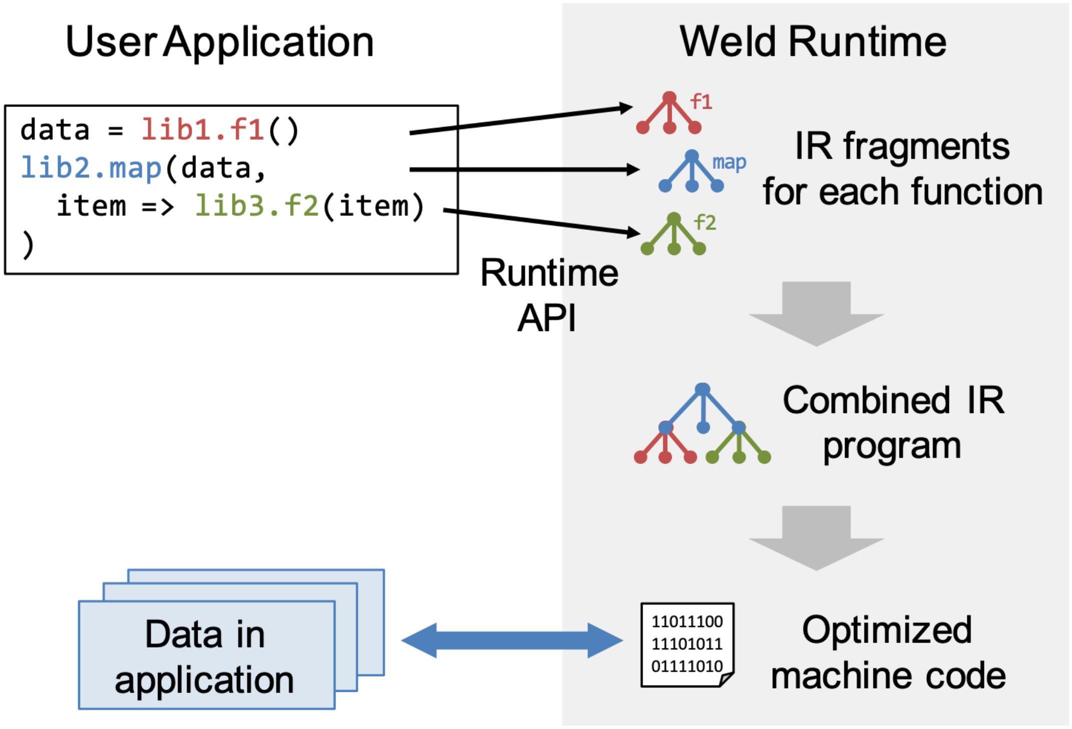
GOALS

Work with independently written libraries

Enable the most impactful cross-library optimizations

Integrate incrementally into existing systems

SYSTEM OVERVIEW



User Application

```
data = lib1.f1()
lib2.map(data,
  item => lib3.f2(item)
)
```

Runtime API

Weld Runtime

IR fragments for each function

Combined IR program

Optimized machine code

Data in application

WELD IR

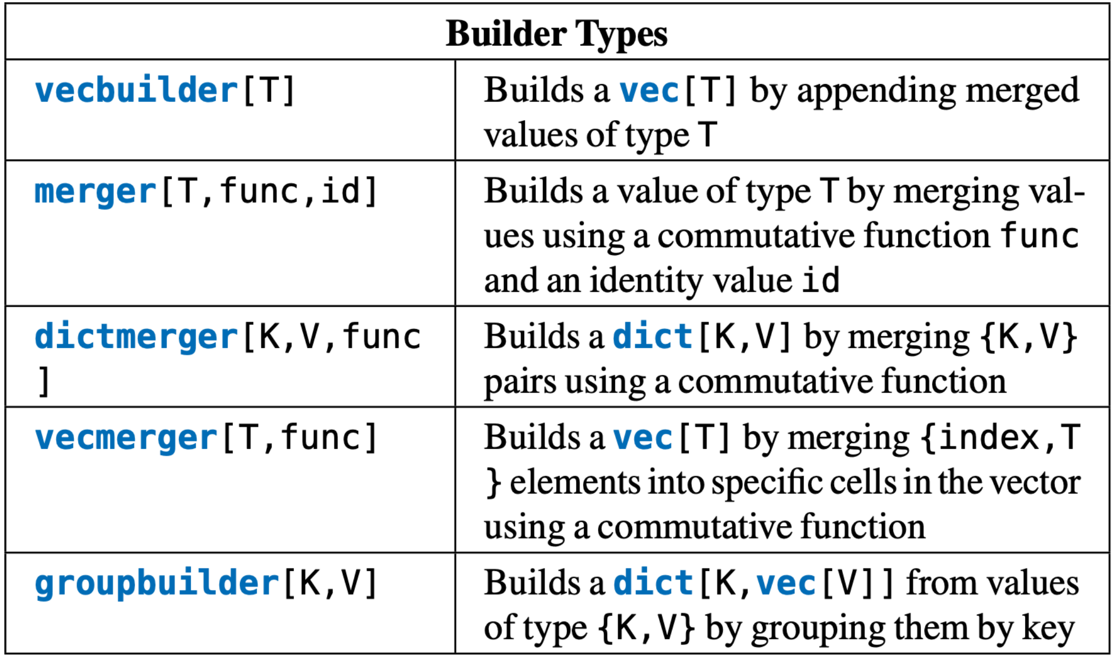
Data types

- Scalars, structs, vectors, dictionaries

Parallel loops and builders

- merge(builder, value)
- for(vector, builders, func)
- result(builder)

BUILDER TYPES



Builder Types	
`vecbuilder[T]`	Builds a `vec[T]` by appending merged values of type `T`
`merger[T,func,id]`	Builds a value of type `T` by merging values using a commutative function `func` and an identity value `id`
`dictmerger[K,V,func]`	Builds a `dict[K,V]` by merging `{K,V}` pairs using a commutative function
`vecmerger[T,func]`	Builds a `vec[T]` by merging `{index,T}` elements into specific cells in the vector using a commutative function
`groupbuilder[K,V]`	Builds a `dict[K,vec[V]]` from values of type `{K,V}` by grouping them by key

EXAMPLES OF BUILDERS

```
b1 := vecbuilder[int];
b2 := merge(b1, 5);
b3 := merge(b2, 6);
result(b3)
```

```
b1 := vecbuilder[int];
b2 := for([1,2,3], b1, (b, x) => merge(b, x+1));
result(b2)
```

MULTIPLE BUILDER

```
data := [1,2,3];
r1 := map(data, x => x+1);
r2 := reduce(data, 0, (x, y) => x+y);
```

```
data := [1,2,3];
result(
    for(data, {vecbuilder[int], merger[+]},
        (bs, x) =>
            {merge(bs.0, x+1), merge(bs.1, x)}
))
```

RUNTIME API

API to express IR fragments in libraries

Capture dependencies across functions/libraries.

Lazy Evaluation

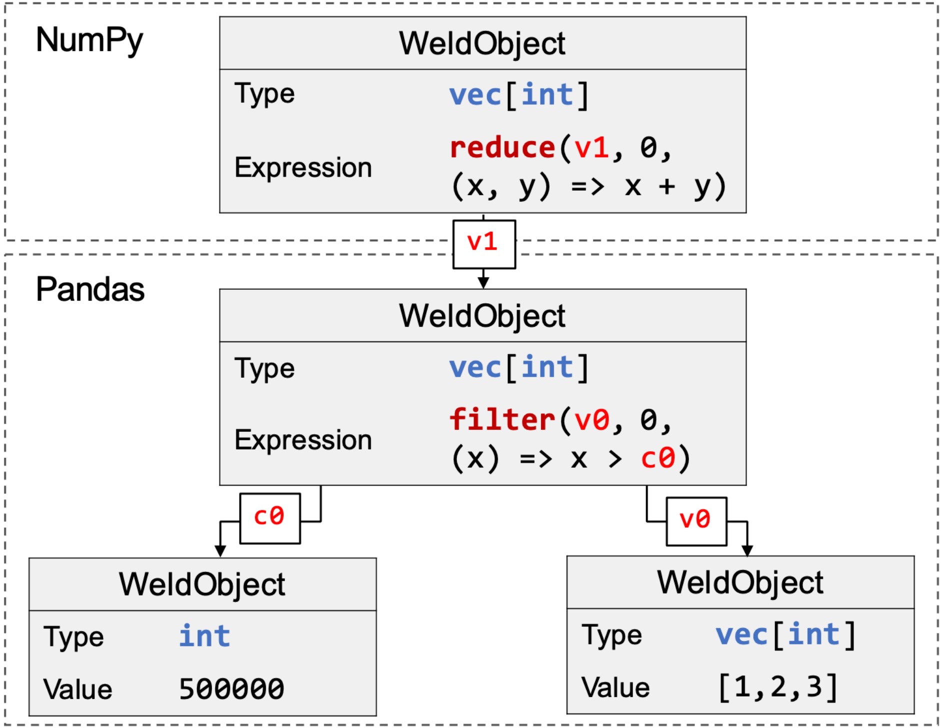
```
def square(self, arg):
    # Programatically construct an IR expression.
    expr = weld.Multiply(arg, arg)
    return NewWeldObject([arg], expr)
```

RUNTIME API

```
def large_cities_population(data):
  # data is a Pandas DataFrame object.
  filtered = data[data["population"] > 500000]
  sum = numpy.sum(filtered)
  print sum

# Dataframe col > f, Input Weld expr: v0: vec[int], c0: int
filter(v0, x => x > c0)

# Numpy.sum Input Weld expr: v0: vec[int]
reduce(v0, 0, (x, y) => x+y)
```

RUNTIME API

```
reduce(
    filter(v0,
           (x) => x>500000),
    0,
    (x,y) => x+y)



result(
    for(v0, merger[+,0],
      (b, x) =>
         if (x > 500000)
            merge(b, x)
         else
            b
))
```



OPTIMIZATIONS

Loop Fusion

Fuse adjacent loops when output of one loop is input of other

Fuse multiple passes over the same vector

Loop Tiling

Break nested loops into blocks

OPTIMIZATIONS

Vectorization

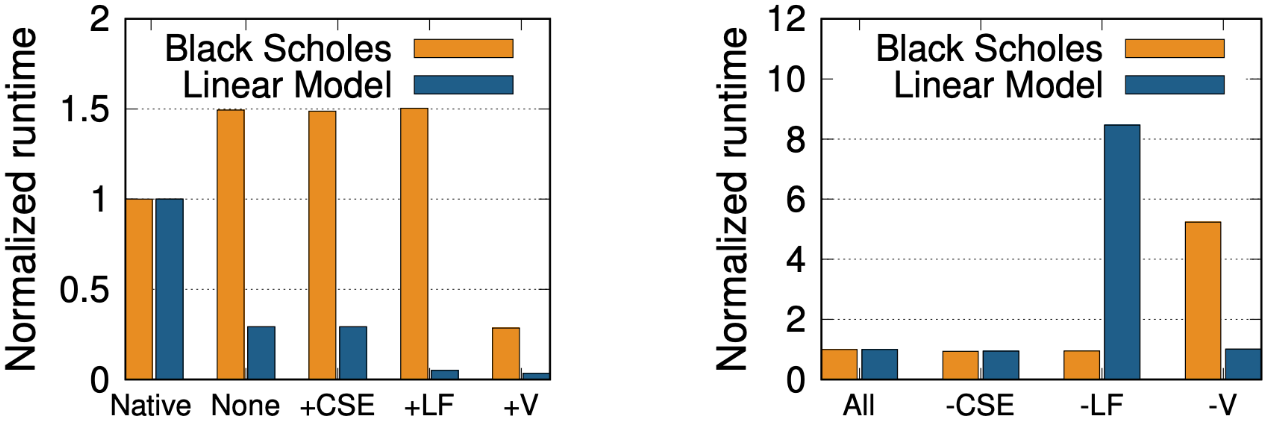
Transform loops to use vector instructions

Common subexpression elimination

Transforms to not run the same computation multiple times

DISCUSSION

https://forms.gle/DxHfcmuS2juK1tuE7



(a) Adding Optimizations

(b) Removing Optimizations

What are some possible limitations of Weld as described in the paper?

What does the Weld paper tell us about the using scale-up vs. scale-out?

NEXT STEPS

Next class: PyWren

Project check-in meetings