Weld: A Common Runtime for Data Analytics

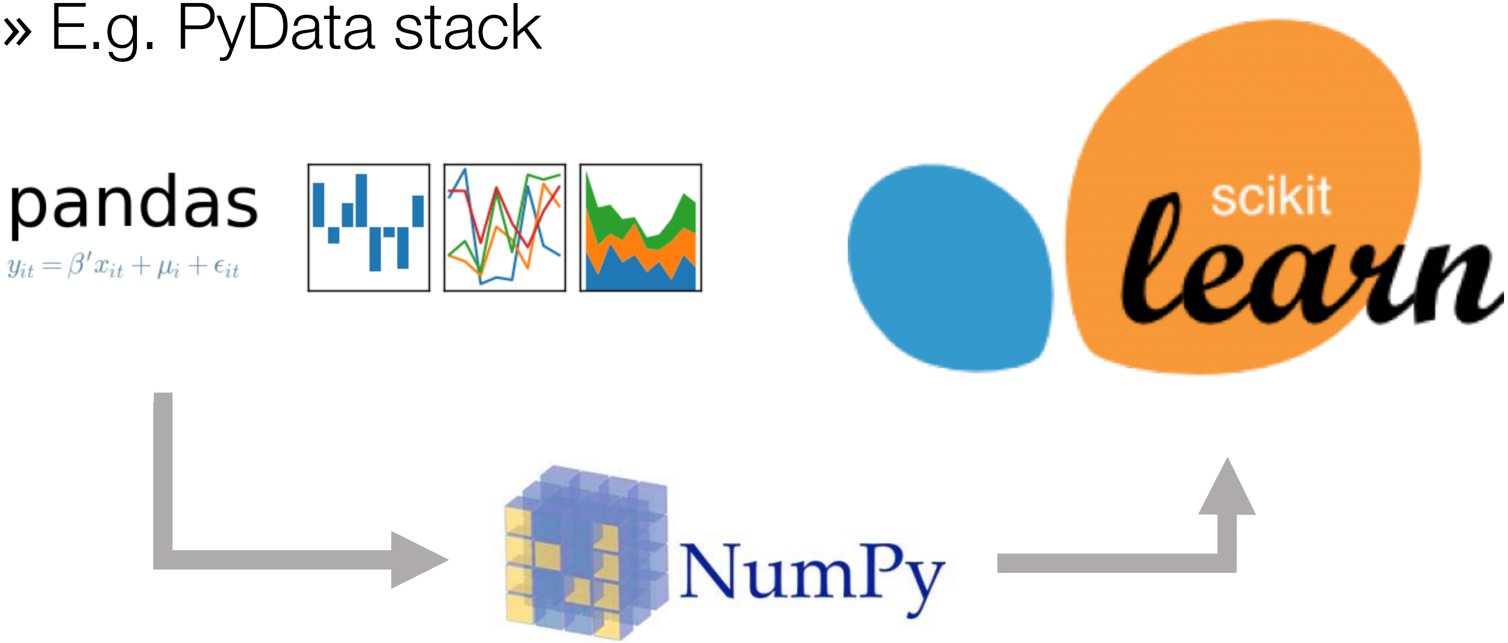
Shoumik Palkar, James Thomas, **Deepak Narayanan,** Anil Shanbhag*, Rahul Palamuttam, Holger Pirk*, Malte Schwarzkopf*, Saman Amarasinghe*, Sam Madden*, Matei Zaharia

Stanford InfoLab, *MIT CSAIL

Motivation

Modern data apps combine many disjoint processing libraries & functions

- » SQL, statistics, machine learning, …
- » E.g. PyData stack



Motivation

Modern data apps combine many disjoint processing libraries & functions

» SQL, statistics, machine learning, …

» E.g. PyData stack

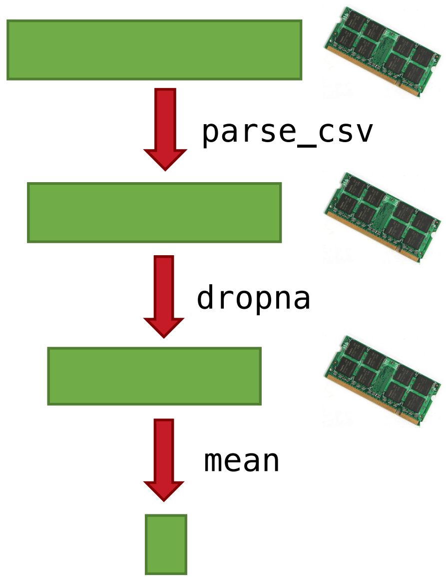
+ Great results leveraging work of 1000s of authors

– No optimization across these functions

How Bad is This Problem?

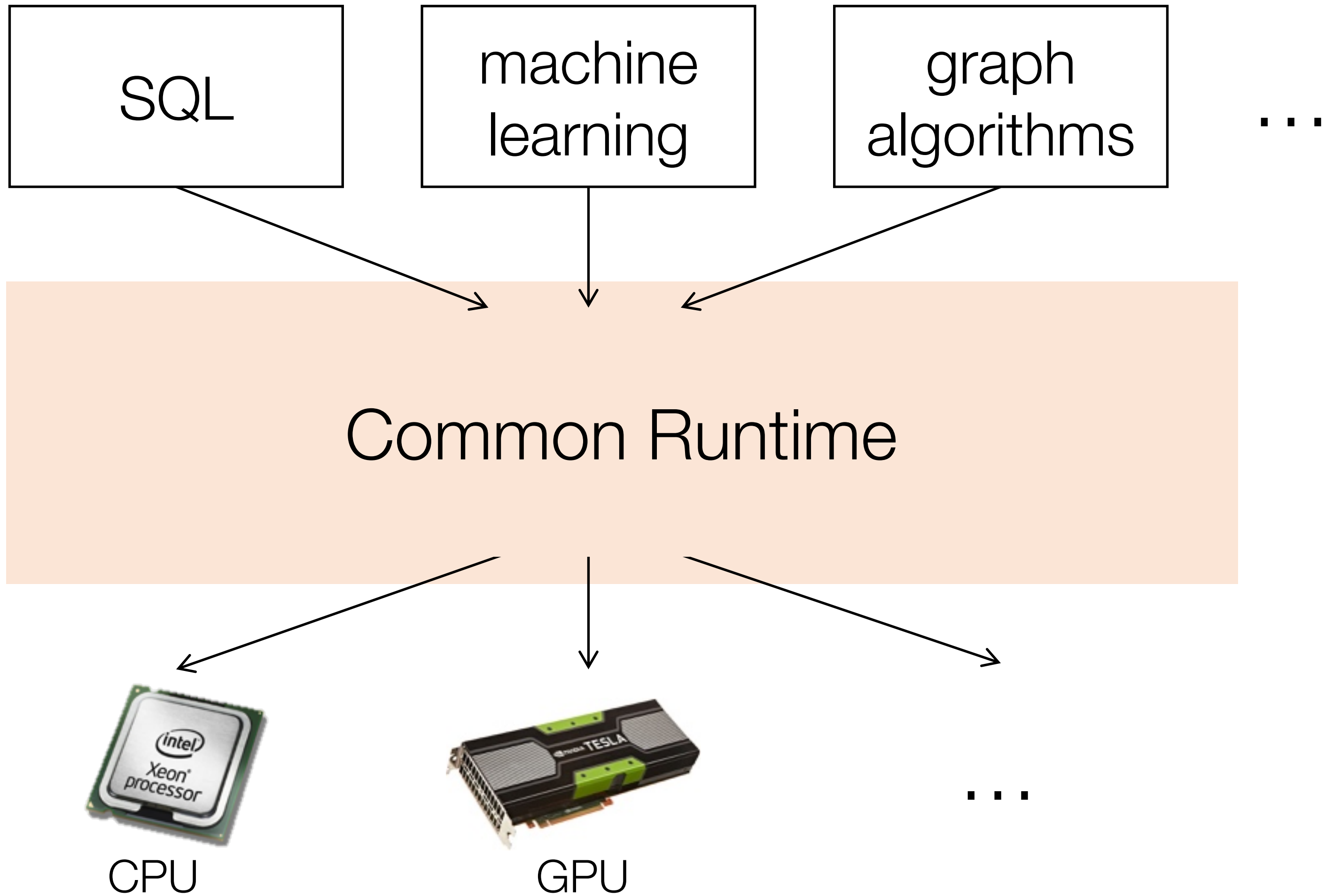
Growing gap between memory/processing makes traditional way of combining functions worse

```
data = pandas.parse_csv(string)

filtered = pandas.dropna(data)

avg = numpy.mean(filtered)
```

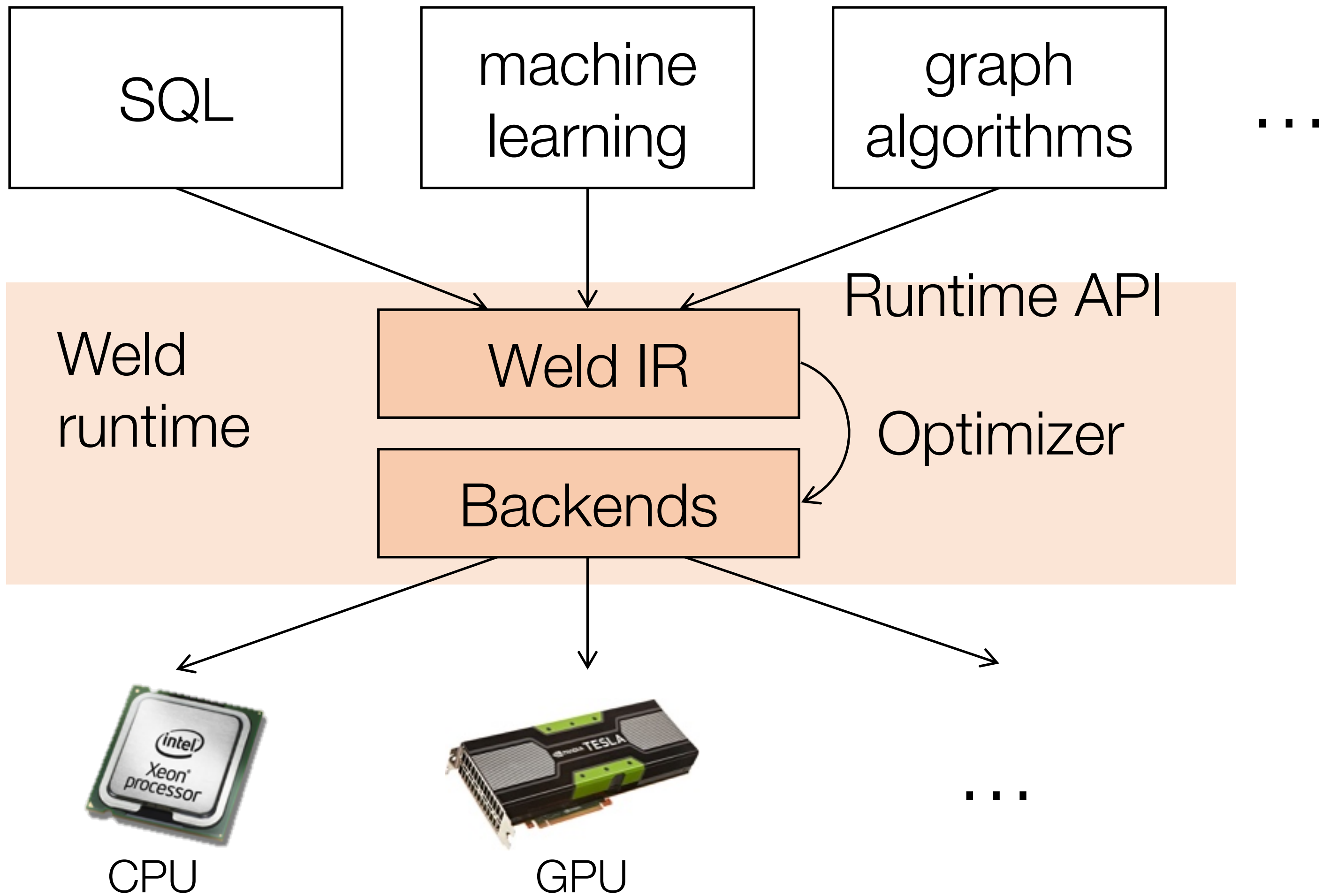


Up to 30x slowdowns in NumPy, Pandas, TensorFlow, etc. compared to optimized C implementation

Optimizing Across Frameworks



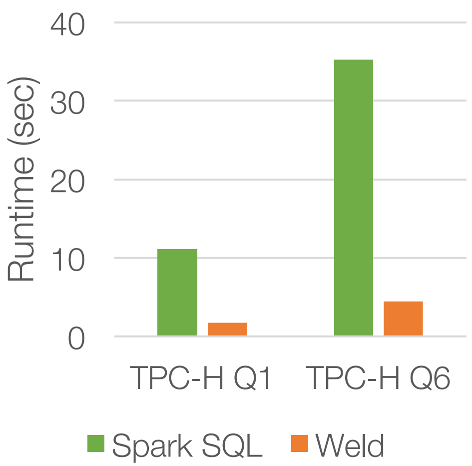
Optimizing Across Frameworks

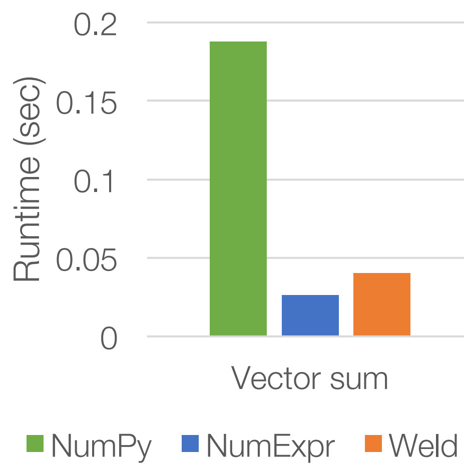


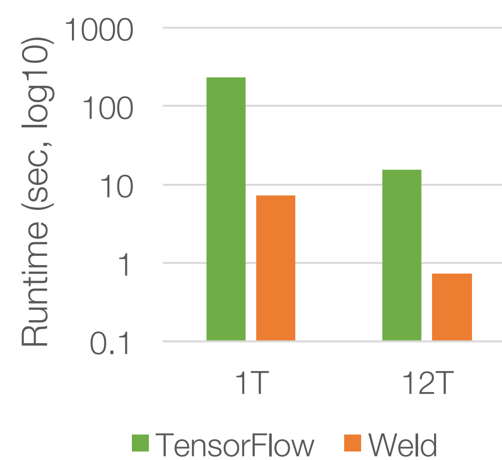
Integrations

Partial Prototypes of Pandas, NumPy, TensorFlow and Apache Spark

Existing Libraries + Weld = 30x







Spark SQL

TPC-H

6.8x Faster

NumPy

Vector Sum

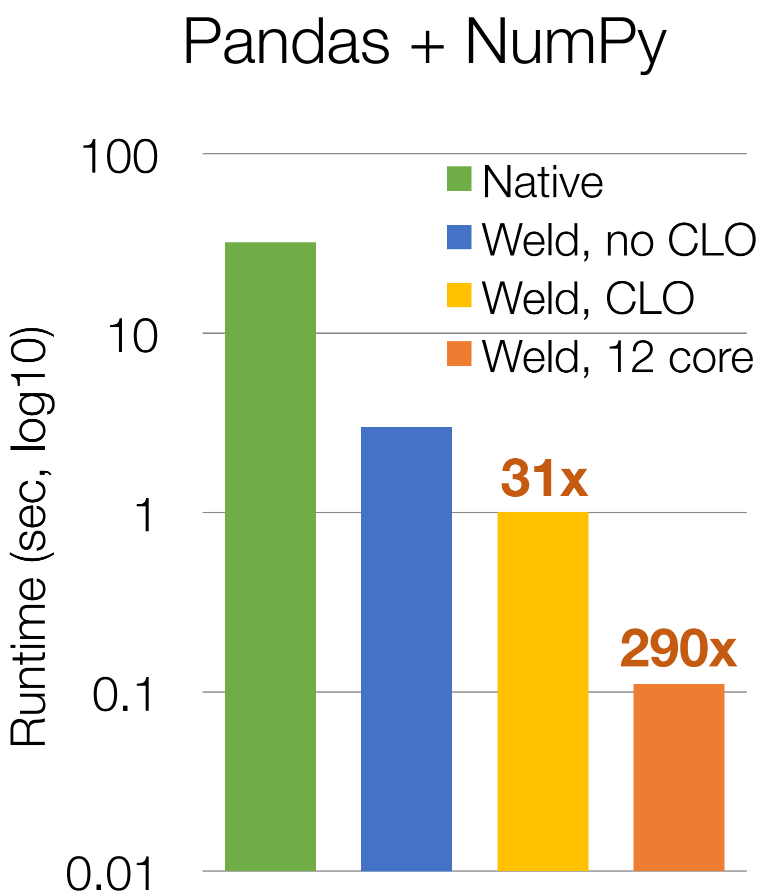
4.5x Faster

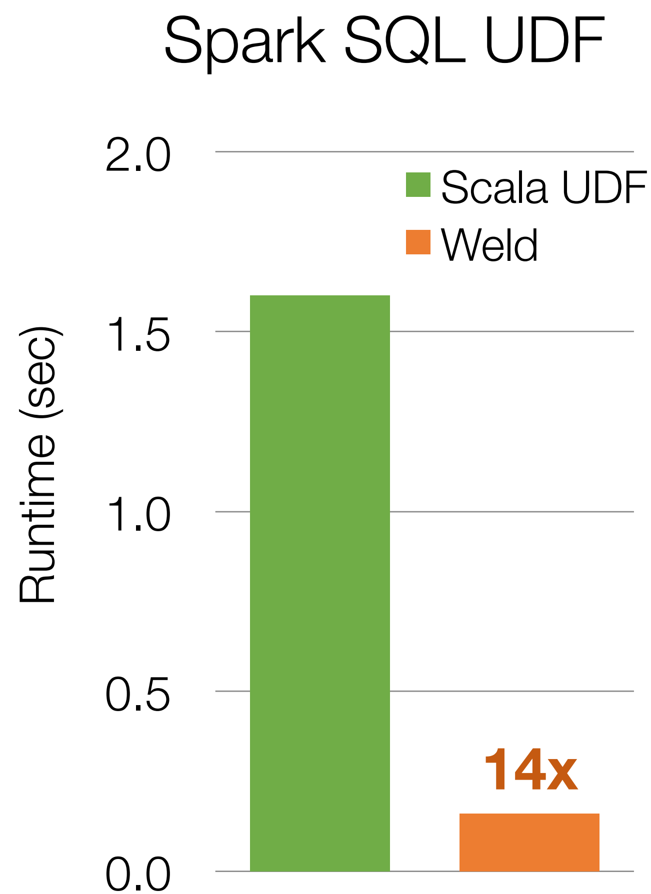
TensorFlow

Logistic Regression

30x Faster

Cross libraries + Weld = 290x





CLO = with cross library optimization

Integration Effort

Small up front cost to enable Weld integration

- » 500 LoC

Easy to port over each operator

- » 30 LoC each

Incrementally Deployable

- » Weld-enabled ops work with native ops

Implementation

Around 12K lines of Rust

» Fast and safe! Perfect for embedding

Parallel CPU backend using LLVM

GPU support coming soon

Weld is Open Source!

STANFORD INFOLAB

Weld

Fast parallel code generation for data analytics frameworks.

Download ZIP File	Download TAR Ball	View On GitHub

Hosted on GitHub Pages — Theme by orderedlist

News

- Weld is being presented at Strata + Hadoop World in San Jose, CA!
- A short position paper describing Weld was presented at CIDR 2017!

Installation and Tutorials

- Installation Instructions
- Python API Tutorial
- Grizzly: Pandas on Weld
- Support

Motivation

Weld is a runtime for improving the performance of data-intensive applications. It optimizes across libraries and functions by expressing the core computations in libraries using a small common *intermediate representation*, similar to CUDA and OpenCL.

Modern analytics applications combine multiple functions from different libraries and frameworks to build complex workflows. Even though individual functions can achieve high performance in isolation, the performance of the *combined workflow* is often an order of magnitude below hardware limits due to extensive data movement across the functions. Weld's take on solving this problem is to lazily build up a computation for the entire workflow, optimizing and evaluating it only when a result is needed.

weld.stanford.edu

Rest of This Talk

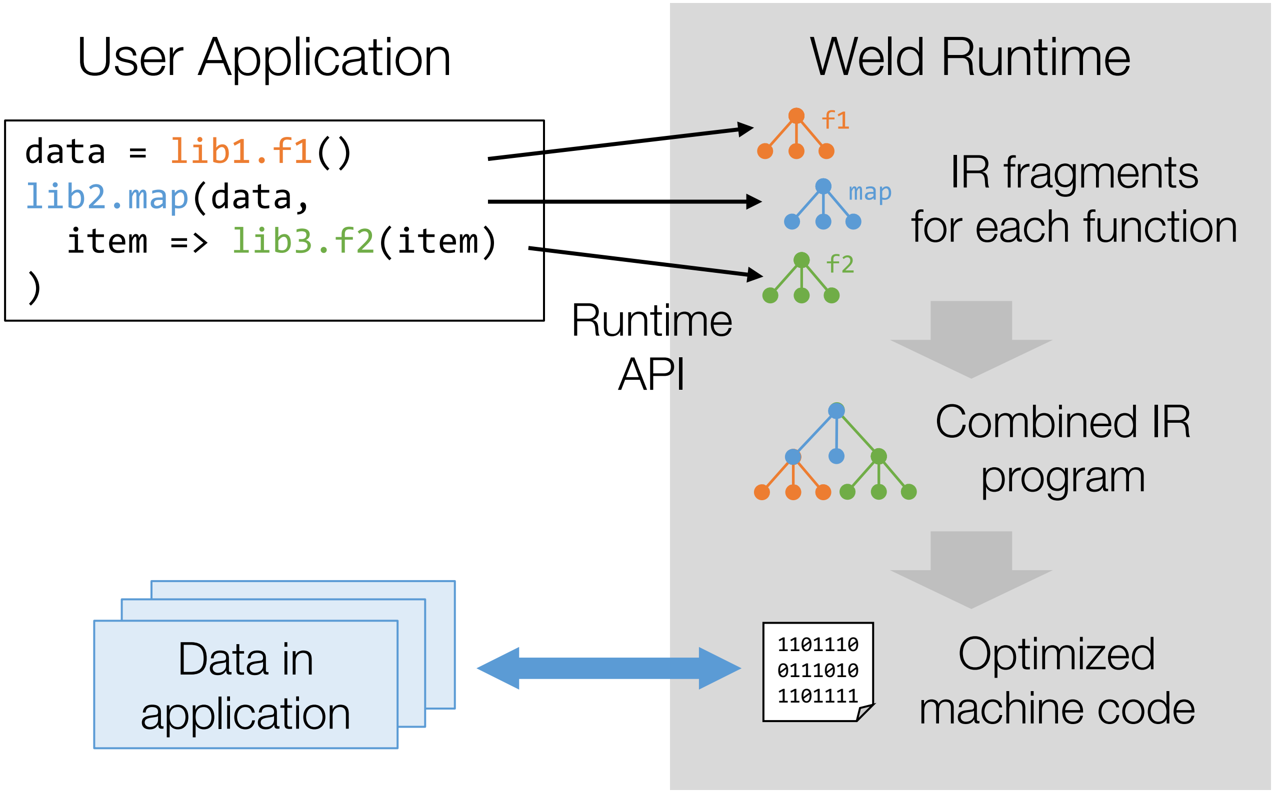
Runtime API: Enabling cross-library optimization

Weld IR: Enabling speedups and automatic parallelization

Grizzly: Using Weld to build a Faster Pandas

Runtime API

Uses lazy evaluation to collect work across libraries



Weld IR

Designed to meet three goals:

1. **Generality:** support diverse workloads and nested calls

2. **Ability to express optimizations:** e.g., loop fusion, vectorization, loop tiling

3. **Explicit parallelism and targeting parallel hardware**

Weld IR: Internals

Small IR with only two main constructs.

Parallel loops: iterate over a dataset

Builders: declarative objects for producing results

- » E.g. append items to a list, compute a sum
- » Can be implemented differently on different hardware

Captures relational algebra, functional APIs like Spark, linear algebra, and composition thereof

Examples

Implement functional operators using builders

```
def map(data, f):
    builder = new vecbuilder[int]
    for x in data:
        merge(builder, f(x))
    result(builder)


def reduce(data, zero, func):
    builder = new merger[zero, func]
    for x in data:
        merge(builder, x)
    result(builder)
```

Example Optimization: Fusion

```
squares = map(data, x => x * x)
sum = reduce(data, 0, +)
```

```
bld1 = new vecbuilder[int]
bld2 = new merger[0, +]
for x in data:
    merge(bld1, x * x)
    merge(bld2, x)
```

Loops can be merged into one pass over data


“Active, agile, hard to out-run, top speed of 35 MPH”

Grizzly

A subset of Pandas integrated with Weld

- Operators ported over including `unique`, `filter`, `mask`
- Easy to port more!

Same API as native Pandas so zero changes to applications

Transparent single-core and multi-core speedups

Grizzly in action

```
import pandas as pd

# Read dataframe from file
requests = pd.read_csv('filename.csv')


# Fix requests with extra digits
requests['Incident Zip'] = requests['Incident Zip'].str.slice(0, 5)

# Fix requests with 00000 zipcodes
zero_zips = requests['Incident Zip'] == '00000'
requests['Incident Zip'][zero_zips] = np.nan

# Display unique incident zips
print requests['Incident Zip'].unique()
```

Adapted from http://pandas.pydata.org/pandas-docs/stable/tutorials.html (chapter 7)

Grizzly in action

```
import pandas as pd, grizzly as gr

# Read dataframe from file
raw_requests = pd.read_csv(‘filename.txt’)
requests = gr.DataFrame(raw_requests)

# Fix requests with extra digits
requests['Incident Zip'] = requests['Incident Zip'].str.slice(0, 5)

# Fix requests with 00000 zipcodes
zero_zips = requests['Incident Zip'] == '00000’
requests['Incident Zip'][zero_zips] = np.nan

# Display unique incident zips
print requests['Incident Zip'].unique().evaluate()
```

Adapted from http://pandas.pydata.org/pandas-docs/stable/tutorials.html (chapter 7)

Demo

Conclusion

Changing the interface between libraries can *speed up data analytics applications by 10-100x* on modern hardware

Try out Weld for yourself: weld.stanford.edu

STANFORD INFOLAB