Phylanx : Updates

Primitives

- Data Types
  - Support for 0 , 1 and 2 - Dimensional data
  - Blaze for underlying data representation
    - High performance , Vectorization , Parallel Execution , Portable
- More than 50 (and counting) primitives in total currently
- Basic operations/ Controls / Decisions / IO
  - Addition, Multiplication, Subtraction, Division, For, While, If, If-else, file_read
- NumPy Functionalities
  - Dot product, inverse operation, transpose operation, stacking operations etc.
  - More than 20 (and counting) NumPy Operations currently supported

Logistic Regression

- Logistic Regression Algorithm is a regression algorithm used in machine learning
  - Used for classification
- Reference implementation of Logistic Regression Algorithm
  - Simple implementation
- Primitives Used
  - Exponential
  - Transpose
  - Dot

Simplified Logistic Regression in Python

```
def LogisticRegression(x, y, iterations, alpha):
    w = np.zeros(x.shape[1])
    Transx = x.transpose()
    for step in range(iterations):
        g = np.dot(x, w)
        p = 1.0 / (1.0 + np.exp(-g))
        error = p - y
        gradient = np.dot(Transx, error)
        w = w - alpha * gradient
    return w
```

@Phylanx("PhySL") Decorator

```
@Phylanx("PhySL")
def lra(x, y, alpha, iterations):
    weights = np.zeros(shape(x, 1))
    transx = np.transpose(x)
    p = np.zeros(shape(x, 0))
    error = np.zeros(shape(x, 0))
    gradient = np.zeros(shape(x, 1))
    step = 0
    while step < iterations:
        p = 1.0 / (1.0 + np.exp(-np.dot(x, weights)))
        error = p - y
        gradient = np.dot(transx, error)
        weights = weights - (alpha * gradient)
        step += 1
    return weights
```

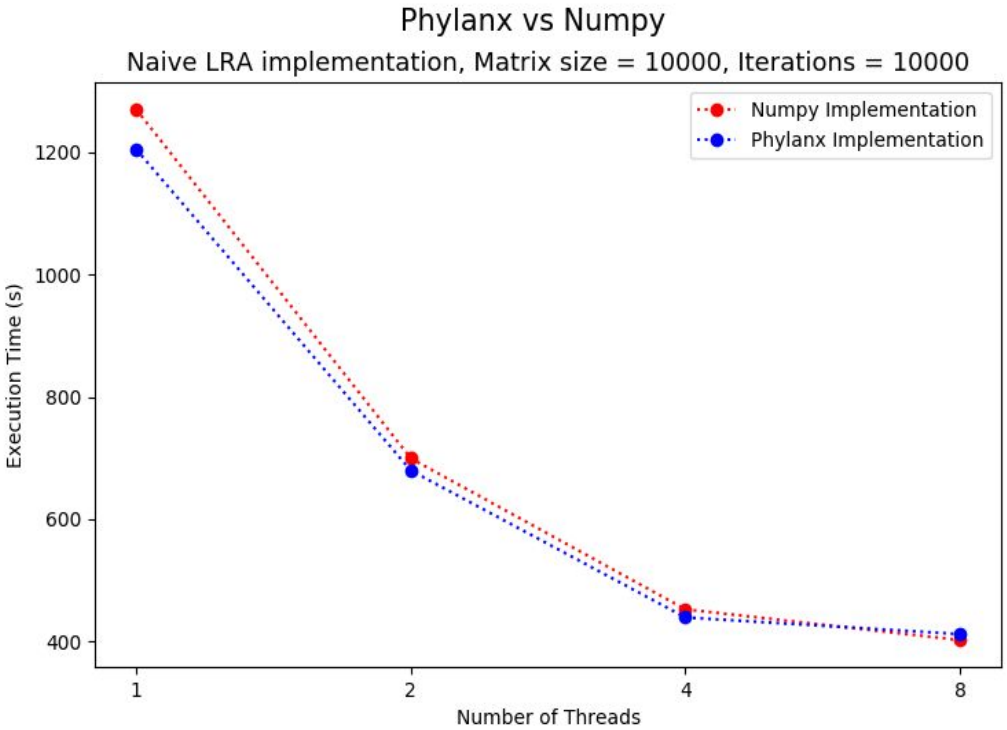
```
def LogisticRegression(x, y, iterations, alpha):
    w = np.zeros(x.shape[1])
    Transx = x.transpose()
    for step in range(iterations):
        g = np.dot(x, w)
        p = 1.0 / (1.0 + np.exp(-g))
        error = p - y
        gradient = np.dot(Transx, error)
        w = w - alpha * gradient
    return w
```

Logistic Regression in PhySL

```
define(lra, x,y, alpha, iteration,
    block(
        define(weights, constant(0.0, shape(x, 1))),
        define(transx, transpose(x)),
        define(p, constant(0.0, shape(x, 0))),
        define(error, constant(0.0, shape(x, 0))),
        define(gradient, constant(0.0, shape(x, 1))),
        define(step, 0),
        while(
            step < iteration,
            block(
                store(p, 1.0 / (1.0 + exp(-dot(x, weights)))),
                store(error, p - y),
                store(gradient, dot(transx, error)),
                parallel_block(
                    store(weights, weights - (alpha * gradient)),
                    store(step, step + 1)
                )
            )),
        weights
    ))
```

```
def LogisticRegression(x, y, iterations, alpha):
    w = np.zeros(x.shape[1])
    Transx = x.transpose()
    for step in range(iterations):
        g = np.dot(x, w)
        p = 1.0 / (1.0 + np.exp(-g))
        error = p - y
        gradient = np.dot(Transx, error)
        w = w - alpha * gradient
    return w
```

Initial Performance Comparisons



Future Work : Improving the LRA

- Solvers
- Multiple Implementations / Optimizations
  - Very Large Datasets
  - Multinomial LRA
- Distributed LRA
- More Algorithms !!!!

Future Work : Research Prospects

- Fine grained tasks
- Overheads associated with fine grained tasks
- Reduction of overheads
- Task inlining
- Adaptively reduce overheads
- Parcel coalescing
- Adaptive parcel coalescing

Thank you!