Blaze

Stephen Diehl

Continuum Analytics

Warning

Extreme talk, code and math ahead!

Problems with NumPy

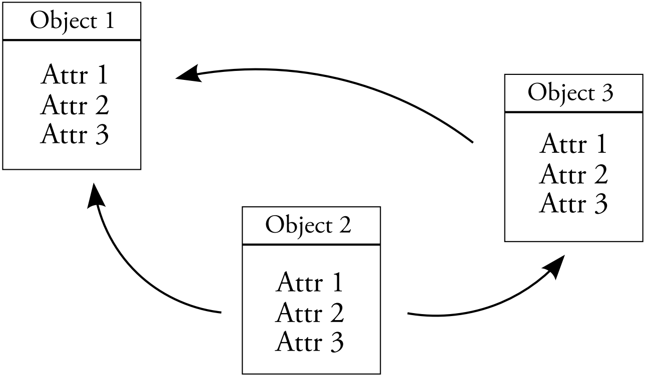
- Limited support for heterogeneous data
- Missing values (`na`)
- Uses a single buffer with strides overlayed
- Cannot resize in place
- Labeled arrays
- No support for arrays larger than available memory (out-of-core)
- No support for Distributed Arrays

What is Blaze

How do we extend NumPy so that we can map array computations onto modern hardware?

- Blaze is
  - a generalization of NumPy
  - a datashape description language
  - a generalized notion of data access
  - a way to view data stores as arrays and tables

Object Oriented Programming



Array Oriented Programming

	Attr 1	Attr 2	Attr 3
Object 1			
Object 2			
Object 3			

Array Oriented Programming

- APL
  - First class arrays
- Chapel / ZPL
  - Domain Maps
  - Tensor Expressions

APL Dialects

The notion of array shape is extremely fluid.

```
>   3 4 2 $ a
0 1 2
3 4 5

6 7 8
0 1 2

3 4 5
6 7 8

0 1 2
3 4 5

...
```

```
forall ijkl in A do
    A[ijkl] = reduce f [ijl] + reduce g [jkl]
```

Very low barrier between the high-level mathematical notation and the programming notation.

$$(\mathbf{a} \times \mathbf{b})^i = \varepsilon_{ijk} a^j b^k$$

Functional Programming

- Haskell
  - Types
  - Lazy Evaluation
  - Data Parallel Haskell
  - Stream Fusion
  - Repa

End-User Perspectives

Blaze is built around separation of concerns.

- Domain experts
  - Richer structure to express high level ideas.
- Algorithm writers
  - More information (type, shape, layout) to do clever optimizations.
- Researchers
  - A platform in which to explore data and task parallelism.

Core Idea

Leverage the existing community and projects.

Ecosystem Requirements

	Numpy	DataFrame	Business Data
Structures	• Matrices	• Timeseries • Hierarchical	• Multidimensional Models • Data Cubes • OLAP / MDX
	Vectors	→	Sets
Data Access	Sequential	→	Random
Dimensions	Scalar	→	Categorical

Strong Coupling of Values and Data Locality → Weak Coupling of Values and Data Locality

Walk kernels over grid structure of the array → Statistical Aggregations

Current Work

- Data Structures
  - PyTables
    - `carray`
    - `varray`
    - `ctable`
  - pandas
  - memmap, masked array
- Computation
  - theano
  - numexpr
  - clyther

- What defines a NumPy array?

```
In[0]: A = np.array([[1, 2], [3, 4]], dtype=np.int32)
array([[1, 2],
       [3, 4]], dtype=int32)
```

Data

```
In[1]: bytes(A.data)
Out[1]: b'\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00...'
```

Shape

```
In[2]: A.shape
Out[2]: (2,2)
```

DType

```
In[3]: A.dtype
Out[3]: dtype('int32')
```

Strides

```
In[4]: A.strides
Out[4]: (8,4)
```

Flags

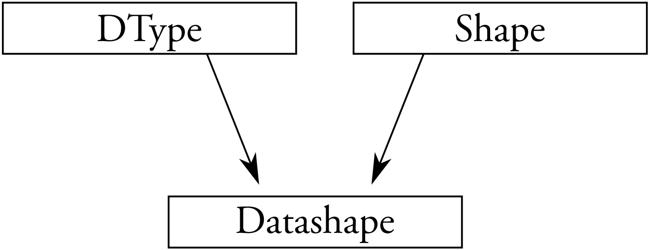
```
In[5]: A.flags
Out[5]:
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
```

How do we make extend NumPy so that we can map array computations onto modern hardware?

- How do we generalize dtype system?
- How do we construct Numpy arrays out of multiple buffers?
- How do we dispatch Numpy expressions to specialized kernels?

How do we make `dtype` more general?

Datashape



Datashape

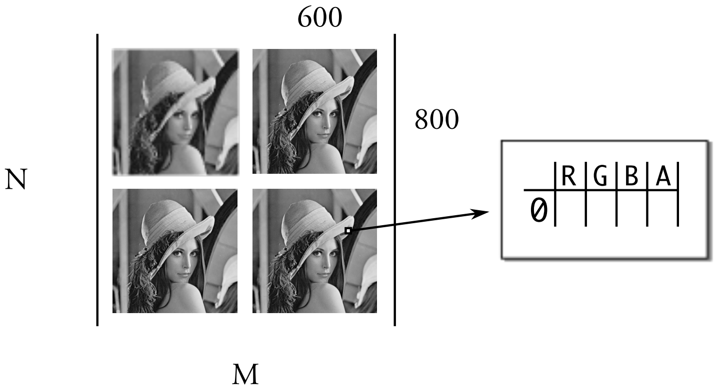
Datashapes are an extension of dtype, expressed as a small DSL.

Directory of Images

Endow existing data with structure.

```
$ ls
/folder1
    lena1.png
    lena2.png
/folder2
    lena3.png
    lena4.png
/folder3
    lena5.png
    lena6.png
...
/folderN
```

`(N, M, 800, 600, RGBA)`



CSVs

Endow existing data with structure.

```
$ ls
/particle_accelerator
    measurement1.csv
    measurement2.csv
    measurement3.csv
```

- Multiple views of the same hypercube.
- Fluid interpretation of shape.

	Ax	Ay	Az	Bx	By	Bz
0	3.1	4.1	5.9	2.6	5.3	5.8
1	2.7	1.8	2.8	1.8	2.8	4.5
2	0.5	7.7	2.1	5.6	6.4	9.0

$$
\begin{array}{cc}
y & \begin{vmatrix} (A_0,\ B_0) & (A_1,\ B_1) & (A_2,\ B_2) \\ (A_3,\ B_3) & (A_4,\ B_4) & (A_5,\ B_5) \\ (A_6,\ B_6) & (A_7,\ B_7) & (A_8,\ B_8) \end{vmatrix} \\
 & x
\end{array}
$$

$$
\begin{array}{ccc}
 & \begin{array}{cccc} \phantom{\begin{vmatrix}1 & 5\end{vmatrix}} & \phantom{\begin{vmatrix}1 & 5\end{vmatrix}} & \phantom{\begin{vmatrix}1 & 5\end{vmatrix}} & B_x \end{array} & \\
A_y & \begin{vmatrix} \begin{vmatrix}1 & 5\end{vmatrix} & \begin{vmatrix}2 & 7\end{vmatrix} & \begin{vmatrix}0 & 1\end{vmatrix} & \begin{vmatrix}3 & 1\end{vmatrix} \\ \begin{vmatrix}3 & 1\end{vmatrix} & \begin{vmatrix}7 & 8\end{vmatrix} & \begin{vmatrix}6 & 4\end{vmatrix} & \begin{vmatrix}4 & 9\end{vmatrix} \\ \begin{vmatrix}8 & 2\end{vmatrix} & \begin{vmatrix}1 & 5\end{vmatrix} & \begin{vmatrix}0 & 8\end{vmatrix} & \begin{vmatrix}1 & 2\end{vmatrix} \end{vmatrix} & B_y \\
 & A_x &
\end{array}
$$

```
5, 5, int32

N, M, 800, 600, RGBA

300, Record(uid=int32, name=string)

Var(25), Record(uid=int32, name=string)

200, 300, Either(int32, na)

10, 10, Quaternion
```

Record

```python
class Stock(RecordClass):
    name    = string
    open    = decimal
    close   = decimal
    max     = int64
    min     = int64

    @derived
    def mid(self):
        # Types are inferred
        return (self.min + self.max)/2
```

Type Constructors

- Alias types

```
RGBA = Record(r=int32, g=int32, b=int32, a=int8)
```

```
Stock = Record(open=decimal, close=decimal, name=string)
```

- Parametric types

```
SquareMatrix A = A, A
```

```
Mesh A B = 5, 5, Record(x=A, y=B)
```

```
Portfolio R = R, Stock
```

Usage

```
In[0]: stream = blaze.open('yahoo://GOOG:20120100')

In[1]: nd = NDTable(stream, datashape='Var(100), Stock')

In[2]: nd[0]
name          open    close   max  min  mid
Google Inc.   681.79  705.92  695  621  658
```

Syntax not final.

Coordinates

- How do we construct NumPy arrays out of multiple buffers?

Memory Access

```c
void *
PyArray_GetPtr(PyArrayObject *obj, npy_intp* ind)
{
    int n = obj->nd;
    npy_intp *strides = obj->strides;
    char *dptr = obj->data;

    while (n--) {
        // Core pointer arithmetic of Numpy
        dptr += (*strides++) * (*ind++);
    }
    return (void *)dptr;
}
```

Numpy Data Access

$$S_n = \text{itemsize} \times \prod_{n+1}^{m} \text{shape}_n$$

$$I = (i, j, k, \ldots)$$

$$A_{i,j,k} = S \cdot I$$

Blaze Data Access

$$S_n = \prod_{n+1}^{m} \text{datashape}_n$$

$$I = (i, j, k, \ldots)$$

$$A_{i,j,k} = f(S; I)$$

- Stride vectors are now functions of indexers and the datashape.

$$\mathrm{S}_n = \prod_{n+1}^{m} \mathrm{datashape}_n$$

- Generalized indexers: scalars, labels, references.

$$\mathrm{I} = (i, j, k, \ldots)$$

$$\mathrm{A}_{i,j,k} = f(\mathrm{S}; \mathrm{I})$$

- In the case of strided memory access:

$$f = (\cdot)$$

Coordinates

- Coordinates
  - Atlas of regions in memory to resolve partitions.
  - Charts of memory with coordinate systems.
  - Coordinate transformations between charts.
  - Index space transformations are homomorphisms between charts.

$$
\operatorname{vstack}\left(\underset{3\times 4}{\begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}}, \underset{3\times 4}{\begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix}} \right)^{T}
$$

$$
6\times 4 \quad \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hdashline 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix}
\qquad\qquad
\overset{4\times 6}{\left|\begin{array}{ccc:ccc} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right|}
$$

```
f = vstack
p = (0,3)
```

```
g = transpose
```

$$f(i,j) = (i,j) + p$$

$$g(i,j) = (j,i)$$

$$(f.g)(i,j) = (j,i) + g(p)$$

$$4 \times 6$$

$$\texttt{(3,5)} \quad \begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix}$$

(2,3) $$\begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix}$$

`(3,2)`
$$\begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix}$$

$$\begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & \boxed{1} & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{vmatrix}$$

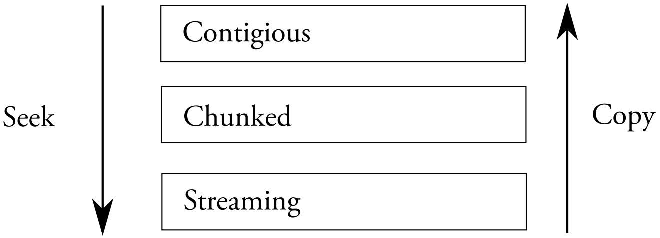
```
int32  = 4 bytes
offset = 16
```

```
\x00\x00\x00\x00\x00\x00\xf0?\x00\x00\x00\x00\x00\x00
\xf0?\x00\x00\x00\x00\x00\x00\xf0?\x00\x00\x00\x00\x00
\x00\xf0?\x00\x00\x00\x00\x00\x00\xf0?\x00\x00\x00\x00
\x00\x00\xf0?\x00\x00\x00\x00\x00\x00\xf0?\x00\x00\x00
\x00\x00\x00\xf0?\x00\x00\x00\x00\x00\x00\xf0?\x00\x00
\x00\x00\x00\x00\xf0?\x00\x00\x00\x00\x00\x00\xf0?\x00
\x00\x00\x00\x00\x00\xf0?'
```

Blaze Data Descriptors

At some level our coordinates manifest simply as byte offsets in some storage medium.

- Memory Like
  - Arbitrary Slices
  - Random Seeks
- File Like
  - Chunks
  - Random Seeks
- Stream Like
  - Chunks
  - Sequential Seeks


Contigious
Chunked
Streaming
Seek
Copy

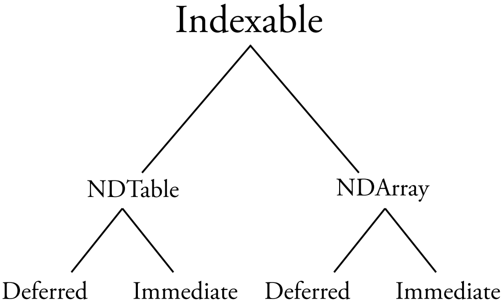
Dispatch

How do we dispatch NumPy expressions to specialized kernels?

Lazy Evaluation

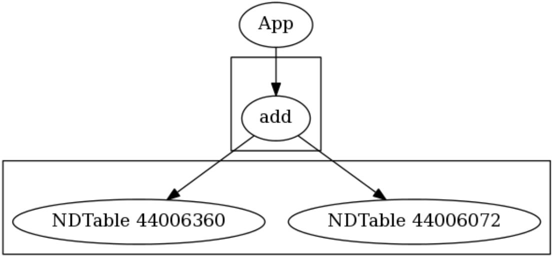
Expressions are not evaluated until forced.

- Deferred table.
  - Appends graph nodes onto existing graph.
  - Data processing scripts, fine grained control over evaluation.
- Immediate table.
  - Forces graph immediately.
  - Data exploration at REPL, when you just want a number.


Indexable
NDTable
NDArray
Deferred
Immediate
Deferred
Immediate

Expression Graph

```
In[0]: x = a + b
In[1]: type(x)
<type 'blaze.graph.ArrayNode'>
```



```
In[0]: x = a + b
In[1]: type(x)
<type 'blaze.graph.ArrayNode'>

In[2]: e = LA.eig(x)
In[3]: e.eval()
Out[3]:
array([[ 0.70710678+0.j          ,  0.70710678+0.j          ],
       [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]])
```

Expression Typing

Typing and polymorphic operators coexist.

```
class add(BinaryOp):
    signature = 'a -> b -> b'
    dom = [scalar, array]

In[0]: expr = 2 + [1,2,3]
In[1]: expr = [1,2,3] + 2

In[2]: expr.dom
[int32[:], int32[:]]

In[3]: expr.cod
int32[:]
```

Transformations

$$f(a, b) :: (\text{Indexable}, \text{Indexable}) \to \text{Indexable}$$

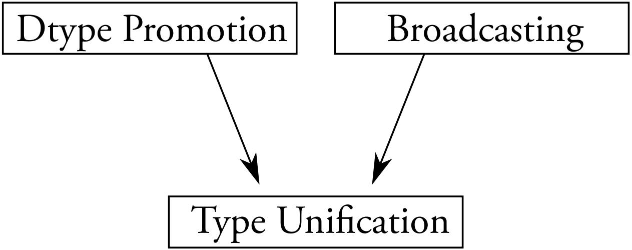
- **Index space transformation**
  - How coordinates transform under f.
- **Value space transformation**
  - How values transform under f.
- **Metadata space transformation**
  - How metadata is merged from operands a, b.

Preserving type and metadata information allows us to write more clever algorithms that avoid needless seeks and scans.

Nuances of Evaluation

- Forcing Evaluation
  - Many properties of arrays are not knowable a priori. Evaluation is forced when we need to peek into a opaque type.
  - Can't write coordinate transformations in terms of coordinates we can't describe.
  - Side effectful operations. (`numpy.fill`, `numpy.put`)
  - (`reduce`, `groupby`) also force evaluation. In general operations which are not index space homomorphisms.
- Just like NumPy. There are benefits to writing in functional style, but one can still drop into raw memory manipulation when needed.

Type Unification



NumPy

```
A                5 x 4
B                    1
broadcast(A,B)   5 x 4
```

Blaze

```
A                3, 2, Var(10), int32
B                   1, Var(100), float32
unify(A,B)       3, 2, Var(100), float32
```

Metadata

- Data Warehousing
  - Provenance: Where does this number come from?
  - Propagate metadata about source through ETL pipeline.
- Orthogonal matrix
  - $A^{-1} \rightarrow A^{T}$
  - **Value Space Transform** $\rightarrow$ **Index Space Transform**
- Matrix density
  - density(A) + density(B) in terms of density(A,B)

Provide an extensible algebra of propagating information through the expression graph.

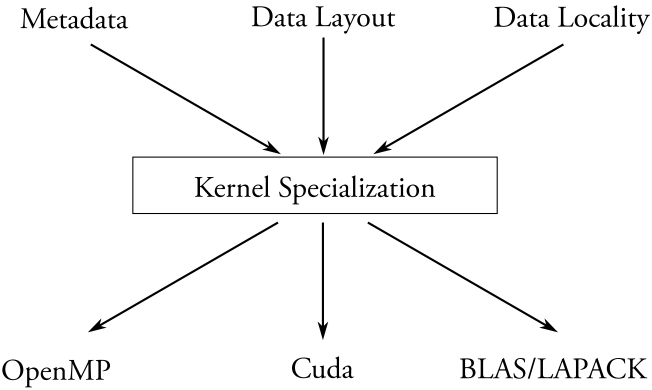
Global Arrays

Blaze as the lingua franca of Python data exchange.

- Remote Computation
  - Push code to data
- Remote Data Retrieval
  - Pull data to code

Compiler Backend

- Code generation
  - Numba & Minivect (target LLVM)
  - Array VM (target custom bytecode)



Blaze

Blaze is still a work in progress.

Look for more details at the end of November.

- Blaze will be open source!
- Blaze will have tight integration with Numba.
- Blaze Studio will have premium features on top of Blaze and Numba and will include IOPro.

Blaze Project

People

- Travis Oliphant
- Peter Wang
- Mark Wiebe (`numpy`)
- Francesc Alted (`pytables`, `numexpr`)
- Mark Florisson (`minivect`, `numba`)
- Stephen Diehl

Contact

- Github: github.com/ContinuumIO/blaze
- URL: http://www.continuum.io
- Twitter: @ContinuumIO