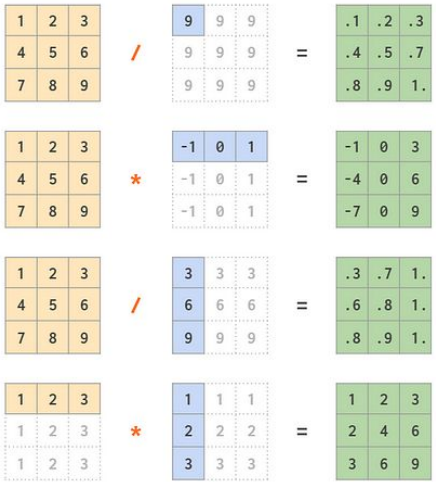
PyData/Sparse and Finch

PyData Global 2025

Array Programming is Productive



Matrix multiplication

$$c_{ik} = \sum_j a_{ij} b_{jk}$$

```
c = np.einsum('ij,jk->ik', a, b)
```

Tensor multiplication

$$c_{ijlm} = \sum_k a_{ijk} b_{klm}$$

```
c = np.einsum('ijk,klm->ijlm', a, b)
```

TensorFlow

PyTorch

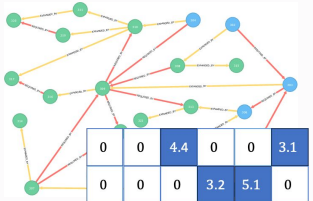
GRAPHBLAS

NumPy

Sparsity is everywhere…

Social Networks

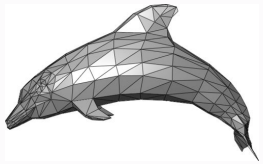
sparse:



$A =$

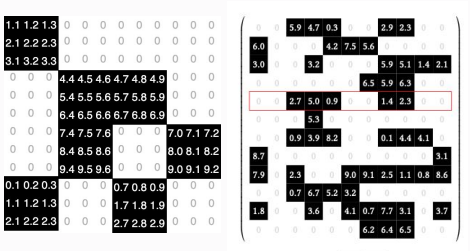
0	0	4.4	0	0	3.1
0	0	0	3.2	5.1	0
4.4	0	0	2.0	0	2.5
0	3.2	2.0	0	4.7	0
0	5.1	0	4.7	0	2.4
3.1	0	2.5	0	2.4	0

Machine Learning



$$r_i = \frac{1-d}{N} + \sum_j dA_{ij} r_i$$

block sparse:



Scientific Computing

$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

$$\begin{bmatrix} B_{11} & B_{12} & 0 & \cdots & \cdots & 0 \\ B_{21} & B_{22} & B_{23} & \ddots & \ddots & \vdots \\ 0 & B_{32} & B_{33} & B_{34} & \ddots & \vdots \\ \vdots & \ddots & B_{43} & B_{44} & B_{45} & 0 \\ \vdots & \ddots & \ddots & B_{54} & B_{55} & B_{56} \\ 0 & \cdots & \cdots & 0 & B_{65} & B_{66} \end{bmatrix}$$

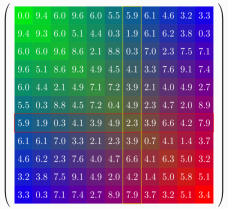


Image Processing

convolution (Toeplitz):

$y_i = \sum_j x_{i-j} k_j \qquad A_{ij} = x_{i-j}$

i

5	3	7	2	0	0	0
0	5	3	7	2	0	0
0	0	5	3	7	2	0
0	0	0	5	3	7	2

j

A

$i - j$

5	3	7	2

x

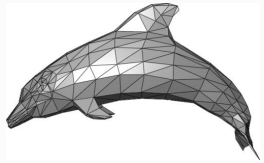
padding:

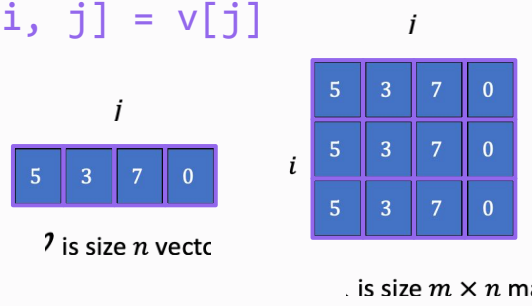
0	0	0	0	0
0	5	3	7	0
0	5	3	7	0
0	5	3	7	0
0	0	0	0	0

Sparsity is Challenging

Dense Tensors

```
for i = 1:n
  for j = 1:m
    A[i, j] = v[j]
```



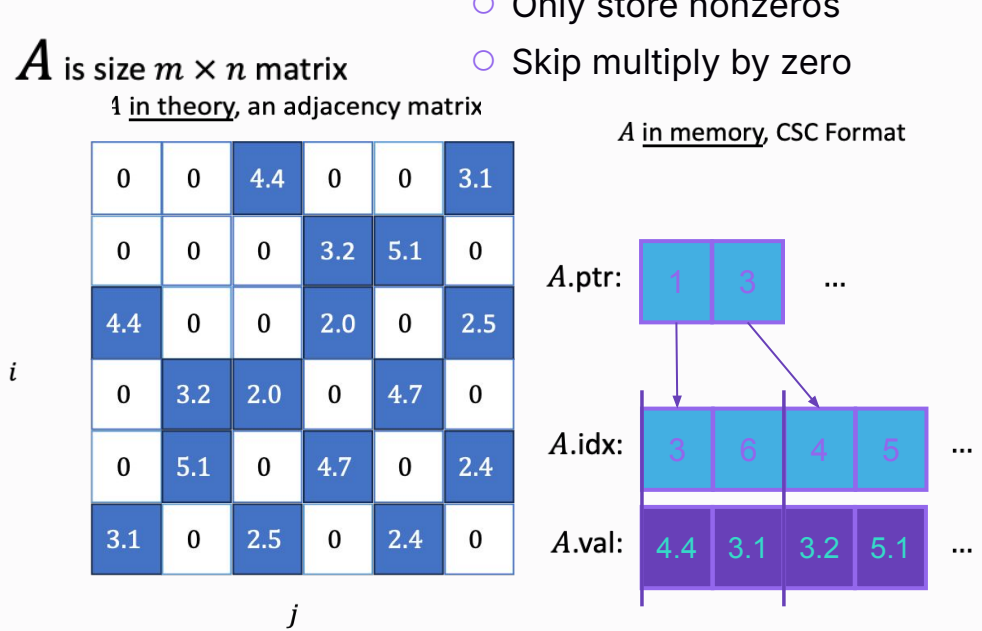



```
A[i, j] = A.data[(i - 1)*n + j]
```

A is dense

Sparse Tensors

- Theory != Memory
  - Nonzeros << size
  - Only store nonzeros
  - Skip multiply by zero



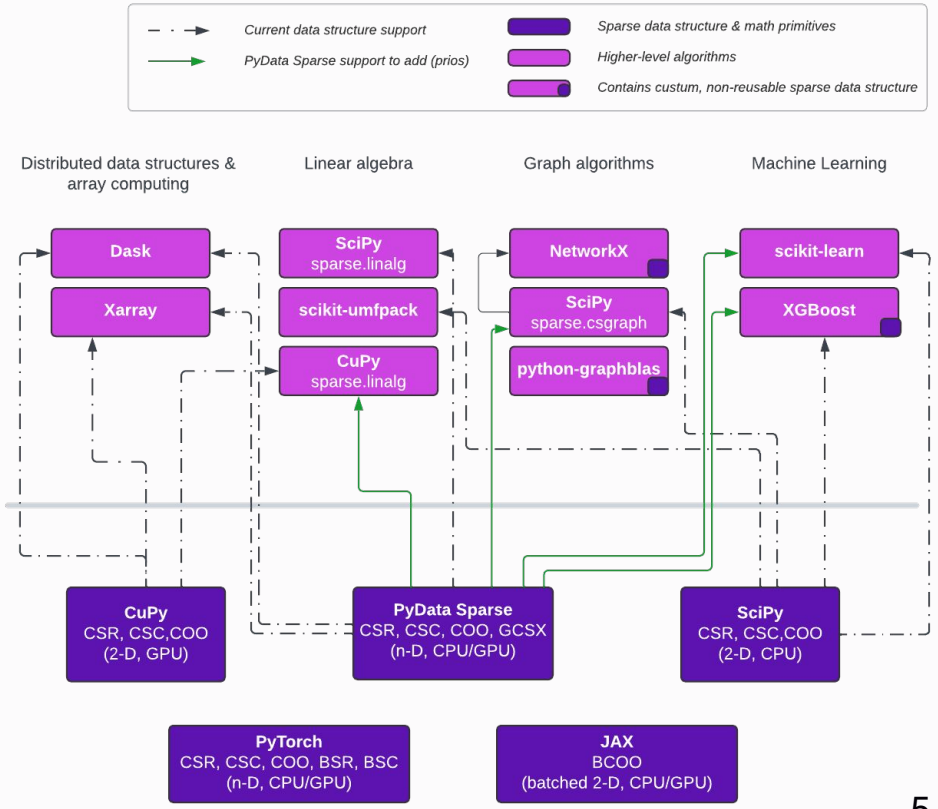
Fragmented sparse support today

Support across key libraries in the PyData ecosystem is fragmented.

There's a desire to have a single library central to the ecosystem that is n-dimensional, with CPU and GPU support, performant, and with broad linear algebra and graph algorithm support.

	Internals	Backend	Adoption
SciPy	2-dim*	C, Cython	Widely used
PyTorch	N-dim	PyTorch	Beta version
pydata/sparse	N–dim	Numba	E.g. in Dask

* only COO is N-dim in SciPy



Implementation Burden Grows Exponentially...

Operation

$$C_{ij} = \sum_k A_{ik} * B_{kj}$$

$$y_i = \sum_j A_{ij} x_j$$

$$C_{ij} \mathrel{+}= M_{ij} \sum_k A_{ik} * B_{kj}$$

⋮

×

Structure

Sparse
Banded
Symmetric
Blocked
⋮

×

Data Format

CSR
COO
Hash
DIA
⋮

×

Architecture

CPU
GPU
TPU
ASIC
⋮

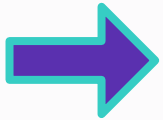
$$a = Bc$$
$$A = B \quad a = B^T c$$
$$a = Bc + a \quad a = Bc + b \quad a = b + c$$
$$A_{ij} = \sum_k B_{ijk} * c_k \quad a = B^T c + d \quad a = \alpha Bc + \beta a$$
$$a = BCd \quad A = B + C \quad A = B \odot C \quad A = BC \quad A = 0 \quad A = B^T \quad A = (B + C)d$$
$$A = (B + C)(d + e) \quad A_{ik} = \sum_j B_{ijk} * c_j \quad A_{ij} = \sum_k B_{ijk} * c_k \quad A = A + \alpha I \quad A = B + C + D$$
$$A_{ilk} = \sum_j B_{ijk} * C_{lj} \quad A_{il} = \sum_{j,k} B_{ijk} * C_{jl} * D_{kl}$$
$$A_{jl} = \sum_{i,k} B_{ijk} * C_{il} * D_{kl} \quad A_{ik} = \sum_j B_{ijk} * c_j \quad A_{ijl} = \sum_k B_{ijk} * C_{lk}$$
$$A_{ij} = B_{ij}(C_{ik} D_{kj}) \quad A_{ij} = B_{ij} \sum_k (C_{ik} D_{kj}) \quad A_{ij} = \sum_k B_{ijk} C_{ijk} + D_{ij} \quad A_{ij} = \sum_{k,l} C_{ijk} D_{ijk} \quad A_{ij} = \sum_{t,k} B_{tik} * C_{tj} * D_{kj} \quad A_{ij} = \sum_{k,l} B_{ikjl} * C_{kl}$$

Compilers can help...

Sparse Tensor Compilers Can Help...

Sparse tensor compilers compile high-level imperative descriptions of sparse programs to efficient implementations. We can **unify** the ecosystem by compiling different array interfaces to the **same expressive intermediate representation**, which is then handled by the compiler.

```python
import numpy as np
from scipy.sparse import csr_array

A = csr_array(
    [[1, 2], [0, 0], [4, 0]]
)
v = np.array([1, 0])

res = A @ v
```



```
A = Tensor(Dense(SparseList()))
x = Tensor(Dense())
y = Tensor(Dense())
function SpMV(A, x, y)
    y.= 0.0
    for i =_
        for j =_
            y[i] += A[i, j] * x[j]
        end
    end
    return y
end
```

Physical Layouts

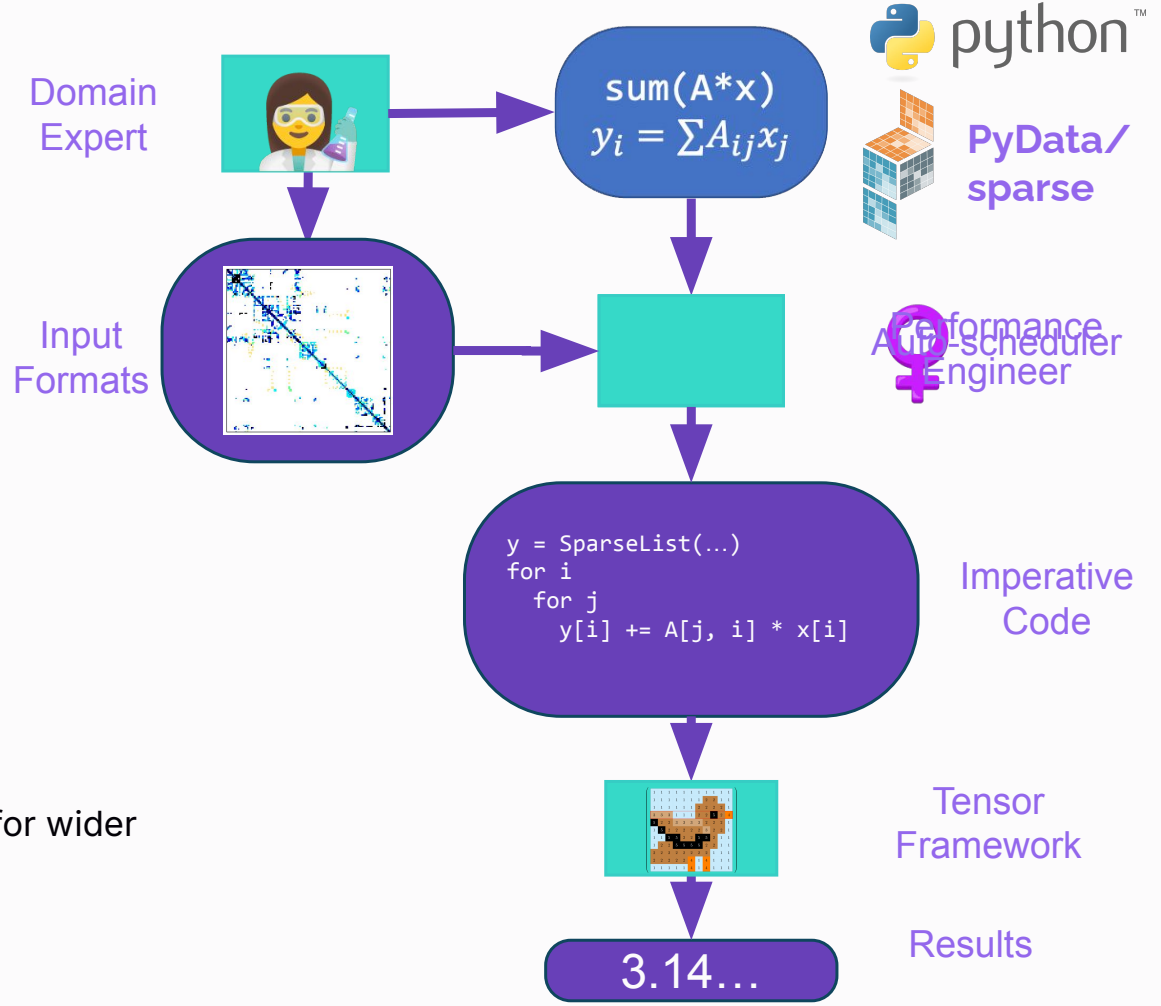
Loop Order

Pointwise Expression

Aggregate Function

High-Level Sparse Array Programming in Python

- Domain expert:
  - Writes Numpy Code
  - Writes Array Formats
- Optimization used to be manual
  - Loop Order Choices
  - Format Choices
  - Fusion Choices
- Python/Numpy interface is necessary for wider applicability



Agenda

- Project Overview
  - Array API standard
  - Finch compiler
- Compilation Example
- Use cases & benchmarks
  - HITS algorithm
  - CP Decomposition
- Ecosystem Overview
  - finch-tensor.org
  - https://github.com/GraphBLAS/binsparse-specification

Project Overview

Array API standard

Purpose: Standardize functionality that exists in most array/tensor libraries, including:

- Array manipulation, broadcasting, type promotion, indexing, sorting, statistical functions, ...
- Extensions: Fourier transform, linear algebra

- Allows for faster adoption of new backends in downstream libraries, such as SciPy and scikit-learn.
- Reduces inconsistencies in library APIs.
- Test suite for measuring compliance.

```python
import numpy as np

a, b = np.array(in1), np.array(in2)
res = np.dot(a, b.T) + const
a = np.reshape(a, newshape=shape)
c = np.unique(a, return_counts=True)
```

```python
import sparse

a, b = sparse.as_coo(in1), np.array(in2)
res = a @ b.transpose() + const
a = sparse.reshape(a, shape=shape)
c = sparse.unique_counts(a)
```

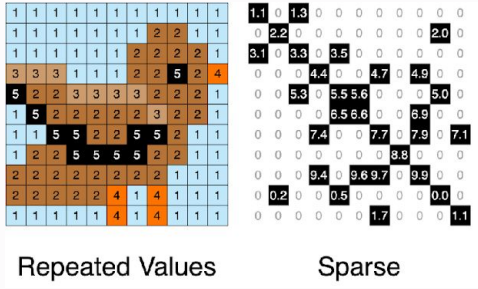
Array API

```python
import numpy as np
import jax.numpy as jnp
xp = np

a, b = xp.asarray(in1), xp.asarray(in2)
res = a @ b.mT + xp.asarray(const)
a = xp.reshape(a, shape=shape)
c = xp.unique_counts(a)
```

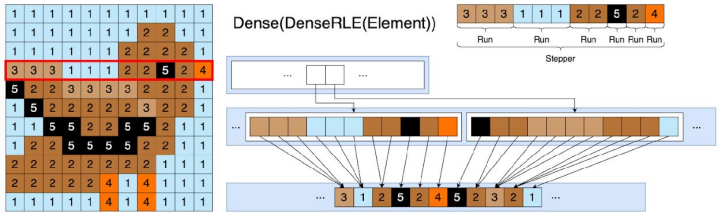
Finch compiler & Galley

Finch - compiler for sparse and structured multidimensional arrays. Specializes for control flow and different data structures.

Galley - an optimizer and scheduler for the array interface of Finch.jl.

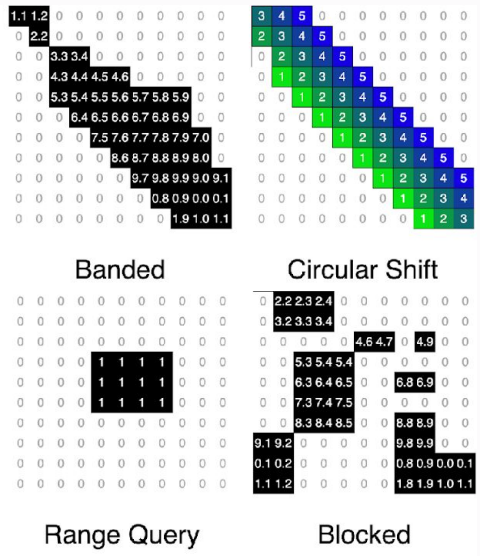


Structured Array Formats



Imperative Control Flow

```
@finch begin
    B .= 0
    for j = _
        w .= 0
        for k = _, i = _
            w[i] += A[i, k] * B[k, j]
        end
        for i = _
            C[i, j] = w[i]
        end
    end
end
```



Available from Python layer thanks to the auto-scheduler and Array API compatible finch-tensor library.

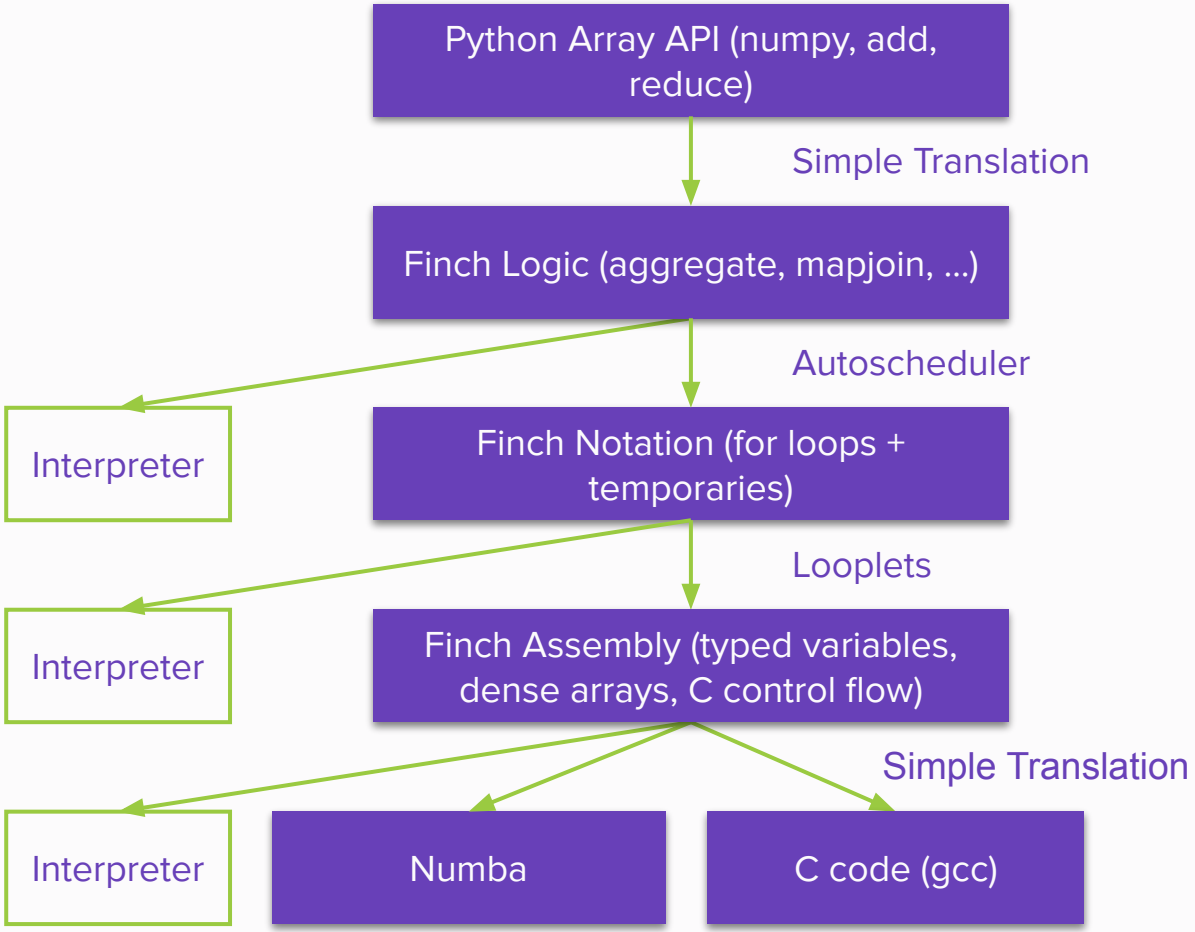
Finch Architecture

Finch Level Formats (proto)

Data Buffer Formats

Algebra/Rewriting Properties of Functions

Symbolic Compiler Utilities

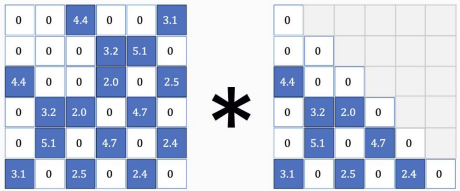


Compilation Example:

Lowering example

Python API - user facing code

```
arr1, arr2 = np.array(...), np.array(...)

f = dense(
  dense(
    element(0, float, intp, buffertype)
  )
)

tns1 = f(arr1)
tns2 = f(arr2)

tns1 = lazy(tns1)
tns2 = lazy(tril(tns2))

res = compute(tns1 * tns2)
```



Finch Logic

```
#A#0 = Table(
  FiberTensor(
    DenseLevel(DenseLevel(ElementLevel(NumpyBuffer(...))))
  ),
  ['#i#1', '#i#2']
)
#A#3 = Table(
  FiberTensor(
    DenseLevel(LoTriMask(DenseLevel(ElementLevel(NumpyBuffer(...))))))
  ),
  ['#i#4', '#i#5']
)

#A#8 = Reorder(
  MapJoin(
    mul,
    (
      Reorder(Relabel(#A#0, ['#i#6', '#i#7']), ['#i#6', '#i#7']),
      Reorder(Relabel(#A#3, ['#i#6', '#i#7']), ['#i#6', '#i#7']),
    )
  ),
  ['#i#6', '#i#7']
)
#A#9 = #A#8
return ('#A#9',)
```

Lowering example

Finch Logic

```
#A#0 = Table(FiberTensor(...))
#A#3 = Table(FiberTensor(...))

#A#8 = Reorder(
  MapJoin(
    mul,
    (
      Reorder(Relabel(#A#0, [...]), [...]),
      Reorder(Relabel(#A#3, [...]), [...]),
    )
  ),
  [...]
)
#A#9 = #A#8
return ('#A#9',)
```

Finch Notation

```
def func(
  A0: FiberTensorFType(...),
  A1: FiberTensorFType(...),
  A2: FiberTensorFType(...)
) -> FiberTensorFType(...):
    i0_size: ExtentFType(start=int64, end=int64) = dimension(A0, 0)
    i1_size: ExtentFType(start=int64, end=int64) = dimension(A0, 1)
    A0_slot = unpack(A0)
    A1_slot = unpack(A1)
    A2_slot = unpack(A2)
    declare(A2_slot, 0.0, InitWrite(0.0), ['i0_size', 'i1_size'])
    loop(i0, i0_size):
        loop(i1, i1_size):
            increment(
              update(A2_slot, ['i0', 'i1'], InitWrite(0.0)),
              mul(
                unwrap(read(A0_slot, ['i0', 'i1'])),
                unwrap(read(A1_slot, ['i0', 'i1'])),
              )
            )
    freeze(A2_slot, InitWrite(0.0))
    repack(A0_slot, A0)
    repack(A1_slot, A1)
    repack(A2_slot, A2)
    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    Declare(body)
    loop(loop(body))

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    loop(loop(body))

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    loop(loop(body))

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    Lookup(loop(body))

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        loop(body)

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        loop(body)

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        Sequence(head=Lookup(body), split=i0, tail=Lookup(body))

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        for i1 in range(i1_size.start, add(i0, 1)):
            Leaf(body)
        for i1_2 in range(add(i0, 1), i1_size.end):
            Leaf(body)

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        for i1 in range(i1_size.start, add(i0, 1)):
            i1__pos = add(i0__pos, mul(A2.lvl.lvl.stride, i1))
            i1__pos_2 = ...
            assert 0.0 == A2_buf[i1__pos]
            A2_buf[i1__pos] = mul(Access(body), Access(body))
        for i1_2 in range(add(i0, 1), i1_size.end):
            i1_2__pos = add(i0__pos, mul(A2_slot.lvl.lvl.stride, i1_2))
            i1_2__pos_2 = ...
            assert 0.0 == A2_buf[i1_2__pos]
            A2_buf[i1_2__pos] = mul(Access(body), Access(body))

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        for i1 in range(i1_size.start, add(i0, 1)):
            i1__pos = add(i0__pos, mul(A2.lvl.lvl.stride, i1))
            i1__pos_2 = ...
            assert 0.0 == A2_buf[i1__pos]
            A2_buf[i1__pos] = mul(Access(body), Access(body))
        for i1_2 in range(add(i0, 1), i1_size.end):
            i1_2__pos = add(i0__pos, mul(A2_slot.lvl.lvl.stride, i1_2))
            i1_2__pos_2 = ...
            assert 0.0 == A2_buf[i1_2__pos]
            A2_buf[i1_2__pos] = mul(Access(body), Access(body))

    return A2
```

Lowering example

Looplets

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        for i1 in range(i1_size.start, add(i0, 1)):
            i1__pos = add(i0__pos, mul(A2.lvl.lvl.stride, i1))
            i1__pos_2 = ...
            assert 0.0 == A2_buf[i1__pos]
            A2_buf[i1__pos] = mul(A0_buf[i1__pos_2], A1_buf[i1__pos_3])
        for i1_2 in range(add(i0, 1), i1_size.end):
            i1_2__pos = add(i0__pos, mul(A2_slot.lvl.lvl.stride, i1_2))
            i1_2__pos_2 = ...
            assert 0.0 == A2_buf[i1_2__pos]
            A2_buf[i1_2__pos] = mul(A0_buf[i1_2__pos_2], 0.0)

    return A2
```

Lowering example

Simplify

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        for i1 in range(i1_size.start, add(i0, 1)):
            i1__pos = add(i0__pos, mul(A2.lvl.lvl.stride, i1))
            i1__pos_2 = ...
            assert 0.0 == A2_buf[i1__pos]
            A2_buf[i1__pos] = mul(A0_buf[i1__pos_2], A1_buf[i1__pos_3])
        for i1_2 in range(add(i0, 1), i1_size.end):
            i1_2__pos = add(i0__pos, mul(A2_slot.lvl.lvl.stride, i1_2))
            i1_2__pos_2 = ...
            assert 0.0 == A2_buf[i1_2__pos]
            A2_buf[i1_2__pos] = 0.0

    return A2
```

Lowering example

Simplify

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        for i1 in range(i1_size.start, add(i0, 1)):
            i1__pos = add(i0__pos, mul(A2.lvl.lvl.stride, i1))
            i1__pos_2 = ...
            assert 0.0 == A2_buf[i1__pos]
            A2_buf[i1__pos] = mul(A0_buf[i1__pos_2], A1_buf[i1__pos_3])
        for i1_2 in range(add(i0, 1), i1_size.end):
            pass

    return A2
```

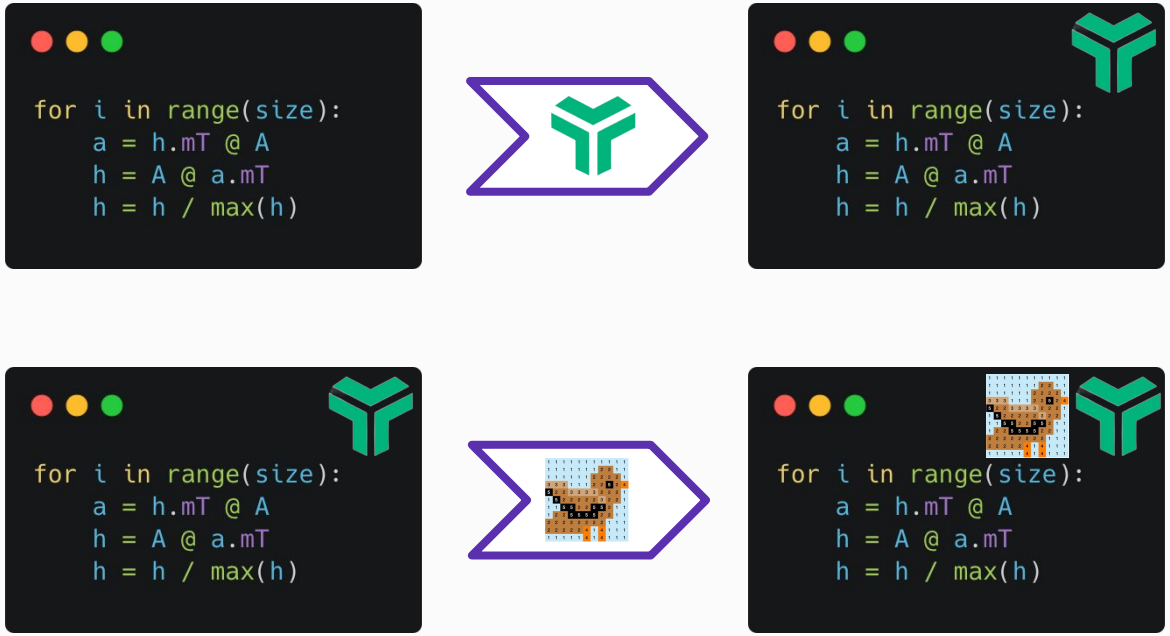
Lowering example

Simplify

```
@njit
def func(A0: Numba_FiberTensor, A1: Numba_FiberTensor, A2: Numba_FiberTensor):
    i0_size = Numba_Extent(0, A0.shape.element_0)
    i1_size = Numba_Extent(0, A0.shape.element_1)
    A0_buf = A0.lvl.lvl.lvl.val
    A1_buf = A1.lvl.lvl.mask.lvl.val
    A2_buf = A2.lvl.lvl.lvl.val

    for i in range(0, len(A2_buf)):
        A2_buf_arr[i] = 0.0
    for i0 in range(i0_size.start, i0_size.end):
        i0__pos = add(0, mul(A2.lvl.stride, i0))
        i0__pos_2: ...
        for i1 in range(i1_size.start, add(i0, 1)):
            i1__pos = add(i0__pos, mul(A2.lvl.lvl.stride, i1))
            i1__pos_2 = ...
            assert 0.0 == A2_buf[i1__pos]
            A2_buf[i1__pos] = mul(A0_buf[i1__pos_2], A1_buf[i1__pos_3])

    return A2
```

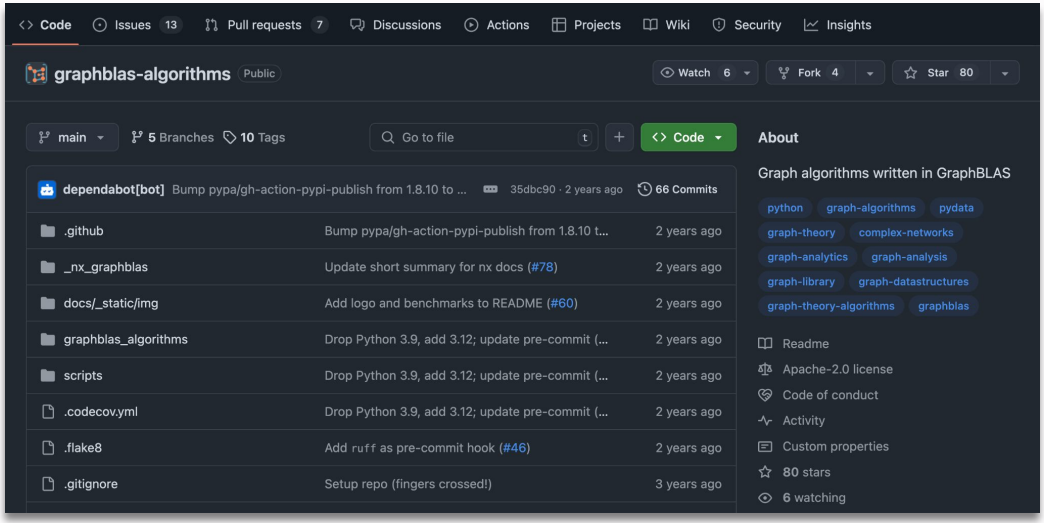
Use cases & benchmarks

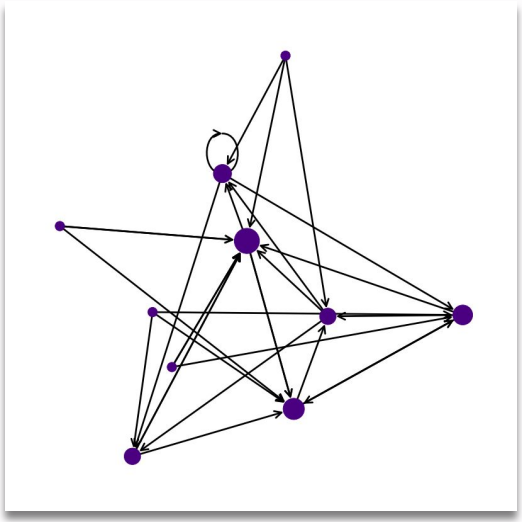
Use case workflow



Use case: HITS algorithm

Link analysis algorithm for estimating prominence in sparse networks.





github.com/python-graphblas/graphblas-algorithms

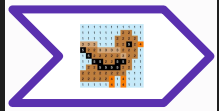
HITS algorithm: Array API conversion

```python
def hits(G, max_iter=100, tol=1.0e-8, normalized=True):
    N = len(G)
    h = Vector(float, N, name="h")
    a = Vector(float, N, name="a")
    h << 1.0 / N
    # Power iteration: make up to max_iter iterations
    A = G._A
    hprev = Vector(float, N, name="h_prev")
    for _i in range(max_iter):
        hprev, h = h, hprev
        a << hprev @ A
        h << A @ a
        h *= 1.0 / h.reduce(monoid.max).get(0)
        if is_converged(hprev, h, tol):
            break
    else:
        raise ConvergenceFailure(max_iter)
    if normalized:
        h *= 1.0 / h.reduce().get(0)
        a *= 1.0 / a.reduce().get(0)
    return h, a

def is_converged(xprev, x, tol):
    xprev << binary.minus(xprev | x)
    xprev << unary.abs(xprev)
    return xprev.reduce().get(0) < xprev.size * tol
```

```python
def hits(G, max_iter=100, tol=1.0e-8, normalized=True):
    N = G.N
    h = xp.full((N, 1), 1.0 / N)
    A = xp.asarray(G.A)
    # Power iteration: make up to max_iter iterations
    for _i in range(max_iter):
        hprev = h
        a = hprev.mT @ A
        h = A @ a.mT
        h = h / xp.max(h)
        if is_converged(hprev, h, N, tol):
            break
    else:
        raise Exception("Didn't converge")
    if normalized:
        h = h / xp.sum(xp.abs(h))
        a = a / xp.sum(xp.abs(a))
    return h, a

def is_converged(xprev, x, N, tol):
    err = xp.sum(xp.abs(x - xprev))
    return err < xp.asarray(N * tol)
```

HITS algorithm: complied with Finch

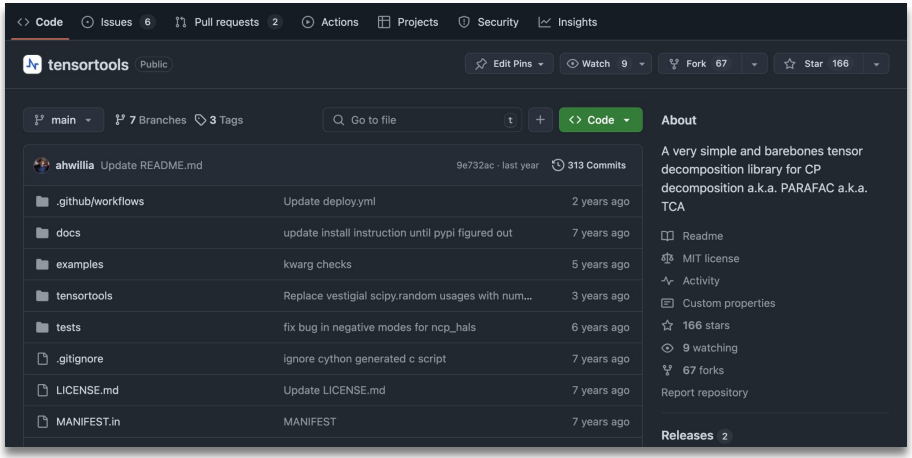
```python
def hits(G, max_iter=100, tol=1.0e-8, normalized=True):
    N = G.N
    h = xp.full((N, 1), 1.0 / N)
    A = xp.asarray(G.A)
    # Power iteration: make up to max_iter iterations
    for _i in range(max_iter):
        hprev = h
        a = hprev.mT @ A
        h = A @ a.mT
        h = h / xp.max(h)
        if is_converged(hprev, h, N, tol):
            break
    else:
        raise Exception("Didn't converge")
    if normalized:
        h = h / xp.sum(xp.abs(h))
        a = a / xp.sum(xp.abs(a))
    return h, a

def is_converged(xprev, x, N, tol):
    err = xp.sum(xp.abs(x - xprev))
    return err < xp.asarray(N * tol)
```

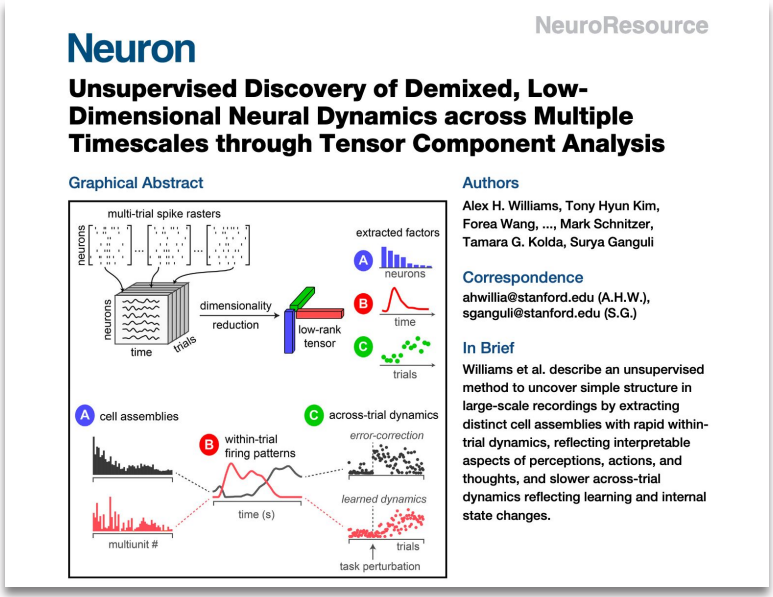


```python
def hits(G, max_iter=100, tol=1.0e-8, normalized=True):
    N = G.N
    h = xp.full((N, 1), 1.0 / N)
    A = xp.asarray(G.A)
    # Power iteration: make up to max_iter iterations
    for _i in range(max_iter):
        h, a, conv = kernel(h, A, N, tol)
        if conv:
            break
    else:
        raise Exception("Didn't converge")
    if normalized:
        h = h / xp.sum(xp.abs(h))
        a = a / xp.sum(xp.abs(a))
    return h, a

@sparse.compiled()
def kernel(hprev, A, N, tol):
    a = hprev.mT @ A
    h = A @ a.mT
    h = h / xp.max(h)
    conv = is_converged(hprev, h, N, tol)
    return h, a, conv

def is_converged(xprev, x, N, tol):
    err = xp.sum(xp.abs(x - xprev))
    return err < xp.asarray(N * tol)
```

Use case: CP Decomposition

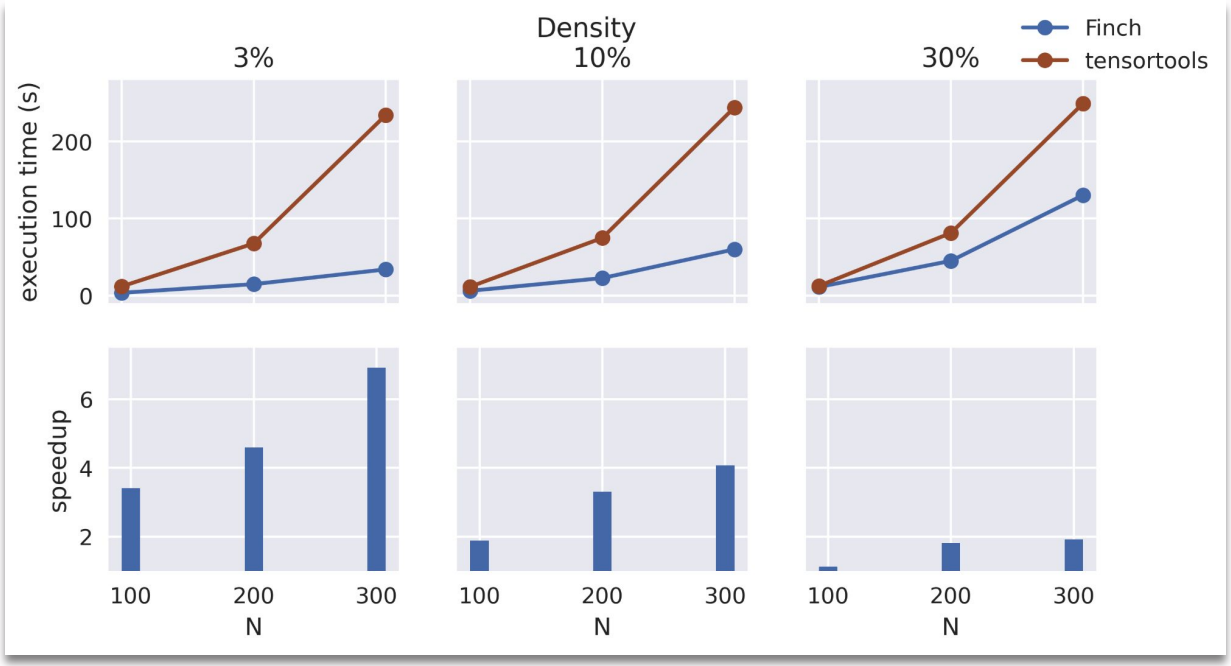
Central ingredient in learning probabilistic latent variables models.





github.com/neurostatslab/tensortools

CP Decomposition: accelerated computation



Benchmark setup:

- X and mask are NxNxN
- Density is: 30%, 10%, and 3%
- Sparse formats are CSF
- Finch has precompile run

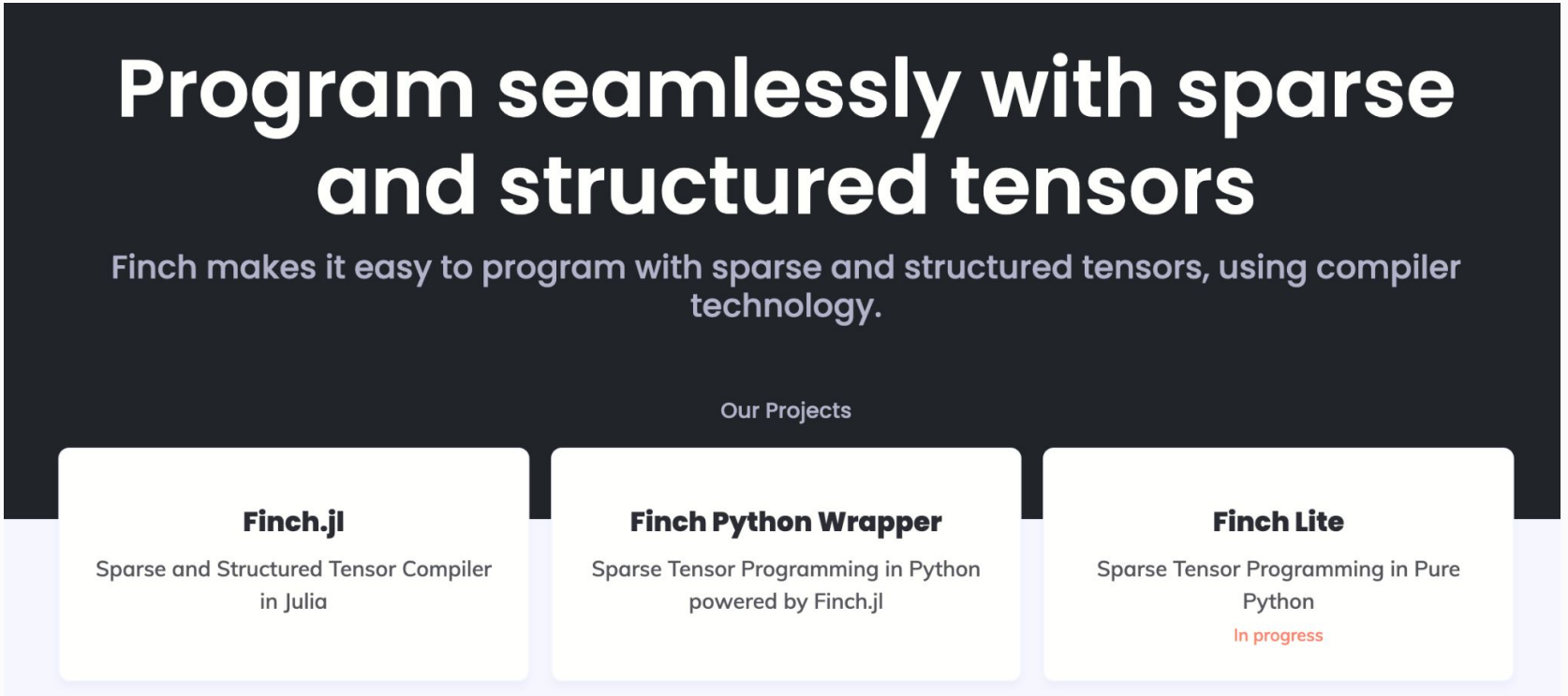
Hardware:

- Linux x86_64
- AMD Ryzen Threadripper 3970X 32-Core Processor

Try it out yourself!

github.com/finch-tensor/finch-tensor

Ecosystem Overview



finch-tensor.org

Ecosystem Integration

Binsparse File IO

Unify Ecosystem with Standardized Interchange Format

Pydata/Sparse Integration

Productionize the Finch compiler for distribution as part of Pydata/Sparse

binsparse: A binary sparse storage format

Existing formats are text-based:

- Slow to parse
- Ambiguous tensor/data formats

Goals for binsparse:

- Support all formats: COO, CSR, DCSR, tensors, etc.
- **Ideal world**: given **in-memory format**, dump it to disk **as is** in a **cross-platform** way.
- Efficient IO for sparse matrices and tensors.

A **binsparse file** has two parts:

- A **JSON descriptor** that describes the binary matrix format
- One or more **dense arrays** stored inside a **binary container** (HDF5, Zarr, etc.)

Binsparse can also be used as a specification for **zero-copy interchange** of sparse arrays between **compliant implementations**.

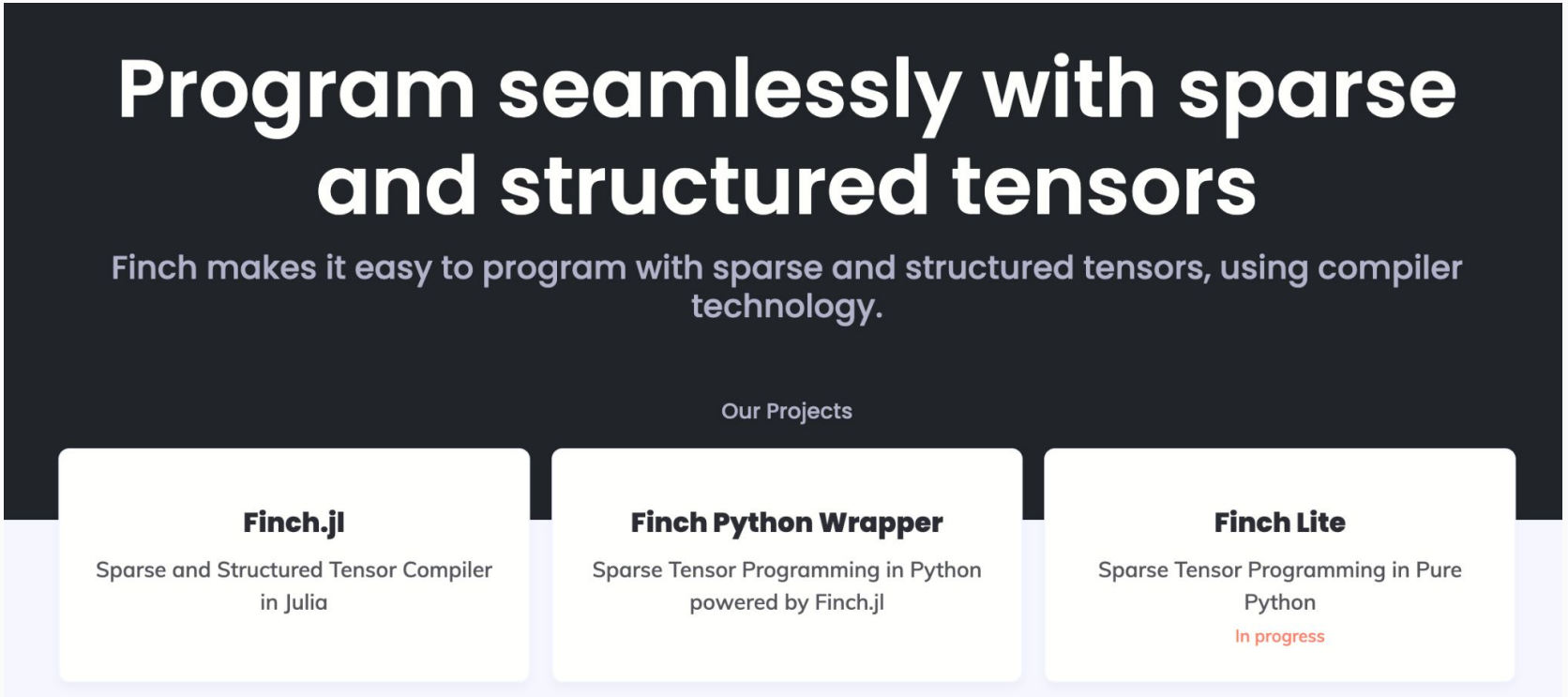
JSON binsparse descriptor

Format `"CSR"`
Dimensions `[5, 5]`
Symmetry, ISO-ness, etc.
Data Types

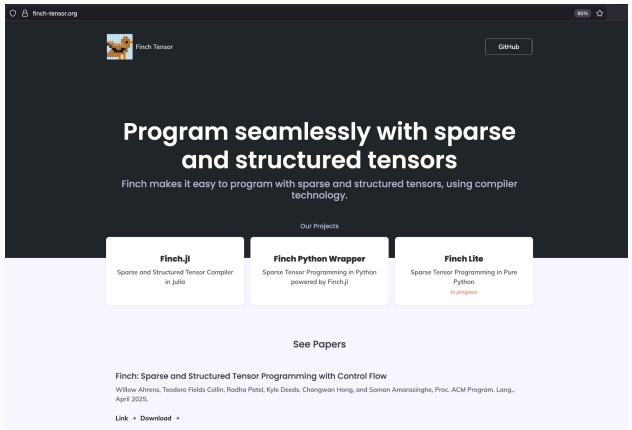
Binary Arrays (e.g. HDF5 Datasets)

pointers_to_1	0	1	3	3	5	6
indices_1	3	1	4	1	2	2
values	1	5	9	7	8	2

Questions?



finch-tensor.org



finch-tensor.org

Q&A