Legate Sparse

An Endeavoring Drop-In Replacement for SciPy Sparse on Many-GPU Machines

Rohan Yadav

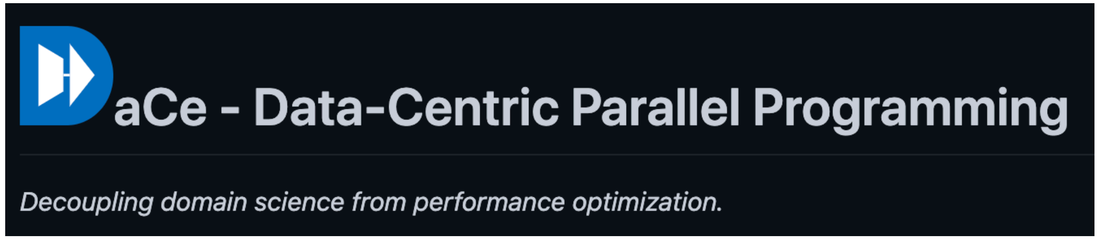
In collaboration with:
Michael Bauer, Michael Garland, Melih Elibol, Steven Dalton, Wonchan Lee, Manolis Papadakis

Distributed (dense) NumPy replacements are popular

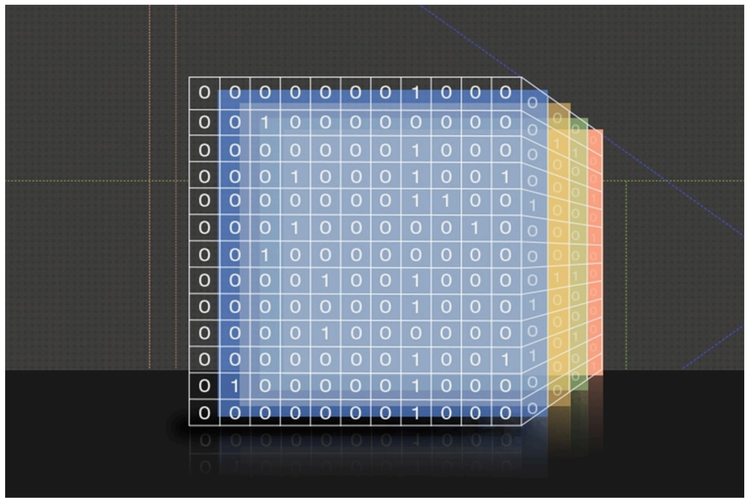
NVIDIA cuNumeric

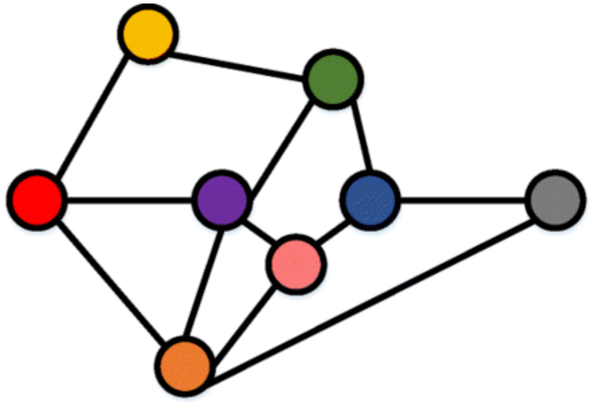
Bringing GPU-Accelerated Supercomputing to the NumPy Ecosystem

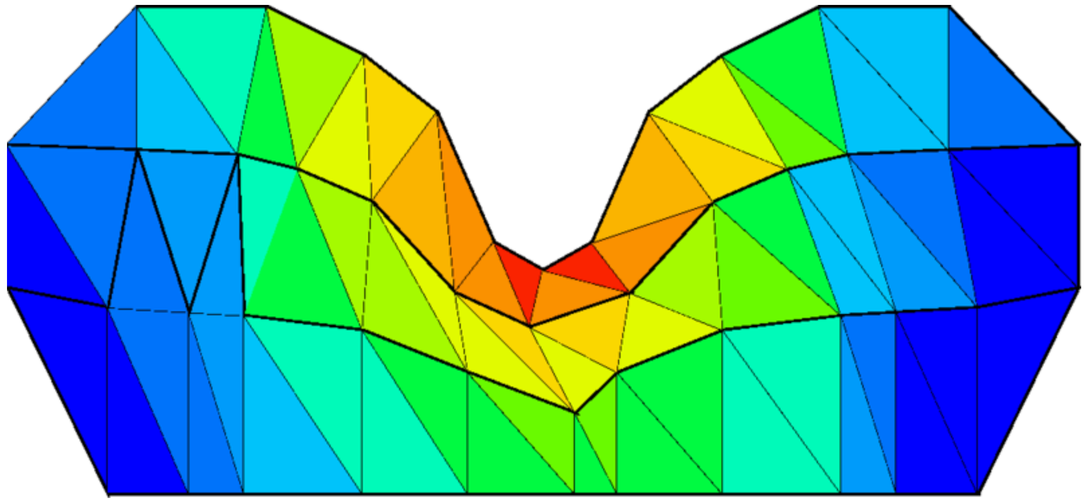
NumS



What about sparse computations?







It’s difficult!

Data dependent and irregular communication patterns

Requires heavy specialization to choice of sparse data structure

Load balancing

The talk in one slide

```python
import numpy as np
from scipy.sparse import diags, linalg

# Create the 2-D mesh.
x = np.linspace(xmin, xmax, nx)
y = np.linspace(ymin, ymax, ny)
X, Y = np.meshgrid(x, y, indexing='ij')

# Create the sparse stencil matrix.
a = 1.0 / dx**2
g = 1.0 / dy**2
c = -2.0*a - 2.0*g
diag_size = (nx - 2) * (ny - 2) - 1
first = np.full((nx - 3), a)
chunks = np.concatenate([[0.0], first])
diag_a = np.concatenate([first, np.tile(chunks, (diag_size - (nx - 3)) // (nx - 2))])
diag_g = g * np.ones((nx-2)*(ny-3))
diag_c = c * np.ones((nx-2)*(ny-2))
diagonals = [diag_g, diag_a, diag_c, diag_a, diag_g]
offsets = [-(nx-2), -1, 0, 1, nx-2]

# Define the solution vector.
b = (np.sin(np.pi*X)*np.cos(np.pi*Y)
     + np.sin(5.0*np.pi*X)*np.cos(5.0*np.pi*Y))
bflat = b[1:-1, 1:-1].flatten('F')

A = diags(diagonals, offsets, dtype=np.float64).tocsr()
p_sol = linalg.cg(A, bflat, tol=1e-10)
assert(np.allclose((A @ p_sol), bflat))
```

Legate Sparse: Distributed Drop-in `scipy.sparse`

Common matrix formats: CSR, CSC, COO, DIA

Arithmetic and linear algebra operations

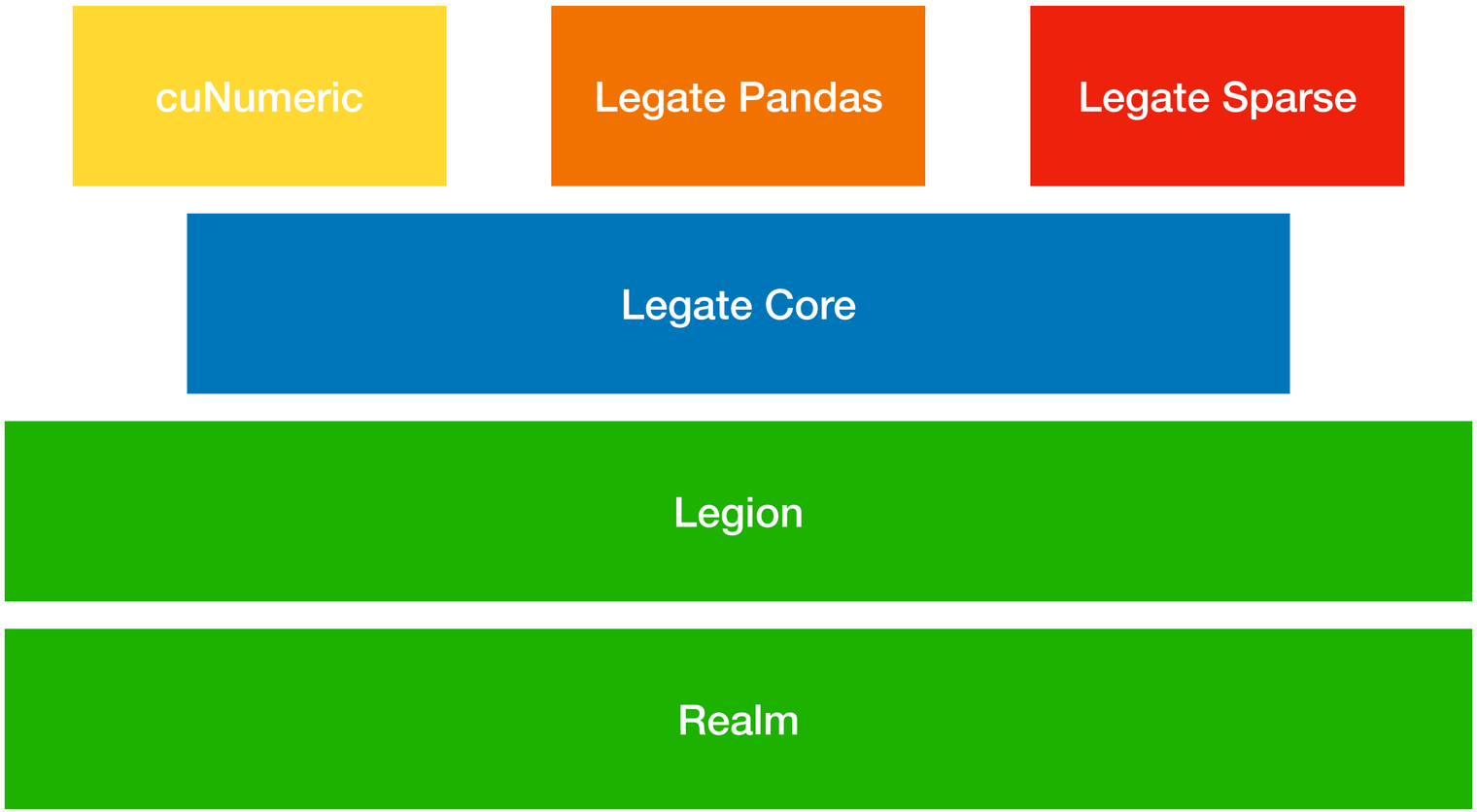
Fast (and parallel) format conversions

Parallel sparse matrix construction utilities

Prebuilt iterative solvers (`scipy.sparse.linalg`)

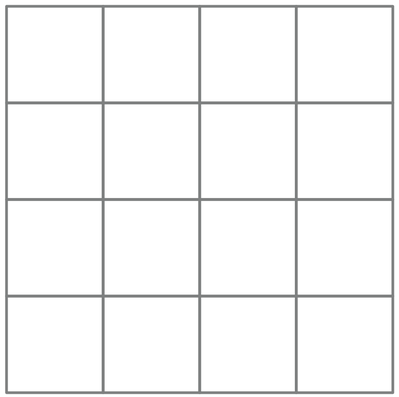
Contact us with use cases!

https://gitlab-master.nvidia.com/legate/legate.sparse


cuNumeric
Legate Pandas
Legate Sparse
Legate Core
Legion
Realm

Legate API

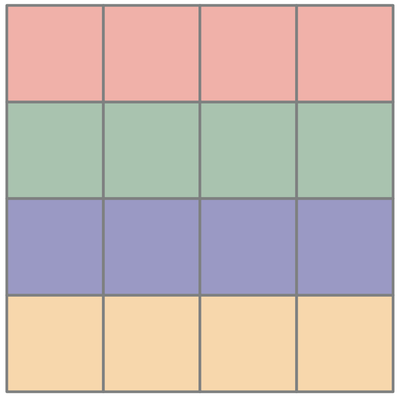
Stores



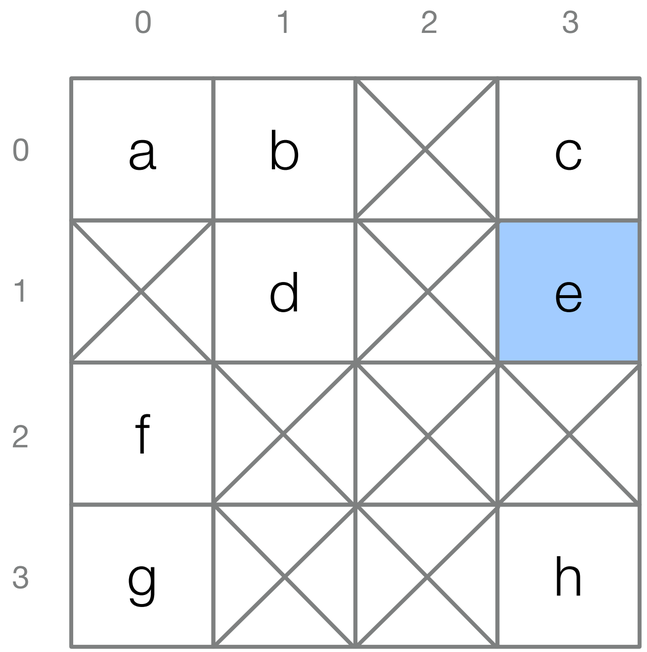
Tasks

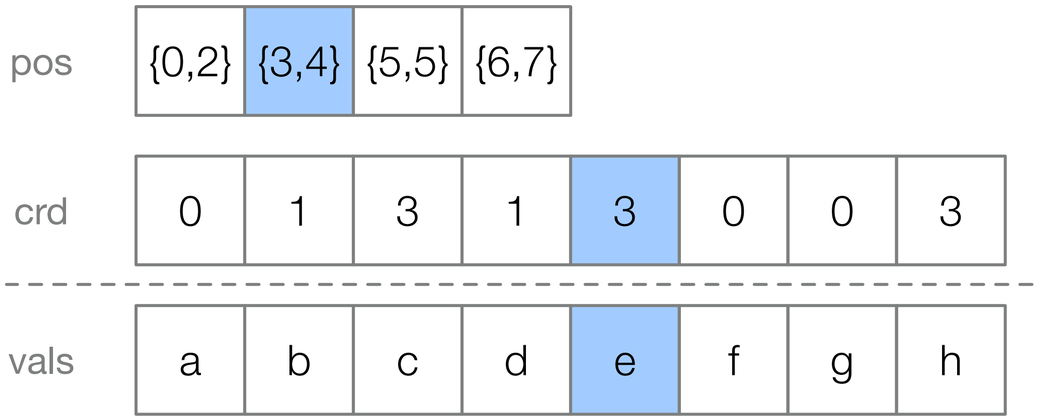
Arbitrary C/C++/CUDA

Partitions

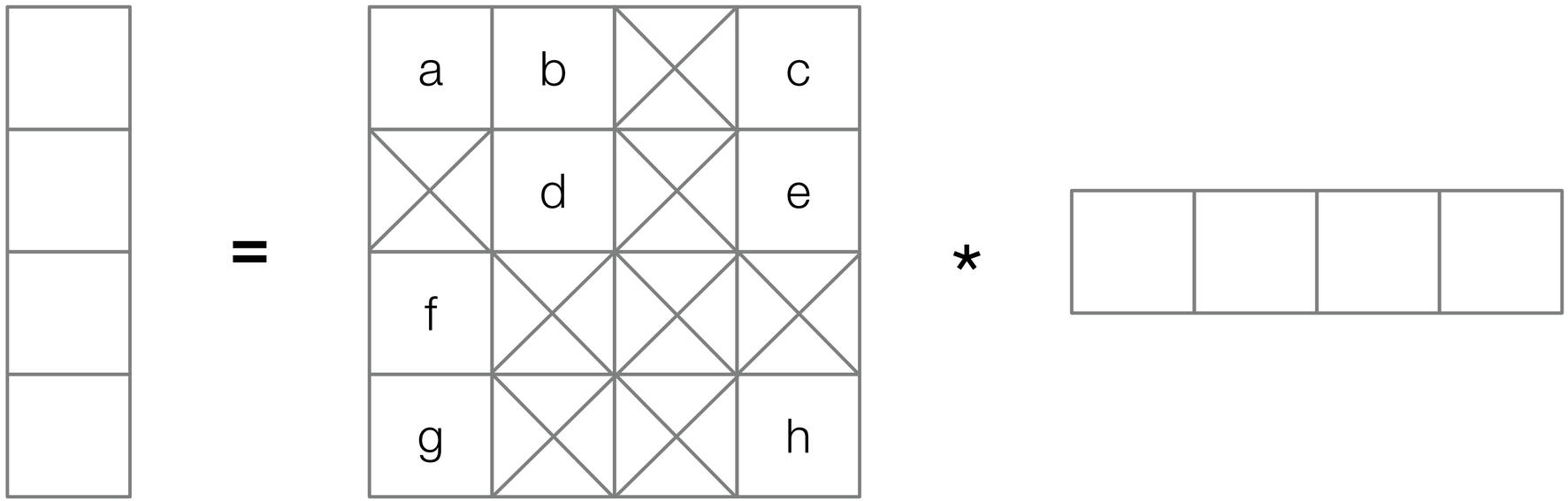


Data Representation (CSR)

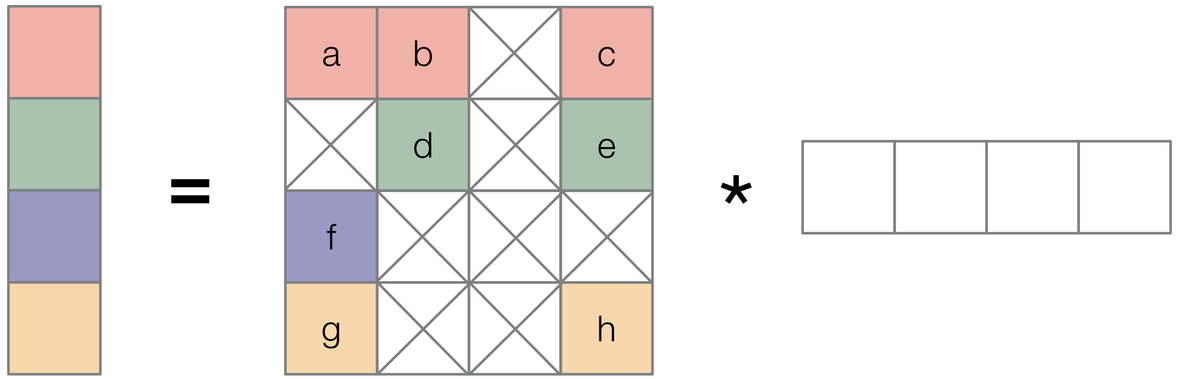


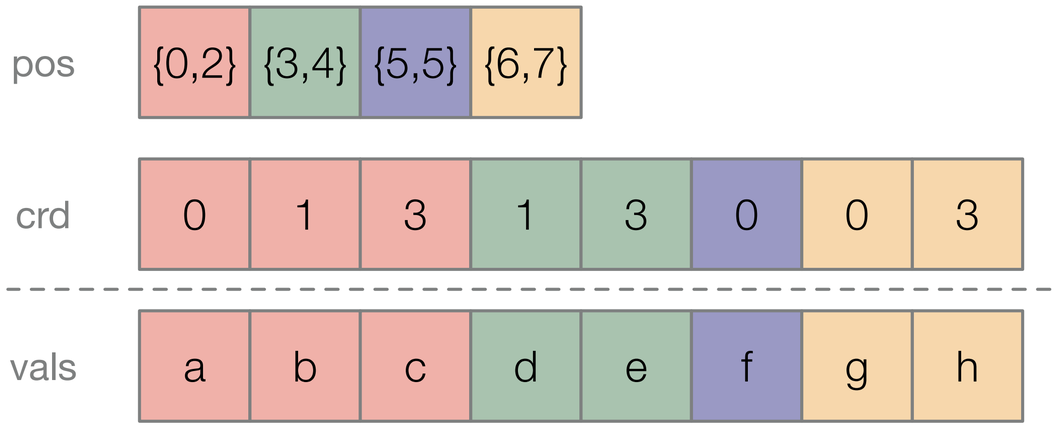


Example: Distributed SpMV

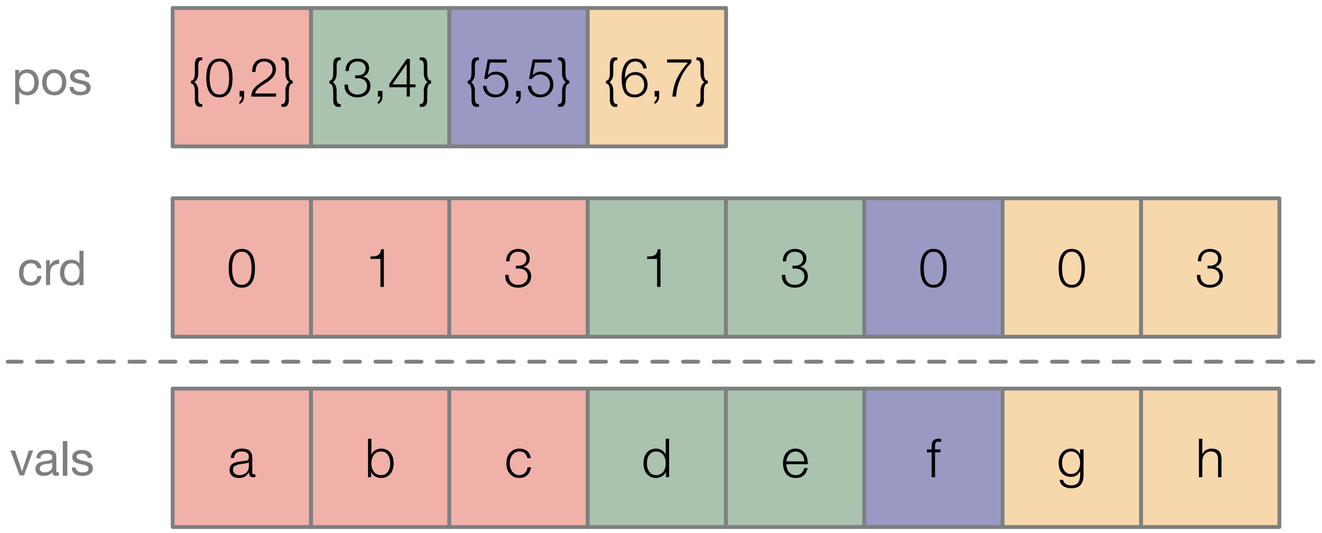


Example: Distributed SpMV

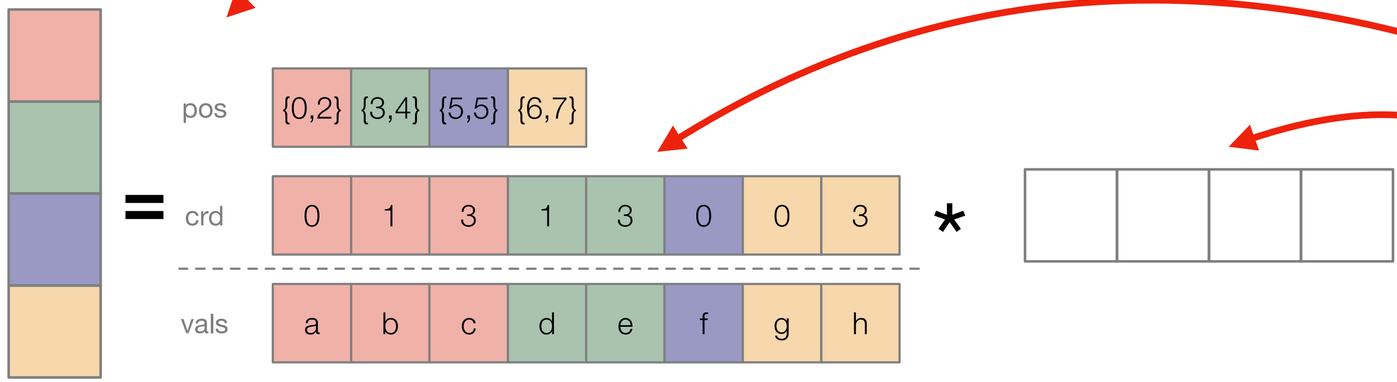




Dependent Partitioning: Images



Example: Distributed SpMV



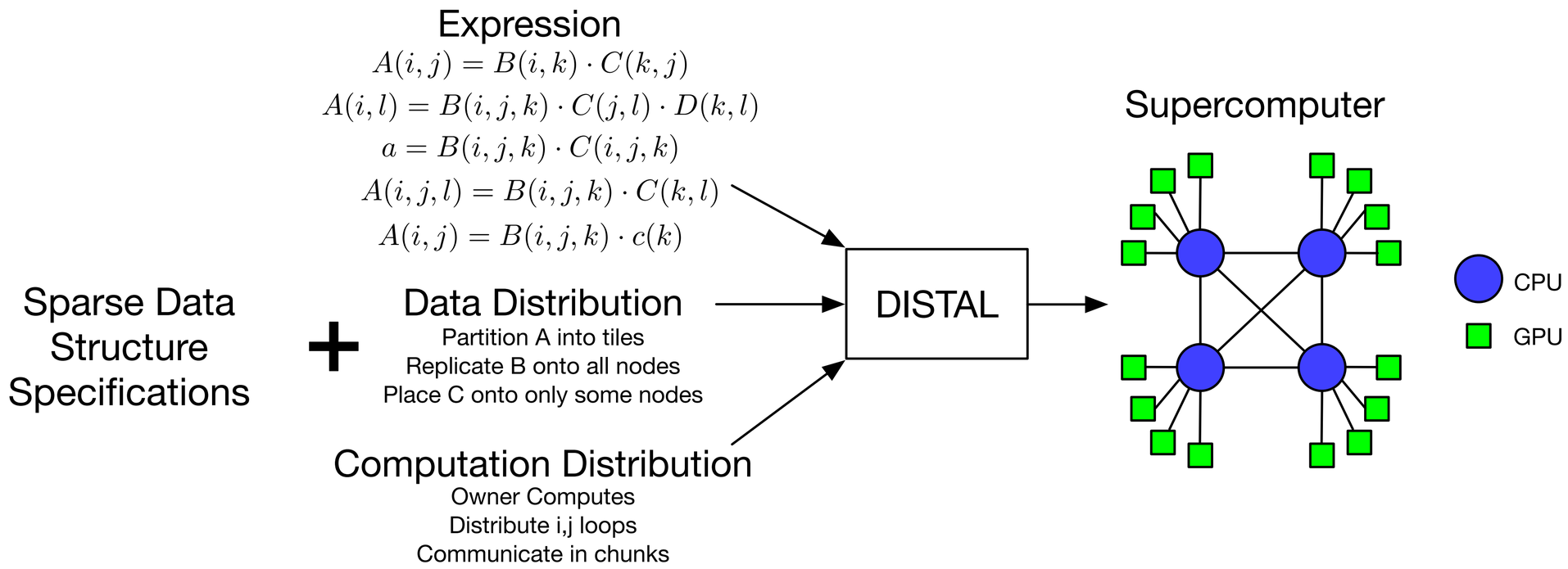

```
task = ctx.create_task(CSR_SPMV_ROW_SPLIT)
task.add_output(y)
task.add_input(self.pos)
task.add_input(self.crd)
task.add_input(self.vals)
task.add_input(x)
task.add_alignment(y, self.pos)
task.add_image_constraint(self.pos, self.crd)
task.add_alignment(self.crd, self.vals)
task.add_broadcast(x)
task.execute()
```

```
void SpMV(bounds...) {
  for (size_t i = bounds.lo()[0]; i <= bounds.hi()[0]; i++) {
    // We importantly need to discard whatever data already lives in the instances.
    val_ty sum = 0.0;
    for (size_t jpos = posacc[i].lo; jpos <= posacc[i].hi; jpos++) {
      auto j = crdacc[jpos];
      sum += valsacc[jpos] * xacc[j];
    }
    yacc[i] = sum;
  }
}
```

Distributed SpMV on *a single-threaded CPU* on *CSR matrices* with a *row-based algorithm* wasn't too hard.

How do we scale to other architectures, formats and algorithms?

My PhD Research: DISTAL



Legate Sparse Implementation

For “Tensor Algebra”-like computations, utilize DISTAL

MatVec, Matrix-Matrix Multiplication, Addition, etc.

Easily extend beyond scipy, providing higher-order operations like SDDMM

Leverage cuNumeric + hand-written code for auxiliary operations

Performance Results

Experiments performed on Selene:

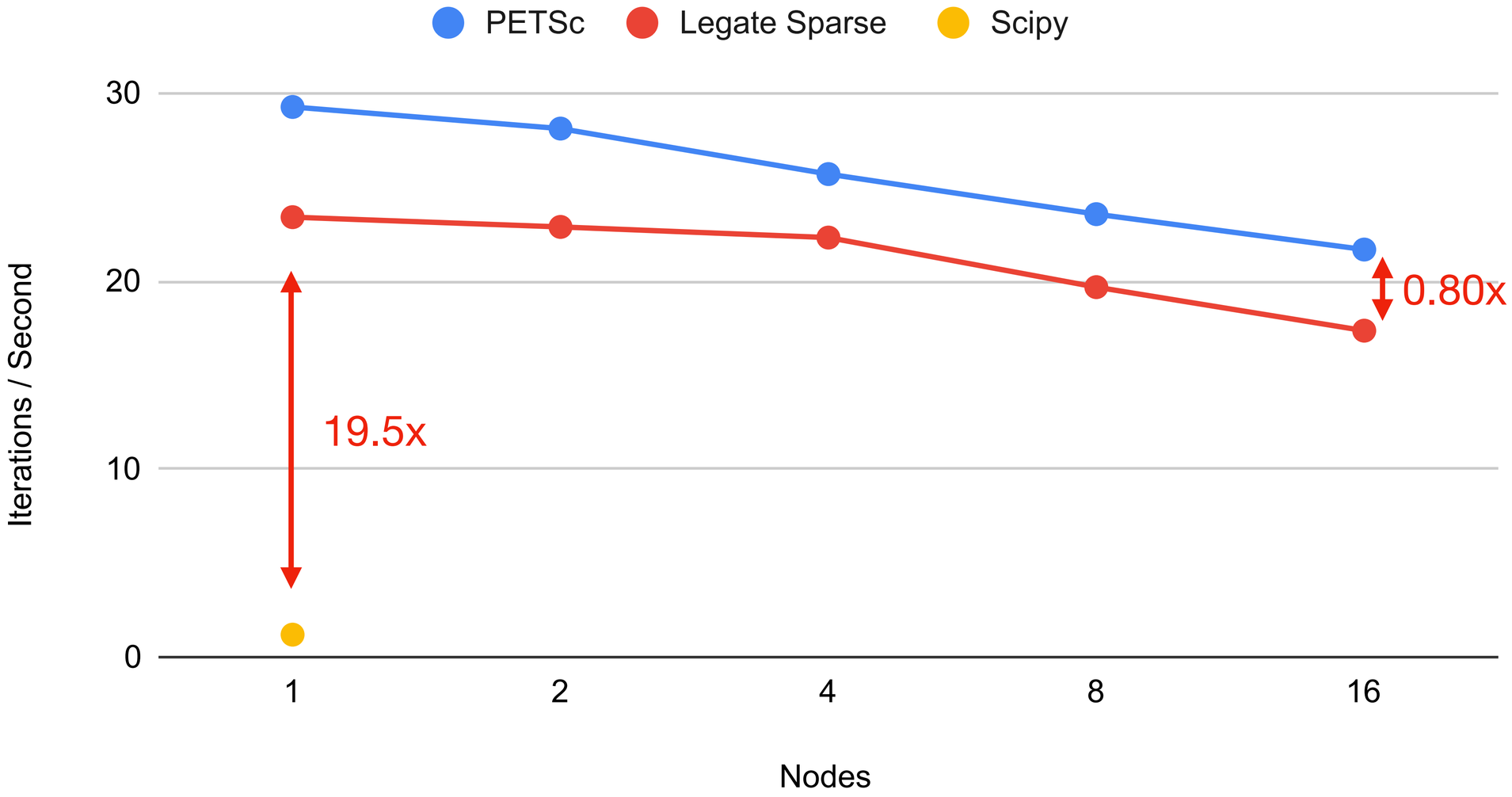
128 CPU Cores / Node
8 A100 GPUs / Node

Performance Results — PDE Solver

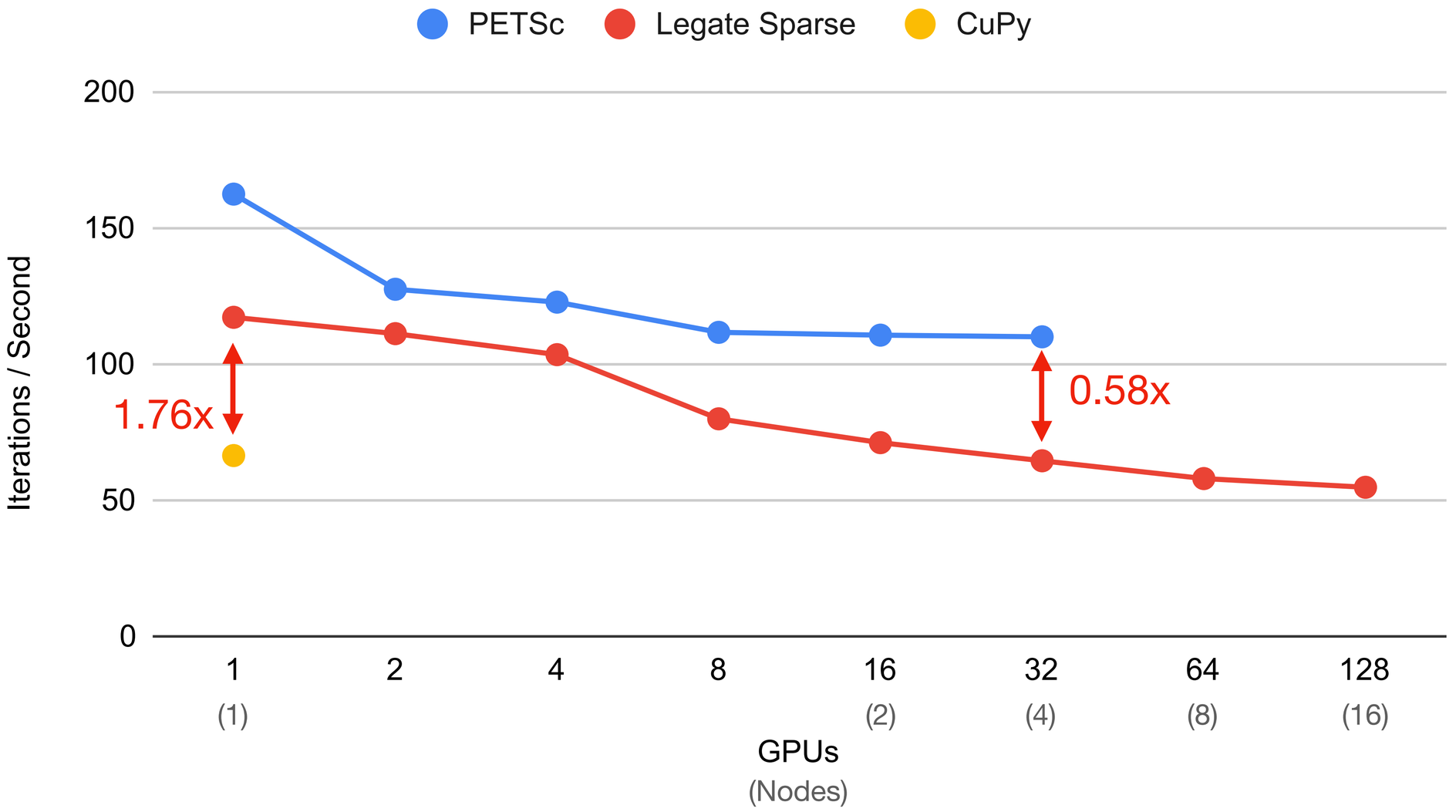
```python
import numpy as np
from scipy.sparse import diags, linalg

# Create the 2-D mesh.
x = np.linspace(xmin, xmax, nx)
y = np.linspace(ymin, ymax, ny)
X, Y = np.meshgrid(x, y, indexing='ij')

# Create the sparse stencil matrix.
a = 1.0 / dx**2
g = 1.0 / dy**2
c = -2.0*a - 2.0*g
diag_size = (nx - 2) * (ny - 2) - 1
first = np.full((nx - 3), a)
chunks = np.concatenate([[0.0], first])
diag_a = np.concatenate([first, np.tile(chunks, (diag_size - (nx - 3)) // (nx - 2))])
diag_g = g * np.ones((nx-2)*(ny-3))
diag_c = c * np.ones((nx-2)*(ny-2))
diagonals = [diag_g, diag_a, diag_c, diag_a, diag_g]
offsets = [-(nx-2), -1, 0, 1, nx-2]

# Define the solution vector.
b = (np.sin(np.pi*X)*np.cos(np.pi*Y)
     + np.sin(5.0*np.pi*X)*np.cos(5.0*np.pi*Y))
bflat = b[1:-1, 1:-1].flatten('F')

A = diags(diagonals, offsets, dtype=np.float64).tocsr()
p_sol = linalg.cg(A, bflat, tol=1e-10)
assert(np.allclose((A @ p_sol), bflat))
```

https://aquaulb.github.io/book_solving_pde_mooc/solving_pde_mooc/notebooks/05_IterativeMethods/05_01_Iteration_and_2D.html

CG Solver CPU Weak Scaling Throughput



CG Solver GPU Weak Scaling Throughput



Performance Results — Word Movers Distance

```python
def sinkhorn_wmd(r, c, vecs, lamb, max_iter):
    sel = r.squeeze() > 0
    r = r[sel].reshape(-1, 1).astype(numpy.float64)
    M = cdist(vecs[sel], vecs).astype(numpy.float64)
    a_dim = r.shape[0]
    b_nobs = c.shape[1]
    x = np.ones((a_dim, b_nobs)) / a_dim
    K = np.exp(-M * lamb)
    it = 0
    while it < max_iter:
        u = 1.0 / x
        v = c * (1 / K.T @ u)
        x = (1 / r) * K @ v
        it += 1
    u = 1.0 / x
    v = c * (1 / K.T @ u)
    result = (u * ((K * M) @ v)).sum(axis=0)
    return result
```

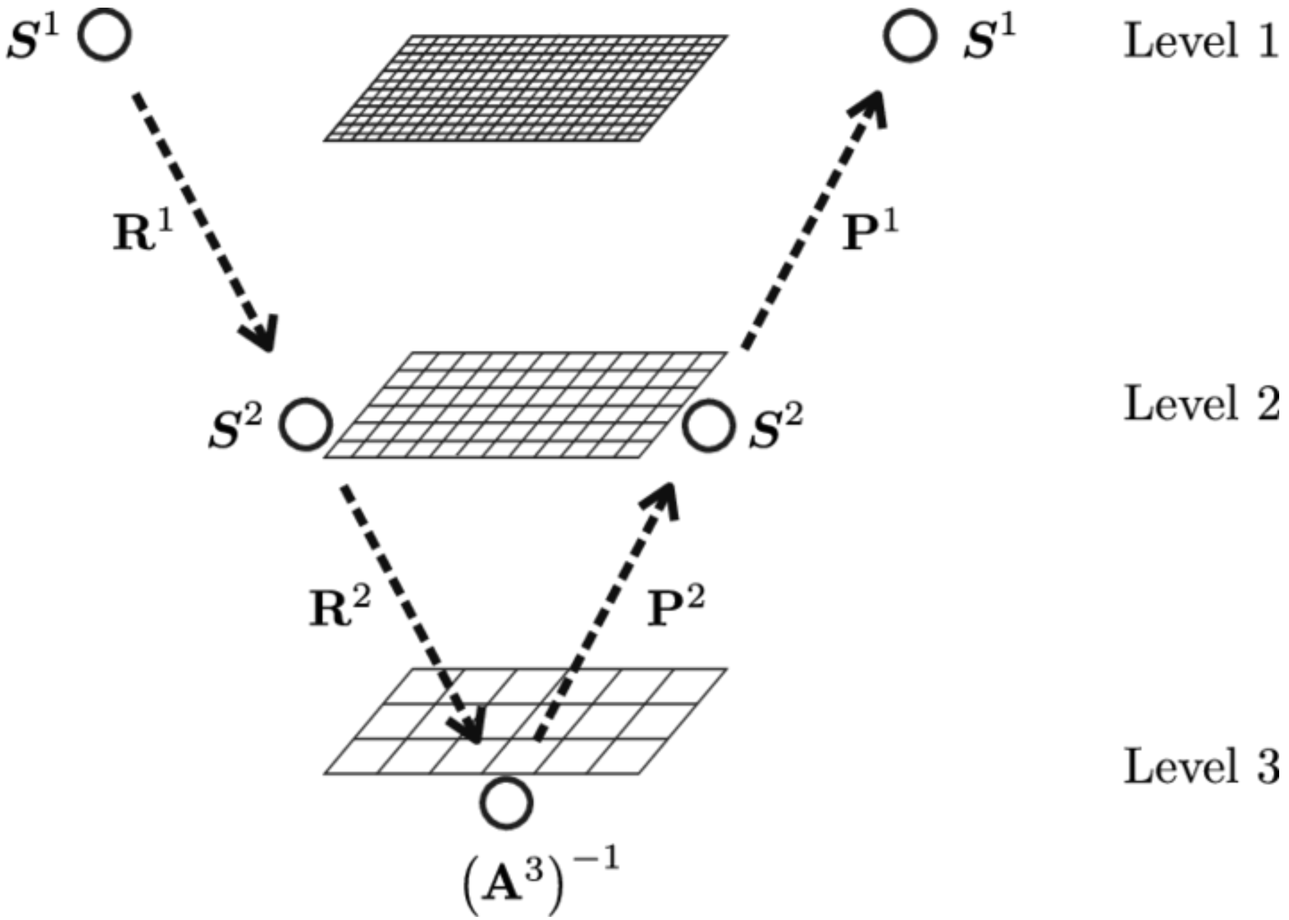
Handwritten C Baseline (500 LOC): 1835ms

Legate Sparse CPU: 1463ms

Legate Sparse 1 GPU: 233.8ms

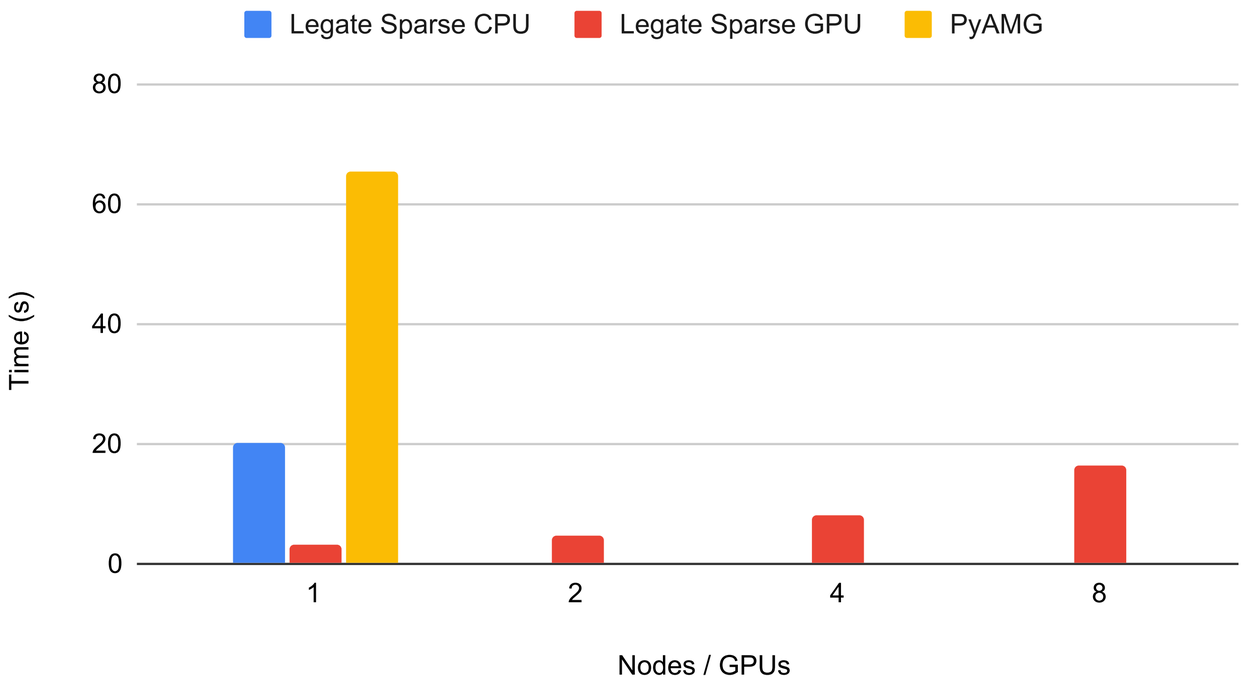
https://github.com/IntelLabs/Optimized-Implementation-of-Word-Movers-Distance

Performance Results — Algebraic MultiGrid Solver

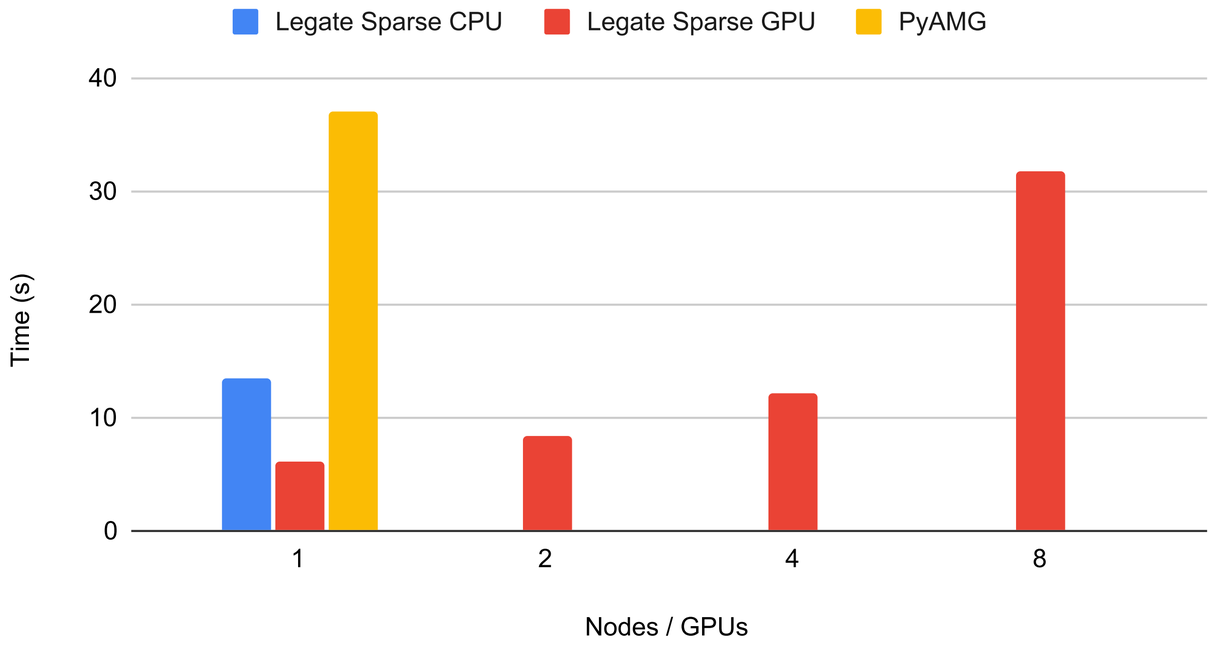



```python
# Define a custom preconditioner that cycles through the levels.
def matvec(b, out=None):
    if out is None:
        out = np.zeros_like(b)
    else:
        out.fill(0.0)
    cycle(levels, 0, out, b)
    return out
M = linalg.LinearOperator(A.shape, matvec=matvec)

# x, _ = linalg.cg(A, b=b)
x, _ = linalg.cg(A, b=b, M=M)
```

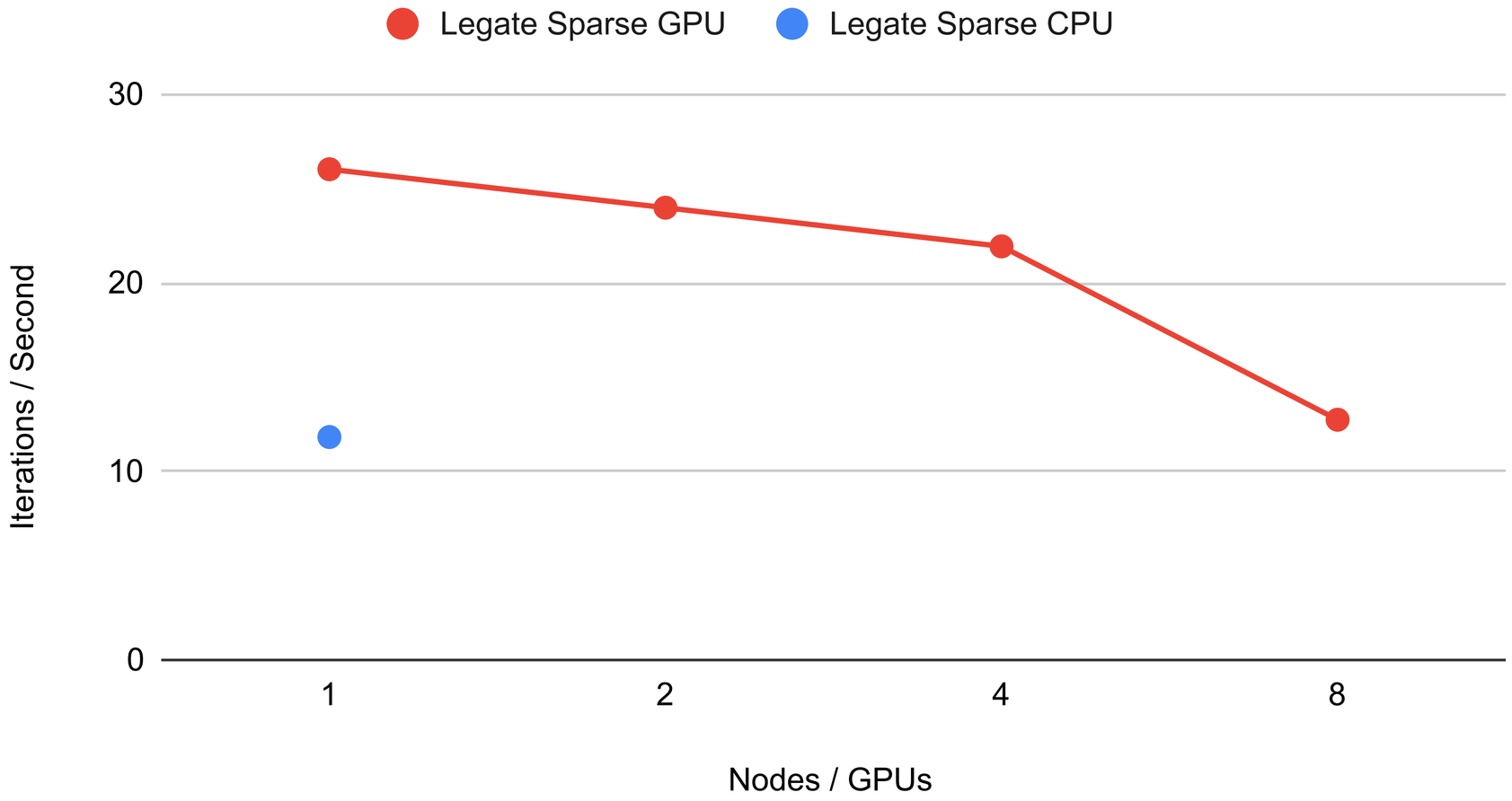
AMG Build Time



AMG Solve Time



AMG Solver Throughput



Conclusion

Legate Sparse is a distributed and accelerated replacement for `scipy.sparse`

Legate Sparse achieves competitive performance with hand-tuned systems

Future Work:

Developing and scaling applications in new domains
Automatically find good (balanced) partitions
Adapt algorithmic choices to sparse data structure

Contact: rohany@cs.stanford.edu https://gitlab-master.nvidia.com/legate/legate.sparse