Implementing Graph Analytics with Python and Numba

Siu Kwan Lam
Continuum Analytics

Python for Exploratory Programming

- **Flexibility**:
  - interpreted
  - duck typing

- **Interactivity**:
  - interactive shell,
  - IPython Notebook

- **Rich Libraries**:
  - math: numpy, scipy
  - machine learning: scikit-learn, theano
  - visualization: bokeh, matplotlib

- **Performance**:
  - numba

Numba

- Python JIT
- Targets: x86, CUDA

```
@cuda.jit(device=True)
def cuda_random_walk_per_node(curnode, visits, colidx, edges, resetprob,
                              randstates):
    tid = cuda.threadIdx.x
    randnum = cuda_xorshift_float(randstates, tid)
    if randnum >= resetprob:
        base = colidx[curnode]
        offset = colidx[curnode + 1]
        # If the edge list is non-empty
        if offset - base > 0:
            # Pick a random destination
            randint = cuda_xorshift(randstates, tid)
            randdestid = (uint64(randint % uint64(offset - base)) +
                          uint64(base))
            dest = edges[randdestid]
        else:
            # Random restart
            randint = cuda_xorshift(randstates, tid)
            randdestid = randint % uint64(visits.size)
            dest = randdestid

        # Increment visit count
        cuda.atomic.add(visits, dest, 1)
```

Numba

- Python JIT
- Targets: x86, CUDA

```
@cuda.jit(device=True)
def cuda_random_walk_per_node(  rnode, visits, colidx, edges, resetprob,
                              randstates):
    tid = cuda.threadIdx.x
    randnum = cuda_xorshift_float(randstates, tid)
    if randnum >= resetprob:
        base = colidx[curnode]
        offset = colidx[curnode + 1]
        # If the edge list is non-empty
        if offset - base > 0:
            # Pick a random destination
            randint = cuda_xorshift(randstates, tid)
            randdestid = (uint64(randint % uint64(offset - base)) +
                          uint64(base))
            dest = edges[randdestid]
        else:
            # Random restart
            randint = cuda_xorshift(randstates, tid)
            randdestid = randint % uint64(visits.size)
            dest = randdestid

        # Increment visit count
        cuda.atomic.add(visits, dest, 1)
```

Simply add a @cuda.jit decorator

Numba

- Python JIT
- Targets: x86, CUDA

```
@cuda.jit(device=True)
def cuda_random_walk_per_node(curnode, visits, colidx, edges, resetprob,
                              randstates):
    tid = cuda.threadIdx.x
    randnum = cuda_xorshift_float(randstates, tid)
    if randnum >= resetprob:
        base = colidx[curnode]
        offset = colidx[curnode + 1]
        # If the edge list is non-empty
        if offset - base > 0:
            # Pick a random destination
            randint = cuda_xorshift(randstates, tid)
            randdestid = (uint64(randint % uint64(offset - base)) +
                          uint64(base))
            dest = edges[randdestid]
        else:
            # Random restart
            randint = cuda_xorshift(randstates, tid)
            randdestid = randint % uint64(visits.size)
            dest = randdestid

        # Increment visit count
        cuda.atomic.add(visits, dest, 1)
```

CUDA special register
cuda.threadIdx.x

NumbaPro

- Commercial extension to Numba
- Key CUDA Features:
  - Sort functions
  - Bindings to cuRAND, cuBLAS, cuSPARSE, cuFFT
  - Reduction kernel builder
  - Array function builder (NumPy universal function)

A Case Study on Large Graph Problems

- WebDataCommon 2012 PayLevelDomain Hyperlinks Graph
- 623 million edges
- 43 million nodes
- Find communities by densest k-subgraph
- 3GB compressed text data

Reference:
http://webdatacommons.org/hyperlinkgraph/2012-08/download.

Densest k-SubGraph (DkS)

Finding a subgraph on k-nodes with the largest average degree

In the context of WebGraph:
    Finding a group of k-domains with the largest average number of links.

NP-hard by reduction to MAXCLIQUE

Reference:
Papailiopoulos, Dimitris, et al. "Finding dense subgraphs via low-rank bilinear optimization." Pro

Our Approach

Approximate DkS with low-rank bilinear optimization (Papailiopoulos, et al)

1. Low-rank approximation with eigen-decomposition (slowest in CPU code)
2. Bilinear optimization with Spannogram

Reference:

Papailiopoulos, Dimitris, et al. "Finding dense subgraphs via low-rank bilinear optimization." Pro

Hardware

- Intel Core 2 Duo
- 30 GB Memory
- 5GB Tesla K20C

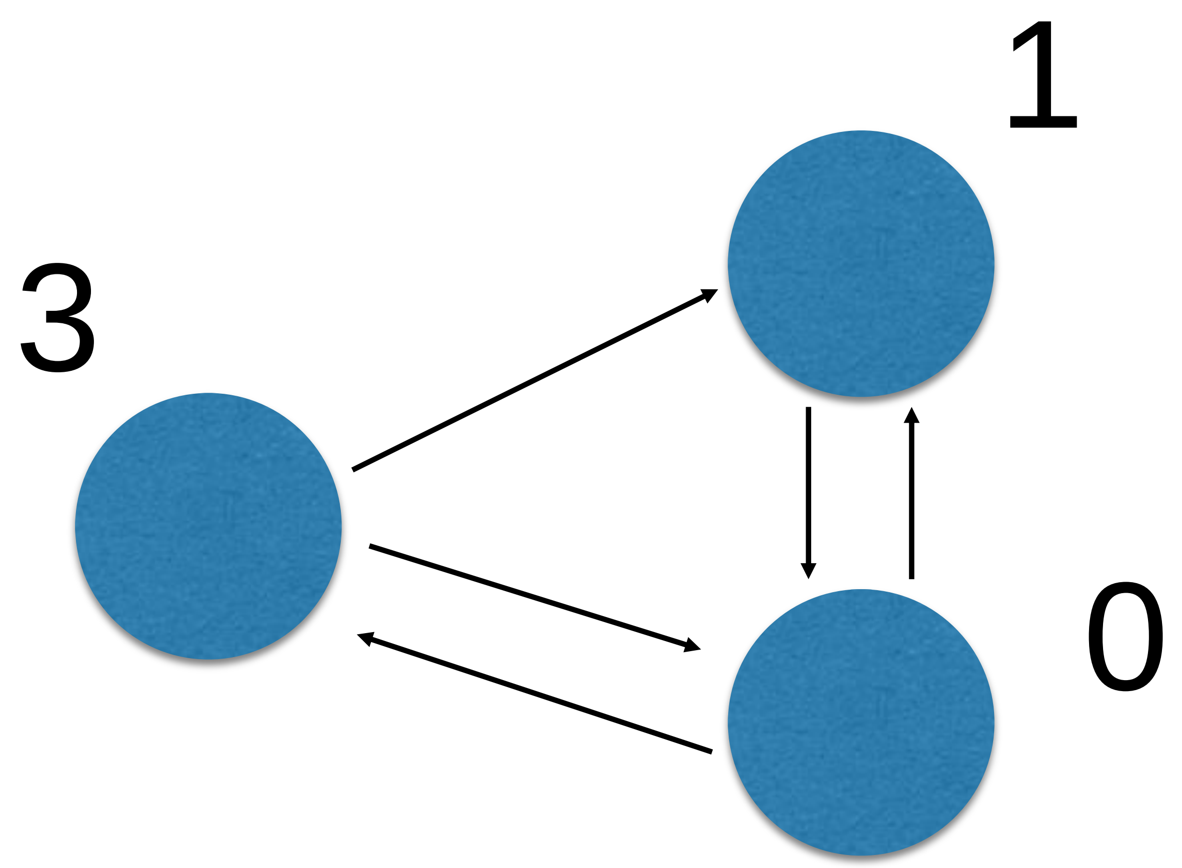
Eigen-Decomposition

- Largest eigenvector == PageRank
- First Attempt: Power iteration
  - Too slow due to high global memory traffic
  - Implemented as out-of-core algorithm
  - Took 15hrs (with I/O time)
- Second Attempt: Random Walk PageRank
  - From a distributed algorithm with low communication overhead (see reference)
  - Simple memory access pattern
  - Simplified storage fits on GPU

Reference:
Sarma, Atish Das, et al. "Fast distributed PageRank computation." Theoretical Computer Scie

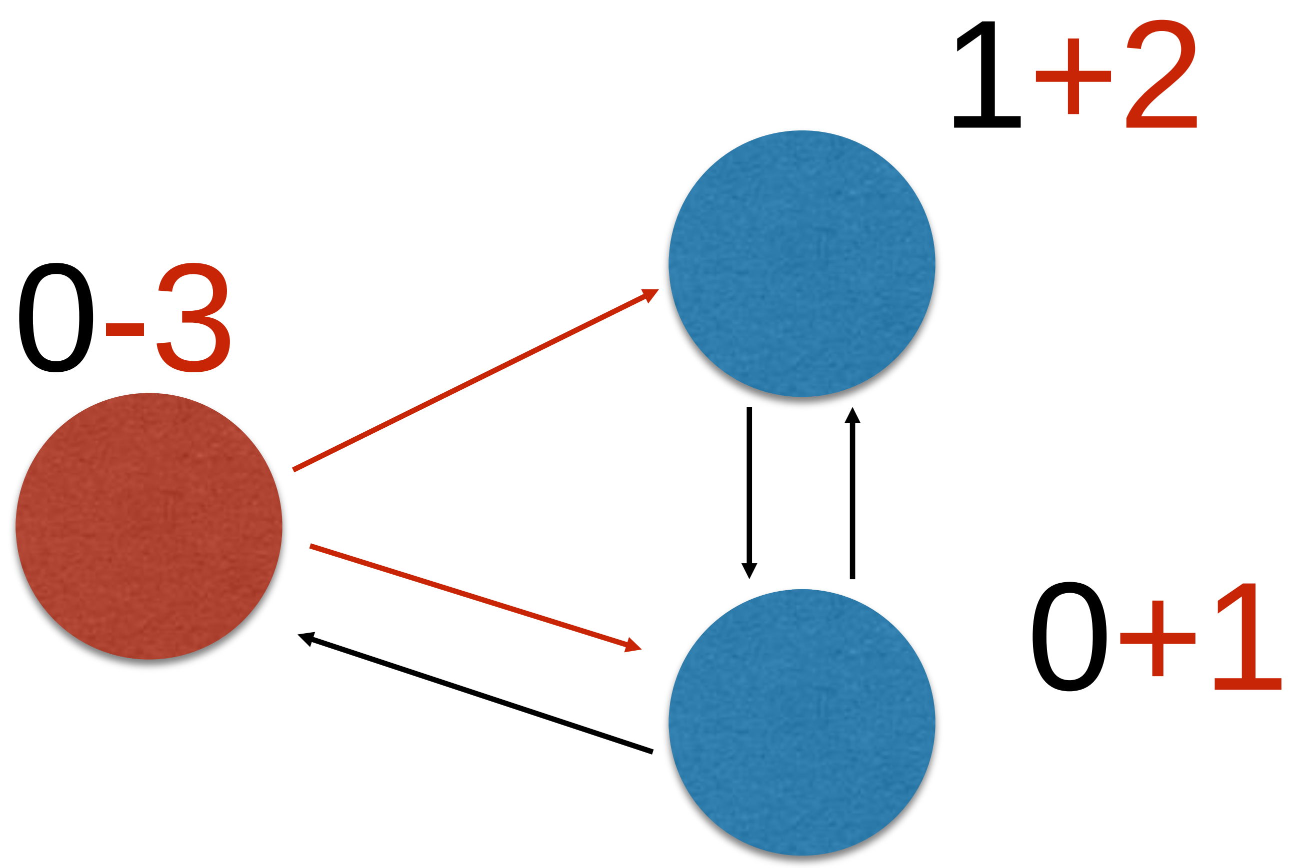
RandomWalk: Algorithm

1. **Initialize each node with some coupons**
2. Randomly forward coupons to connected nodes with small probability to stop
3. Repeat 2 until no more coupons
4. Count the total visit to each node

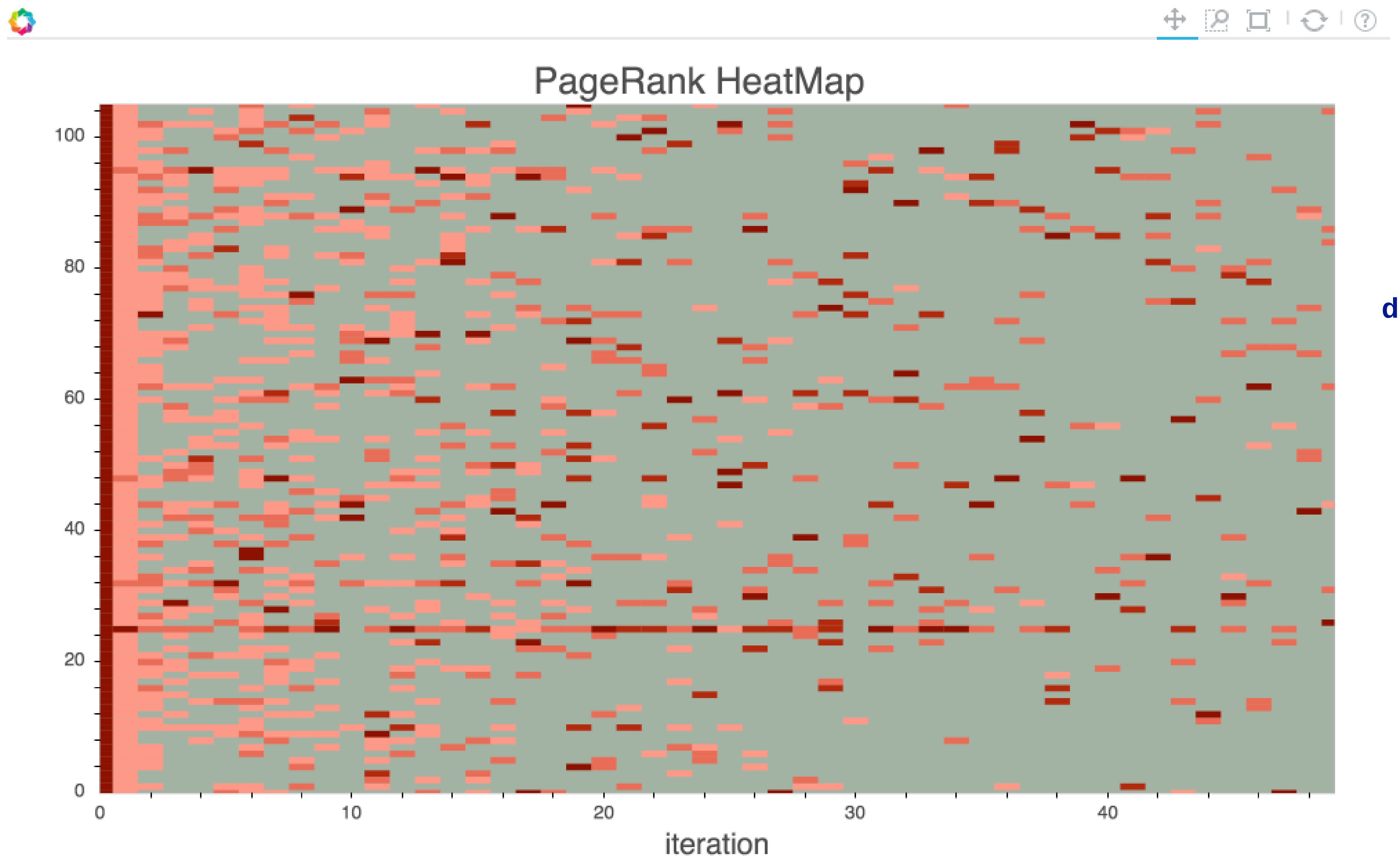


RandomWalk: Algorithm

1. Initialize each node with some coupons
2. **Randomly forward coupons to connected nodes with small probability to stop**
3. Repeat 2 until no more coupons
4. Count the total visit to each node



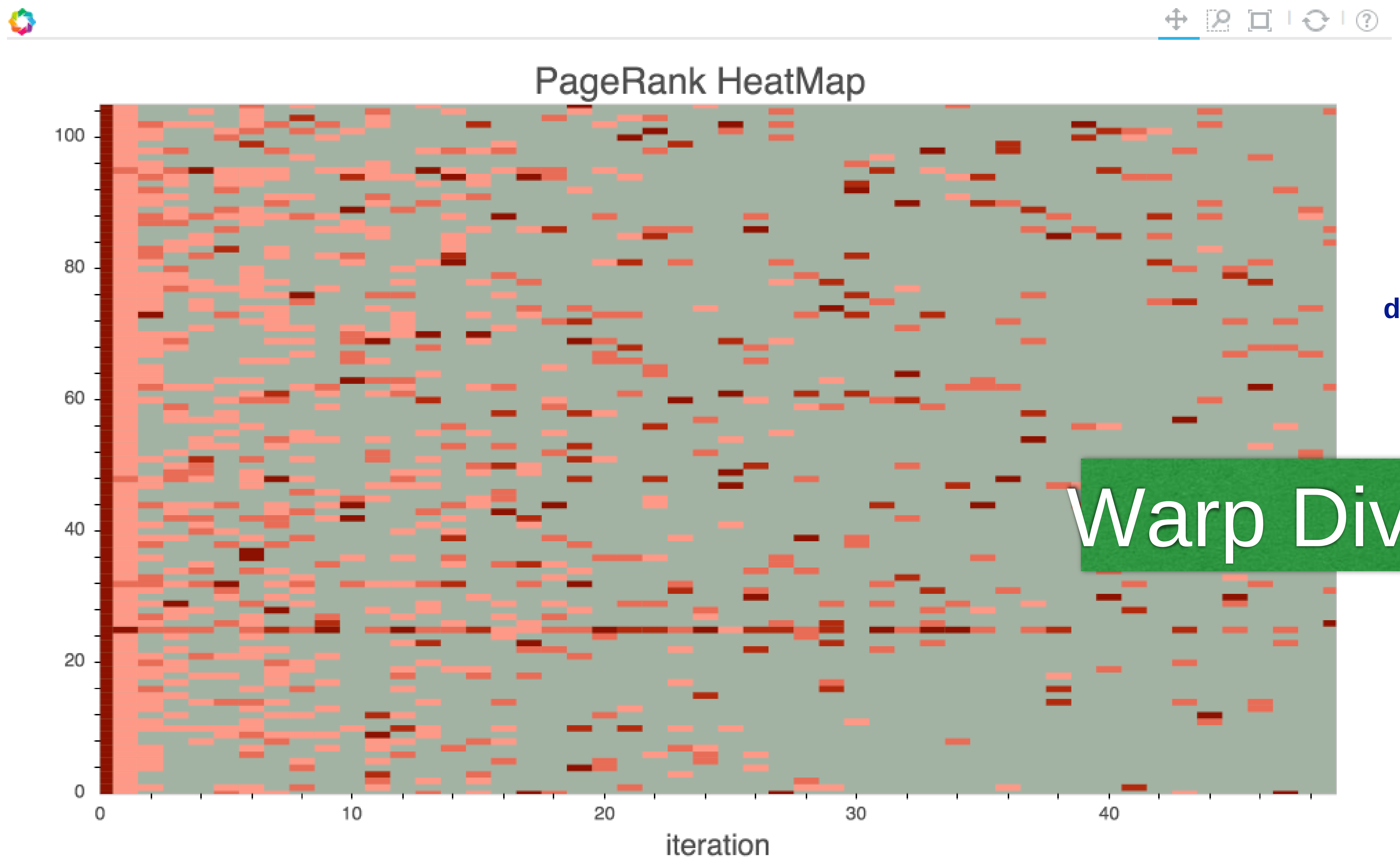
RandomWalk: Visualize the Algorithm



Plot coupon count at each iteration
with Bokeh

```
def plot_heatmap(fig, couponmap, colormap):
    numnodes = couponmap.shape[1]
    for frameid in range(couponmap.shape[0]):
        coupons = couponmap[frameid]
        max_coupon = coupons.max()
        min_coupon = coupons.min()
        drange = (max_coupon - min_coupon)
        if drange == 0:
            normalized = np.ones_like(coupons)
        else:
            normalized = (coupons - min_coupon) / drange
        csels = np.round(normalized * (len(colormap) - 1))
        visibles = (coupons != 0).astype(np.float32)

        colors = [colormap[x] for x in csels.astype(np.int32)]
        fig.rect([frameid] * numnodes, np.arange(numnodes), 1, 1, color=colors,
                 alpha=visibles, line_width=0, line_alpha=0)

    fig.grid.grid_line_color = None
    fig.axis.axis_line_color = None
    fig.axis.major_tick_line_color = None
    fig.xaxis.axis_label = "iteration"
    fig.yaxis.axis_label = "blockIdx"
```

RandomWalk: Visualize the Algorithm



Plot coupon count at each iteration
with Bokeh

Warp Divergence

```
def plot_heatmap(fig, couponmap, colormap):
    numnodes = couponmap.shape[1]
    for frameid in range(couponmap.shape[0]):
        coupons = couponmap[frameid]
        max_coupon = coupons.max()
                      ons.min()
                       on - min_coupon)

            normalized = np.ones_like(coupons)
        else:
            normalized = (coupons - min_coupon) / drange
        csels = np.round(normalized * (len(colormap) - 1))
        visibles = (coupons != 0).astype(np.float32)

        colors = [colormap[x] for x in csels.astype(np.int32)]
        fig.rect([frameid] * numnodes, np.arange(numnodes), 1, 1, color=colors,
                 alpha=visibles, line_width=0, line_alpha=0)

    fig.grid.grid_line_color = None
    fig.axis.axis_line_color = None
    fig.axis.major_tick_line_color = None
    fig.xaxis.axis_label = "iteration"
    fig.yaxis.axis_label = "blockIdx"
```

Fixing Warp Divergence 1

- Reference Redirection
- Remap threadIdx to workitems
  - workitem = remap[threadIdx]
- Sort indices
  - Nodes with highest number of coupons go first

```
from numbapro.cudalib import sorting
…
sorter = sorting.RadixSort(maxcount=d_remap.size, dtype=d_visits.dtype,
                           descending=True)
…
sorter.sort(keys=d_visits_tmp, vals=d_remap)
```

Reference:
Zhang, Eddy Z., et al. "Streamlining GPU applications on the fly: thread divergence elimination through runtim

Fixing Warp Divergence 2

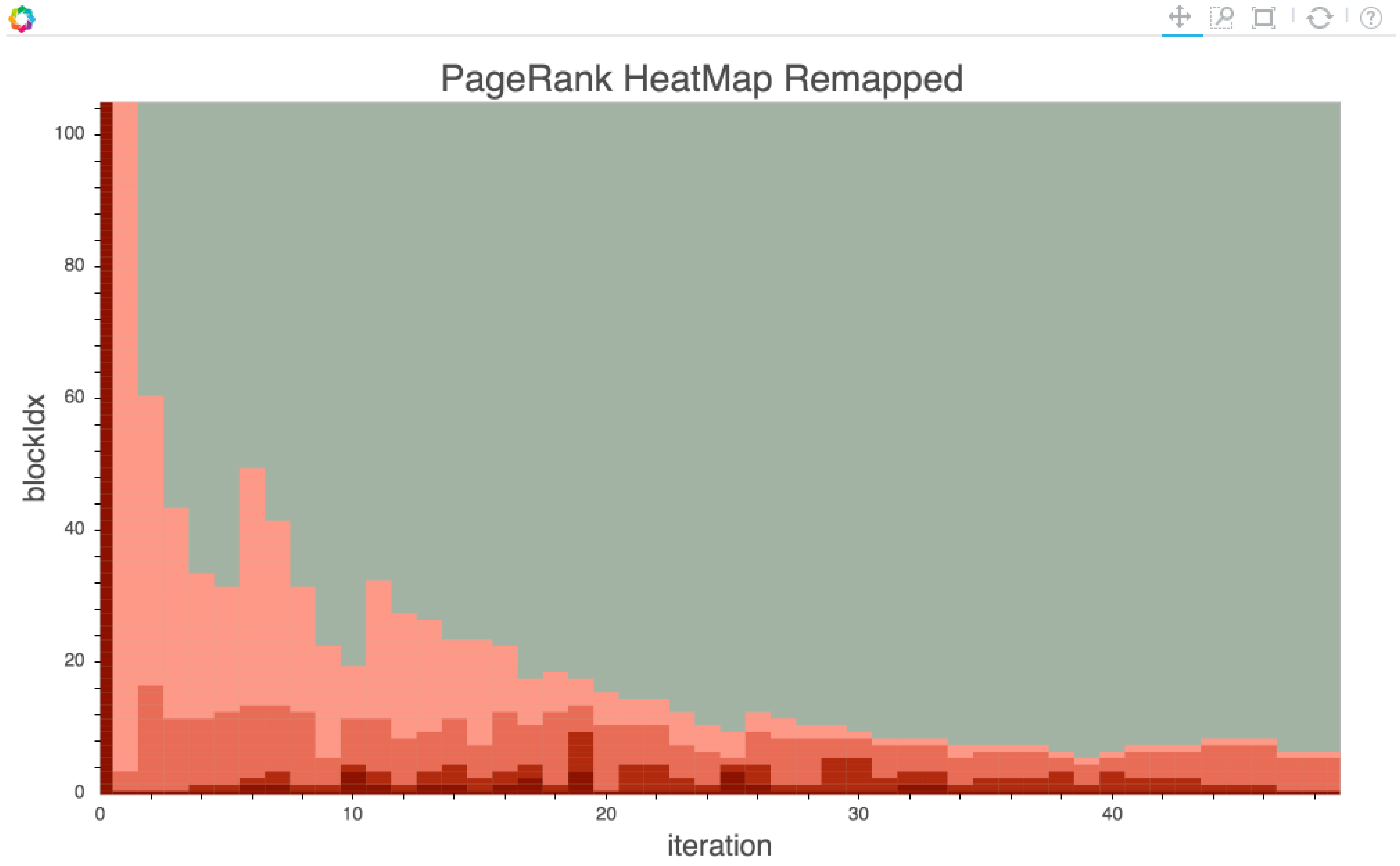
- Dynamic scheduling at block level
- Blocks assign each thread to handle one coupon

```
@cuda.jit
def cuda_random_walk_round(coupons, visits, colidx, edges, resetprob,
                           randstates, remap):
    sm_randstates = cuda.shared.array(MAX_TPB, dtype=uint64)

    tid = cuda.threadIdx.x
    blkid = cuda.blockIdx.x

    if blkid < coupons.size:
        workitem = remap[blkid]
        sm_randstates[tid] = cuda_random_get_state(tid, randstates[workitem])
        count = coupons[workitem]

        # While there are coupons
        while count > 0:
              y to assign coupons to every thread in the block
              gned = min(count, cuda.blockDim.x)
            count -= assigned
            # Thread within assigned range
               d < assigned:
                 da_random_walk_per_node(workitem, visits, colidx, edges,
                                         resetprob, sm_randstates)
```

Assign coupons to threads

Assigned thread work on a coupon

Optimized Result



Reference Redirection	Block Scheduling	Time
Y	N	DNF
N	Y	1137
Y	**Y**	**163**

Bilinear optimization with Spannogram

Core computation:

- Generate random samples
- Apply to eigenvectors
- Find k-best result
- Repeat

Bilinear optimization with Spannogram

Core computation:

- Generate random samples
- Apply to eigenvectors
- Find k-best result
- Repeat

```
from numbapro.cudalib import cublas
…
blas = cublas.Blas()
…
blas.gemm('N', 'N', m, n, k, 1.0, d_V, d_c, 0.0, d_Vc)
```

Bilinear optimization with Spannogram

Core computation:

- Generate random samples
- Apply to eigenvectors
- Find k-best result
- Repeat

```
from numbapro.cudalib import sorting

sorter = sorting.RadixSort(maxcount=config.NUMNODES,
                           dtype=np.float32,
                           descending=True)
…
topk_idx = sorter.argselect(k=k, keys=Vc)
```

RadixSort.argselect(k, keys)

Returns the indices of the first k elements from the sorted sequence.

>90% of the time in CPU implementation

Bilinear optimization with Spannogram

Core computation:

- Generate random samples
- Apply to eigenvectors
- Find k-best result
- Repeat

Sorting 42 million floats 16350 times

Visualizing the DkS

- Bokeh generates beautiful interactive plots
- But lacks graph layout
- Just write it a simple spring-layout in Python
- Speed it up with Numba on CPU

Simple Spring Layout

```
@jit
def _force_by_dist(dist, optdist, connected):
    if dist < optdist:
        # Repluse
        return 20 * (optdist - dist)

    elif dist > optdist:
        # Attract
        if connected:
            return -(dist - optdist) * 0.1
        else:
            return -(dist - optdist) * 0.01
    else:
        return 0

@jit
def _calc_force_map(xs, ys, mass, edges, fxmap, fymap, num):
    for i in range(num):
        for j in range(num):
            if i != j:
                optdist = (mass[i] + mass[j]) / 1.5
                dx = xs[j] - xs[i]
                dy = ys[j] - ys[i]
                dist = math.sqrt(dx ** 2 + dy ** 2)

                force = _force_by_dist(dist, optdist, edges[i, j])

                if dist == 0:
                    dist += 1e-30
                cosine = dx / dist
                sine = dy / dist

                fxmap[i, j] = force * cosine
                fymap[i, j] = force * sine
```

```
@jit
def _sum_forces(fxmap, fymap, fxtotal, fytotal, num):
    for j in range(num):
        fxtotal[j] = 0
        fytotal[j] = 0
        for i in range(num):
            fxtotal[j] += fxmap[i, j] / num
            fytotal[j] += fymap[i, j] / num

def spring_fit(xs, ys, edges, mass, iterations=10 ** 5 * 2):
    num = len(xs)

    fxmap = np.zeros((num, num), dtype=np.float32)
    fymap = np.zeros((num, num), dtype=np.float32)

    fxtotal = np.zeros(num, dtype=np.float32)
    fytotal = np.zeros(num, dtype=np.float32)

    STEP = 300

    for _count_it in range(iterations):
        _calc_force_map(xs, ys, mass, edges, fxmap, fymap, num)
        _sum_forces(fxmap, fymap, fxtotal, fytotal, num)

        # Stop if maximum movement is less than one
        if np.abs(fxtotal).max() < 1 and np.abs(fytotal).max() < 1:
            return

        # Apply forces
        dfx = np.clip(fxtotal, -STEP, +STEP)
        dfy = np.clip(fytotal, -STEP, +STEP)

        xs += dfx
        ys += dfy
```

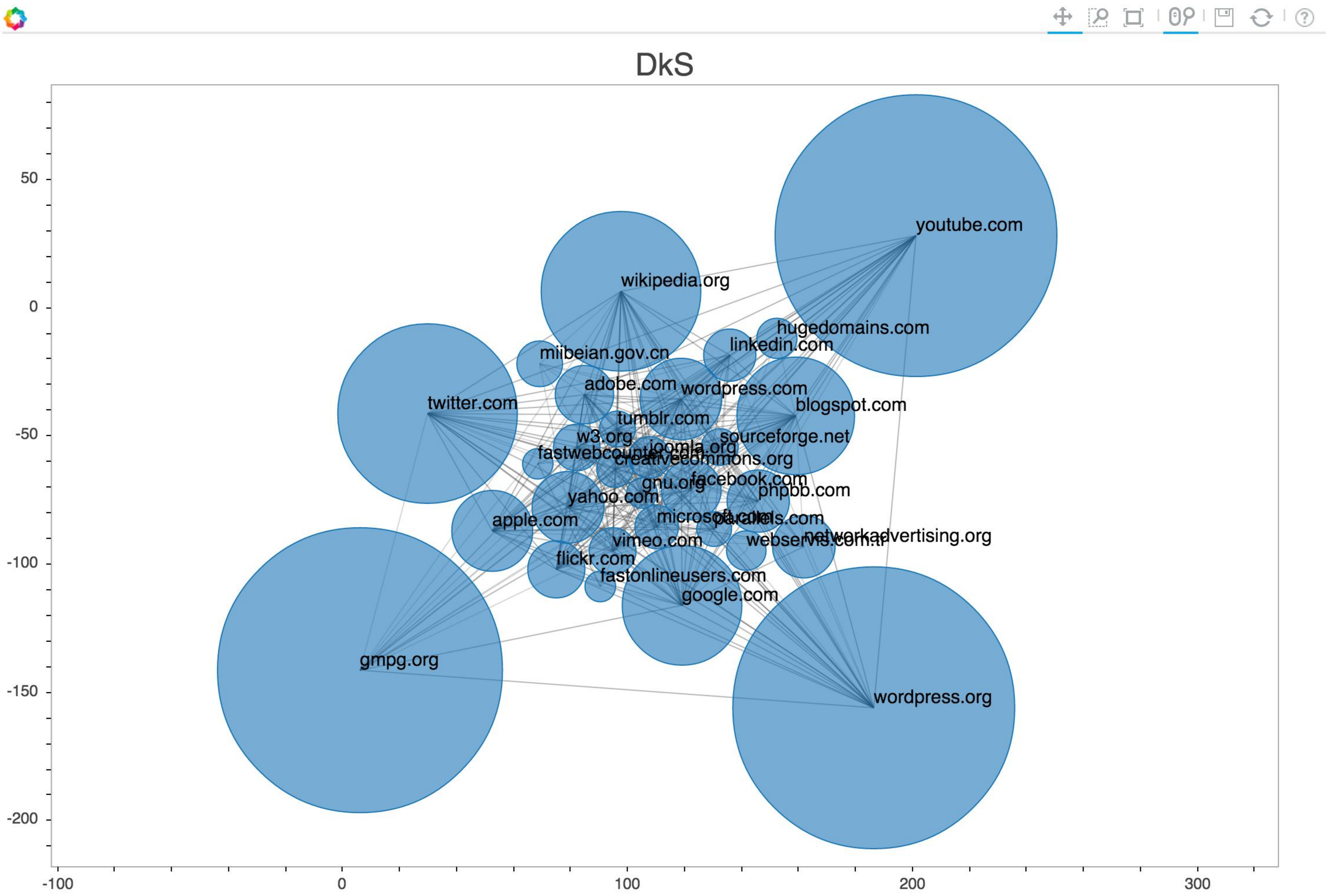
Simple Spring Layout

```
@jit
def _force_by_dist(dist, optdist, connected):
    if dist < optdist:
        # Repluse
        return 20 * (optdist - dist)

    elif dist > optdist:
        # Attract
        if connected:
            return -(dist - optdist) * 0.1
        else:
            return -(dist - optdist) * 0.01
    else:
        return 0

@jit
def _calc_force_map(xs, ys, mass, edges, fxmap, fymap, num):
    for i in range(num):
        for j in range(num):
            if i != j:
                optdist = (mass[i] + mass[j]) / 1.5
                dx = xs[j] - xs[i]
                dy = ys[j] - ys[i]
                dist = math.sqrt(dx ** 2 + dy ** 2)

                force = _force_by_dist(dist, optdist, edges[i, j])

                if dist == 0:
                    dist += 1e-30
                cosine = dx / dist
                sine = dy / dist

                fxmap[i, j] = force * cosine
                fymap[i, j] = force * sine
```

JIT: 20s

Interpreted: >15minutes

```
@jit
def _sum_forces(fxmap, fymap, fxtotal, fytotal, num):
    for j in range(num):
        fxtotal[j] = 0
        fytotal[j] = 0
        for i in range(num):
            fxtotal[j] += fxmap[i, j] / num
            fytotal[j] += fymap[i, j] / num

def spring_fit(xs, ys, edges, mass, iterations=10 ** 5 * 2):
    num = len(xs)

), dtype=np.float32)
), dtype=np.float32)

=np.float32)
=np.float32)

    STEP = 300

    for _count_it in range(iterations):
        _calc_force_map(xs, ys, mass, edges, fxmap, fymap, num)
        _sum_forces(fxmap, fymap, fxtotal, fytotal, num)

        # Stop if maximum movement is less than one
        if np.abs(fxtotal).max() < 1 and np.abs(fytotal).max() < 1:
            return

        # Apply forces
        dfx = np.clip(fxtotal, -STEP, +STEP)
        dfy = np.clip(fytotal, -STEP, +STEP)

        xs += dfx
        ys += dfy
```



Total Time: 35 minutes
PageRank (2eigs): 178s
Spannogram (rank1): 26s
Spannogram (rank2): 30min

Acknowledgement

Dr. Alex Dimakis and his team at UT Austin for discussions on their DkS approximation algorithm.

Thank You

Questions?

email: siu@continuum.io

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!

CONTINUUM ANALYTICS